UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Implementación de una Aplicación en la Nube para Desarrollo de Software

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A N

Antonio de Jesús Lozano Arriaga

Ken Ruiz Inoue

DIRECTORA DE TESIS

M.C. Ma. Jaquelina López Barrientos

Ciudad Universitaria, Cd. Mx., 2017

Agradecimientos

Para iniciar quisiera agradecer a la Universidad Nacional Autónoma de México, la máxima casa de estudios que me brindó la oportunidad de cursar mis estudios superiores, además de permitirme desarrollarme no solo en un entorno profesional y académico, sino como una mejor persona que busca servir a su país; no hay mayor orgullo que decir pertenezco a la UNAM.

También agradezco a la Facultad de Ingeniería, mi amada facultad, en ella llegué a conocer a muchas personas que me ayudaron a tener una mejor perspectiva del mundo, agradezco que me permitiera conocer compañeros, ingenieros, maestros y doctores que comparten el mismo amor que yo tengo a esta gran institución, ya que siempre me brindaron sus conocimientos de la mejor forma posible para así entender que la Ingeniería va más allá de números y ecuaciones, sino que también implica un valor moral y ético con la sociedad, la naturaleza y el país.

A su vez quisiera agradecer a mi madre Leticia Arriaga y a mi padre Celerino Lozano que los dos fueron, son y serán parte fundamental en mi vida; es gracias a ellos que pude concluir una meta que fijé desde pequeño, terminar mis estudios, ellos me enseñaron la importancia de los valores y la educación, sin el apollo de los dos no sería el hombre que soy ahora, siempre estaré eternamente agradecido con ellos por toda la dedicación, cariño y amor que me han demostrado y no encuentro mejor forma de agradecerles todo su esfuerzo que plasmar sus nombres en este documento que es de suma importancia en mi vida. Gracias por todo.

Por último y no menos importante quisiera agradecer a M.C. Ma. Jaquelina López Barrientos directora de tesis, que brindó todo su apoyo con toda devoción al presente proyecto, maestra que tuve la oportunidad de conocer en la materia de Redes de Datos, excelente maestra y ser humano, una de las mejores ya que sus acciones van más allá de solo las que le corresponden, siempre alentando a los alumnos a titularse, siempre viendo por el bienestar del alumno.

Lozano Arriaga Antonio de Jesús

“POR MI RAZA HABLARÁ EL ESPIRITU”

Agradecimientos

Agradezco a mi padre José René Ruiz Gutiérrez y a mi madre Mayumi Yoneta que me dieron lugar en este universo, que me apoyaron en cualquier circunstancia para poder apoyar, aprender y amar en esta valiosa vida. El apoyo y amor que he recibido y seguiré recibiendo de mis padres, es algo imprescindible que me ha permitido fijar metas y objetivos para alcanzarlos. Por eso, quiero dedicarles un agradecimiento por apoyarme a lograr esta meta. Muchas gracias.

También les doy las gracias a mis abuelos maternos por su constante apoyo y amor que me han dado. Siempre me han recibido con un corazón cálido y nunca olvidaré todo el amor, apoyo y comida que he recibido de ustedes.

Mi padre y mi abuelo materno siempre han estado inspirados en enseñarme a estudiar, pensar y analizar las cosas. Gracias a su interés hacia la ingeniería y la ciencia, me animé a estudiar esta carrera. Gracias por confiar en mis capacidades y apoyarme en todo momento con mis estudios.

Le quiero dar las gracias a mi mejor amigo Luis Ricardo Martínez Ramírez y a su familia por siempre tratarme como parte de su familia. Nunca pude imaginar poder formar un vínculo tan fuerte con personas que no fueran de mi familia, pero ellos me enseñaron la importancia de una verdadera amistad. Gracias por todo el apoyo y cariño que me han brindado para terminar esta carrera.

Gracias a mi pareja Aracne Rodríguez Reyes por estar a mi lado en los momentos difíciles e importantes de mi vida. Siempre valoro el apoyo y el cariño que me ofreces sin importar la circunstancia. Sé que siempre estarás a mi lado para apoyarme en todo momento.

Este agradecimiento también va dirigido a mi asesora de tesis a M.C. Ma. Jaquelina López Barrientos por todo el apoyo y tiempo que ha dedicado para el presente documento. Estoy muy contento y agradecido al haber tenido a la maestra como directora de tesis. Ella siempre confía en la capacidad de los alumnos y brinda todo su apoyo y conocimiento para ayudarlos a formarse profesionalmente.

Finalmente quiero agradecer a la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería que me ha permitido a conocer lo hermoso y sofisticado que es la ingeniería, además de enseñarme a pensar y actuar como ingeniero. Gracias a esta gran institución también pude conocer amigos y maestros que marcaron mi vida. Estoy orgulloso y agradecido al haber estudiado en una de las mejores universidades del mundo.

Ruiz Inoue Ken

“POR MI RAZA HABLARÁ EL ESPIRITU”

Índice Temático

Índice de Tablas

Capítulo 1 Análisis del Sistema

Capítulo 2 Diseño del Sistema

Capítulo 3 Desarrollo del Sistema

Capítulo 4 Pruebas del Sistema

Índice de Figuras

Capítulo 4 Pruebas del Sistema

Introducción

En la actualidad los dispositivos electrónicos y los sistemas informáticos son indispensables en el campo laboral, industrial e incluso en el de entretenimiento. Es tan fuerte la dependencia a nivel social, que un mal funcionamiento de éstos, puede causar golpes económicos graves. Los dispositivos electrónicos que se usan a diario como smartphones y laptops se componen principalmente en la parte física de hardware que es el conjunto de los componentes electrónicos y en la lógica de software, que hace referencia al conjunto de programas, instrucciones o reglas informáticas.

La teoría de software surge en el año 1935 por Alan Turing en su ensayo llamado "Números Computables" con una aplicación capaz de tomar decisiones, pero hasta el año 1958 es que se utiliza por primera vez el término "Software" por John W. Tukey. Este software se distribuía en el hardware que las grandes compañías ofrecían, era libre, no tenía ataduras a términos de condiciones y uso, y poco a poco las personas involucradas en la creación de computadoras se adentraban cada vez más al área del software; a finales de la década de los años 70 las grandes empresas empezaron a vender sus programas de forma adicional y obligaba a los usuarios a cumplir con un contrato, lo que conllevó a que el sistema de distribución de software sufriera un cambio.

Sin embargo, en el año de 1984 Richard Stallman inició un proyecto llamado GNU, y un año más tarde fundó la Free Software Foundation (FSF). Fue él quien introdujo el concepto de software libre, este concepto permitía a los usuarios modificar el software a su disposición y necesidades.

Conforme el tiempo pasaba, el campo del software crecía con gran consideración, mientras que el del hardware se reducía considerablemente. A su vez se volvía más rápido y su precio era menor. Fue cuando el software se volvió un negocio y cada día más programadores se adentraron a este campo, esto fue lo que generó una nueva tendencia; y la demanda de software creció [1].

Con esta nueva tendencia, aumentó la necesidad de desarrollar más software y con más velocidad para satisfacer la demanda de los clientes. Un buen indicador que señala la aceleración de producción de software, está relacionado con las aplicaciones móviles de Google Play, una tienda virtual que ofrece la distribución de éstas para dispositivos con sistema operativo Android. Google Play tuvo su puesta en el mercado en agosto de 2008. Para julio de 2013, Sundar Pichai, responsable de Android, anunció que se superó 1 millón de aplicaciones en el catálogo de Google Play [2]. Se puede decir que se desarrollaron 1 millón de aplicaciones móviles en un periodo de 47 meses, siendo un total de 21,276 aplicaciones por mes. Actualmente se tiene un aproximado de 1,995,900 aplicaciones disponibles en la tienda, lo cual significa que se alcanzó una cifra de 995,900 desarrolladas en un periodo de 31 meses, un total de 32,125 por mes. El desarrollo promedio mensual de aplicaciones móviles para Android en el primer periodo mencionado comparado al segundo, incrementó en un 51% [3].

Como se puede observar, en la actualidad, las organizaciones o empresas buscan desarrollar software con una velocidad superior para poder adaptarse a las necesidades de la sociedad y sus clientes. Pero el desarrollo de software ha enfrentado carencias de eficiencia en estas

últimas décadas, etapa conocida como “la crisis del software”. La crisis de software es un problema que se da por la falta de productividad en su proceso de elaboración, este no ha alcanzado niveles comparables con otras tecnologías como lo ha hecho por ejemplo el hardware que tiene más antigüedad. Además de su gran complejidad en perspectiva con otros productos, lo que al comparar la calidad del mismo lo pone en cierta desventaja [4].

Esta crisis surgió en la década de los años 80 y fue debido a la revolución de las nuevas computadoras, ya que eran más potentes, así, el campo de las aplicaciones o desarrollo de software creció de manera impresionante, esto gracias a que ahora podían realizarse programas que anteriormente no se podían debido a las pobres características del hardware. La magnitud fue más grande y compleja, por lo que seguir manejando el desarrollo de los mismos como se hacía antes (con un enfoque informal) no fue el mejor, ya que entre mayor fuera el proyecto mayor era el retraso de entrega y mayor el costo de producción a lo presupuestado, por esto los proyectos de software se volvieron irrealizables, difíciles de mantener y con un bajo desempeño [5].

Es imprescindible hablar de la crisis del software porque debido a este problema surgió una necesidad que obligó a los desarrolladores a crear técnicas, métodos y sistemas que pudiesen controlar la complejidad inherente de los sistemas; un gran paso para el desarrollo de software ya que se tomaron medidas más eficientes, lo que permite hoy en día realizar un desarrollo de software más organizado y profesional. Es importante mencionar que entre mayor sea la demanda de los sistemas en cuanto a su complejidad y volumen, igual será la demanda de crear técnicas más eficientes para reducir los problemas lo más posible [5].

Las metodologías que ayudaron a mitigar la crisis del software son las llamadas metodologías ágiles. Las cuales surgieron en febrero de 2001, con el objetivo de esbozar valores y principios que permitieran a los desarrolladores crear software de una manera rápida que principalmente les permitiera responder adecuadamente a los cambios que se puedan presentar, ya que las metodologías existentes contemplaban una administración más rígida y menos susceptible al cambio por la documentación que requerían.

El software cambiaba con gran consideración y a gran velocidad, es por esto que se necesitaban metodologías rápidas y accesibles al cambio, metodologías que cumplieran con los siguientes principios:

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
7. El software que funciona es la medida principal de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento [6].

Dichas metodologías brindan una programación más eficaz y rápida, que apoyan a la demanda de software, pero las metodologías ágiles no han eliminado en su totalidad los problemas existentes al momento de desarrollar software, sin duda alguna los han disminuido, es por esto la importancia de generar metodologías cada vez más eficientes que puedan responder a las nuevas demandas que el desarrollo de software necesita día con día con el fin de mitigar las antiguas falencias [7].

Hoy día se disponen de varias herramientas para facilitar la implementación del software. Si los integrantes del equipo se tienen que comunicar, organizar o administrar, existen herramientas famosas como Trello, Slack o Jira que lo permiten de una manera sencilla e interactiva. Si se busca desarrollar sobre el mismo proyecto entre varios desarrolladores, existen los sistemas de control de versiones como Git. Incluso existen herramientas que se adaptan perfectamente a los marcos de trabajos sugeridos por las metodologías ágiles.

Si uno de los retos en la actualidad es acelerar la productividad del desarrollo de software, estar capacitando a los integrantes con todas las herramientas que se vayan a utilizar, puede llegar a ser laborioso y ser una causa para la demora en el desarrollo de software. En el caso de una organización con años de experiencia, es altamente probable que tenga implementados sus estándares y normas para desarrollar software, pero ¿qué sería de un equipo que apenas surgió o una compañía recién formada que busca evaluar cómo definir sus estándares? Es evidente que se vuelve indispensable invertir en tiempo adicional para capacitar a los nuevos desarrolladores, además de determinar las metodologías adecuadas para el desarrollo y todo esto antes de iniciar formalmente el proyecto. En el mejor de los casos puede que se haga la correcta selección de herramientas y marco de trabajo, pero en el caso de no acertar, sería volver a consumir tiempo en el campo empírico.

Un paradigma que ha surgido en esta última década es el cloud computing, también conocido como la “nube”. Este paradigma consiste en ofrecer servicios de computación a través de la red, el software como servicio, uno de los servicios que caracteriza a cloud computing, ofrece aplicaciones y software de diferente índole a los clientes a través de los navegadores web evitando la necesidad de instalar una aplicación específica para su uso y permitiendo el consumo del servicio desde cualquier ordenador con un navegador web. Este paradigma ha

reducido los costos en las organizaciones ya que no requieren un ordenador y sistema operativo en específico para cada usuario que se tiene empleado en la organización [8].

Una de las soluciones satisfactorias en el ámbito de desarrollo de software, consiste en la disminución del tiempo y costo. Entonces la pregunta es: ¿cómo podría lucir una solución que reduzca el tiempo de desarrollo y su costo? Es evidente que la demanda para desarrollar software ha crecido y seguirá comportándose de esa manera, por lo que se vuelve indispensable crear una solución que integre respuestas que han funcionado en el recorrido de la evolución del desarrollo de software, esto es, reducir el uso de herramientas con la finalidad de ahorrar tiempo y costo para el desarrollo de software, contando con un ambiente de desarrollo en la nube que integre algunas herramientas de uso frecuente, así evitando la necesidad de instalar programas y aplicaciones en los ordenadores para permitir al usuario enfocarse únicamente en el desarrollo del software.

Objetivo General

Crear un sistema en la nube que permita a los programadores el desarrollo de software desde cualquier plataforma, sea una aplicación web, escritorio o móvil siempre y cuando a través de un navegador web. Y que las aplicaciones incluyan herramientas para permitir un desarrollo rápido y eficiente del software.

Objetivos Particulares

1. Proporcionar un servicio en la nube que permita evitar la instalación de programas ajenos para desarrollar software, así permitir al usuario enfocarse únicamente en el desarrollo de código y no en la preparación de la infraestructura, además desde cualquier plataforma.
2. Facilitar la comunicación entre los diferentes integrantes o desarrolladores de un proyecto a través de un canal de comunicación y así, reducir al mínimo problemas en cuanto al desarrollo se refiere.
3. Proporcionar plantillas de código de uso común a los usuarios del proyecto.

Capítulo 1

Análisis del Sistema

1.1 Reconocimiento del Problema y su Solución

Antes de dar inicio a cualquier elaboración de una solución en la presente tesis, es de gran importancia identificar los principales elementos que conforman el problema a solucionar de una manera detallada, para así, facilitar la ejecución de la solución de dicha problemática.

Con problema se entiende un hecho o circunstancia que impide el bienestar, un estado negativo que requiere una solución debido a que su existencia perjudica a un sector. El problema considerado en esta tesis es la ineficiencia que se llega a presentar en el transcurso de un proyecto de desarrollo de software. La causa principal de este problema es la complejidad que persiste al desarrollar software con un propósito comercial en específico. El software comercial es distinto a los personales, los cuales se pueden desarrollar sin necesidad de establecer un marco de trabajo, contrato legal, ni un lineamiento de entrega, ya que éstos son generalmente desarrollados para fines no lucrativos sino para fines personales. Pero en el caso de un software comercial, generalmente tiene fines lucrativos, estos están basados en régimen de licencias comerciales propietarias buscando un beneficio entre ambas partes, entre el cliente y el proveedor [9], generando la necesidad de trabajar sobre estándares y normas fundamentadas para un proyecto con un propósito comercial.

La complejidad que se llega a presentar en el transcurso de un proyecto de software comercial, es debido a la necesidad y obligación de entregar la solución en el tiempo establecido. La ejecución de un proyecto requiere de administración, instalación de equipos, preparación de infraestrucutra y desarrolladores. Cada vez se vuelve más complicado coordinar todos los elementos de un proyecto debido al crecimiento y avance de la tecnología y la demanda de desarrollo del software en menos tiempo por parte del cliente, lo cual se convierte en un problema.

Esta problemática genera una crisis y para resolver cualquier crisis son necesarios fuertes cambios; para poder lograr dichos cambios surge la Ingeniería del software que amplía la visión del desarrollo del mismo como una actividad esencialmente de programación, sino también contemplando otras áreas o actividades como la de análisis y diseños previos, y de integración y verificaciones posteriores. La distribución de las actividades mencionadas generó algo conocido como el ciclo de vida del desarrollo del software.

Los modelos clásicos del ciclo de vida muestran que el desarrollo de programas se debe llevar a través de una secuencia de actividades sucesivas y diferentes; algunos ejemplos son el modelo en cascada y el modelo en “V” ambos han brindado una gran solución a dicha crisis pero ésta aún existe, debido a que muchos de estos modelos están orientados hacia un desarrollo lineal, definiendo un inicio y fin de cada fase, lo que no los hace capaces de manejar de la mejor manera los errores, ya que una vez terminada una fase los recursos humanos y materiales se enfocan a otra fase y complica la situación para poder retomarla [10] y la creación de software demanda en gran medida el poder retomar antiguas fases, esto debido a errores en la programación.

Esto genera un atraso en tiempo y por ende un incremento monetario al estipulado inicialmente, en un software comercial esto es muy delicado ya que se desarrolla con fines de lucro y si el desarrollo de un software es igual o más costoso que el precio de venta deja de ser beneficioso para quienes lo desarrollan y pierde el concepto de negocio. Para evitar esto se crearon modelos evolutivos [10], sin embargo, no presentaron la solución esperada por lo que también se decidió combinar los modelos para obtener mejores resultados.

A pesar de que dichas soluciones han ayudado a deteriorar la crisis que se sufre, no han logrado erradicarla por completo. Lamentablemente al tratar de solucionar dicha crisis las soluciones han contribuido de cierta manera y han empeorado la situación, esto es debido a la existencia de una cantidad abundante de herramientas para realizar una adecuada administración, tanto las organizaciones como programadores no siguen un estándar, sino que utilizan las herramientas y tecnologías que más les acomoden ya que no es viable que se especialicen en todas, por cuestión de tiempo y nivel de conocimiento que puedan adquirir de cada una de ellas.

Al haber tanta diversidad de herramientas, los proyectos de software se vuelven desorganizados, costosos y lentos, ya que el personal involucrado tiene que capacitarse en cada una de las herramientas que fueron escogidas para la implementación de un proyecto de software cualquiera.

La solución que se propone en esta tesis para mitigar el problema mencionado en la introducción, es el de usar una herramienta homogeneizada que permita el desarrollo y administración de proyectos de software de una manera sencilla, versátil y dinámica. Para lograr esto se realizó un análisis de diferentes aspectos al desarrollar un software, con el fin de obtener las cualidades más fuertes o mejores de cada una de ellas y unificarlas para conseguir el cumplimiento de dicha meta.

Se tienen 2 propuestas de valor principales para el proyecto que se desea desarrollar en esta tesis. El primer valor agregado es un ambiente de desarrollo dinámico alojado en la nube, permitiendo a todo usuario la participación en el desarrollo desde cualquier lugar y momento para la consulta del mismo a través de dispositivos móviles o computadoras personales. Además, se omitiría el proceso de instalación de la infraestructura en los equipos, lo cual se traduce en reducción de timpo y costo en un proyecto. El segundo valor agregado es la integración de herramientas de colaboración en el ambiente de desarrollo mencionado en el prime valor, haciendo posible un desarrollo cómodo y versátil con dichas herramientas.

Dichas propuestas de valor se realizaron con el fin de hacer el sistema amigable y con el mayor alcance posible, para brindar una solución que sea de ayuda para los desarrolladores sin importar si el software a desarrollar sea pequeño, grande, con fines de lucro o software libre. Al no existir barreras para esta herramienta ayudará a que más empresas lo utilicen y así dar un paso más a la estandarización del desarrollo del software que es el primer y más importante paso a conseguir, ya que después de todo, los estándares han logrado conseguir que herramientas distintas trabajen en conjunto brindando una solución eficiente; un ejemplo de éxito muy importantes es el modelo OSI.

Para resumir, la solución consiste en una aplicación web que permiten almacenar información en la nube (proyectos de desarrollo de software), esto con el fin de que esté al alcance de todos los desarrolladores o participantes de un proyecto determinado, logrando así que la infraestructura de un equipo o distancia no sean un obstáculo para lograr la conclusión del desarrollo de software en su tiempo establecido.

La aplicación propuesta cuenta con un compilador alojado en un servidor que permite compilar código en el lenguaje java, junto con un entorno de conversación interno para los equipos de desarrollo que les permita estar siempre comunicados. También se cuenta con almacenamiento en la nube, permitiendo a los usuarios almacenar sus archivos de código fuente en el servidor a través de una base de datos y repositorio que almacenan los datos de usuario, para que los usuarios puedan acceder a sus proyectos y a su contenido a través de una cuenta desde el dispositivo y zona que ellos deseen.

1.2 Especificaciones y Requisitos del Sistema

La solución consiste en poner al alcance de los desarolladores una aplicación web, la cual brinde a estos una herramienta que integre funcionalidades necesarias para desarrollar proyectos de software sin necesidad de instalar un ambiente de desarrollo en una máquina local. La aplicación está conectada a una misma infraestructura (servidor), permitiendo de esta forma el almacenamiento de datos y archivos en la nube que ofrece un sistema estandarizado y unificado, y a su vez una mejor administración y disminución de tiempo en cuestión al desarrollo de un proyecto de cualquier índole de software y de manera consecuente, disminuir los costos de producción de los mismos.

En este capítulo se muestra una descripción a detalle de las funcionalidades con las que se cuenta el presente proyecto. Primero se describen las funcionalidades generales del sistema, las cuales son funcionalidades esenciales que se deben de encontrar en cada módulo de aplicación. Una vez mencionadas las funcionalidades generales, se indicarán las funcionalidades particulares de cada módulo, éstas son de carácter único, lo cual significa que nada más se podrá observar dicha funcionalidad en el módulo indicado. Finalmente, se desglosan las funciones que cubren la interfaz gráfica.

En la tabla 1.1 se pueden observar los requerimientos generales del sistema, ésto se debe de implementar en la aplicación web para permitir que el usuario tenga acceso a la herramienta que brinda la solución a la crisis mencionada en el inicio de este documento.

Tabla 1.1 Requerimientos Generales del Sistema

Requerimiento General	Descripción
RG1	Almacenar información del usuario y de proyectos en una base de datos.
RG2	Creación de proyectos.
RG3	Administración y edición de proyectos exixtentes.
RG4	Compartir proyecto con diferentes usuarios.
RG5	Modificar el contenido de los archivos de proyecto con el formato UTF-8.
RG6	Ejecución o compilación de proyectos generados por el usaurio.
RG7	Establecer conversaciones con diferentes usuarios a través de un chat.
RG8	Generación y edición de plantillas de código.

El canal de comunicación es un requerimiento que se encarga de la comunicación entre los colaboradores a través de una aplicación de mensajería. Esta solución permite mandar entre usuarios que comparten el mismo proyecto, para así facilitar la comunicación entre los colaboradores.

Para un ambiente de desarrollo de software, es indispensable la existencia de un editor de texto para poder insertar líneas de código en los archivos creados. Además, esta solución es capaz de marcar en italica y diferentes colores las palabras reservadas de Java para permitir al usuario distinguirlas. El editor de texto soporta UTF-8 como código de caracteres para la interpretación de los bytes.

El almacenamiento de los datos de usuario y los archivos que posee el usuario en su cuenta, son procesados a través de una interfaz lógica que administra la base de datos. Esta interfaz crea, elimina y actualiza los archivos solicitados por el usuario.

Sin el compilador de Java, no es posible llevar a cabo la ejecución de ningún programa, es obligatorio contar con un compilador en un ambiente de desarrollo de software para poder realizar pruebas o ejecuciones del código fuente.

El gestor de archivos es un requerimiento que cubre la necesidad de visualizar el contenido del proyecto generado por el usuario, colabora con la interfaz lógica de administración de base de datos para traducir la estructura interna del proyecto o archivos a una estructura visual para facilitar la consulta del proyecto. A través de una interfaz gráfica, el usuario puede darle instrucciones a la interfaz lógica de base de datos para organizar, crear o eliminar archivos como desee.

Con lo anterior se cubren los requerimientos generales que cuenta el sistema. Por la facilidad que otorga la máquina virtual de Java, se decidió soportar únicamente el lenguaje de Java para este proyecto. El ambiente de desarrollo elaborado puede compilar y ejecutar código de Java únicamente.

La aplicación será accesible desde cualquier dispositivo electrónico ya sea por computadora de escritorio, tableta hasta incluso smartphone, esto a través de un navegador web. El proceso de comunicación entre la aplicación y el servidor es como se muestra en la figura 1.1.

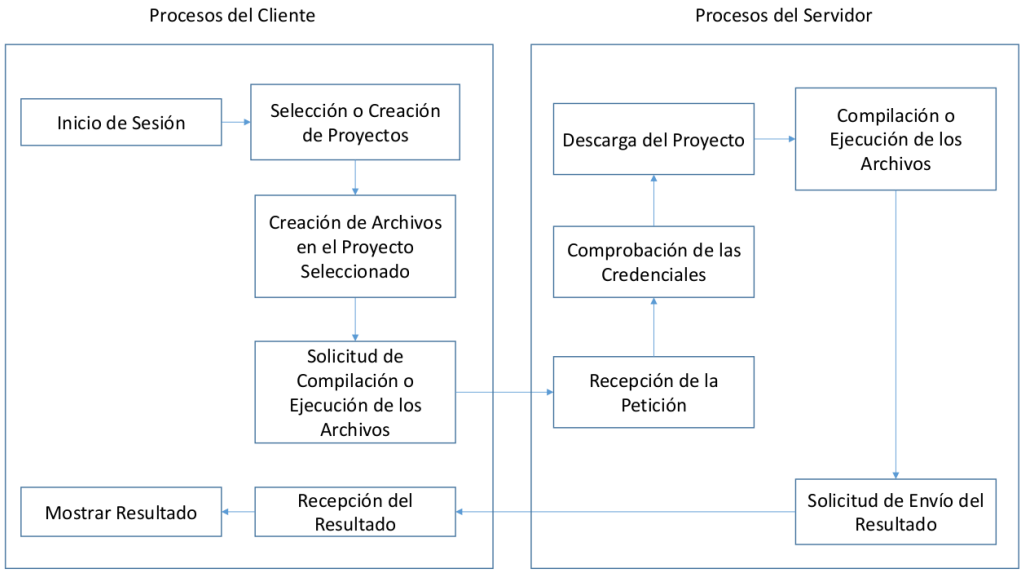

Figura 1.1 Proceso de Comunicación entre la Aplicación y el Servidor

Como se había mencionado, la aplicación web funciona estrictamente bajo el uso de un navegador, el cual requiere estar conectado a internet para operar con la aplicación web.

Para brindar la funcionalidad de compilación y ejecución de archivos de java, es necesario crear un servidor que ofrezca servicios web para recibir el proyecto y retornar el resultado obtenido.

A continuación, se muestran en la tabla 1.2 los servicios web que se consultarán a través de la aplicación web. Los servicios son consumidos al momento que el usuario compile y/o ejecute su código.

Tabla 1.2 Servicios Web

Nombre del servicio web	Entrada	Salida o resultado
compilar_codigo	Archivo de texto JSON con la información necesaria para la compilación	Retorna el resultado de la compilación
ejecutar_codigo	Archivo de texto JSON con la información necesaria para la ejecución	Retorna el resultado de la ejecución

Debido al empleo de la ejecución a través de un servicio web, no se consideran las ejecuciones de interfaces gráficas.

Para manejar y almacenar los datos a través de la aplicación, se debe de utilizar un conjunto de base de datos que cumpla con ciertas especificaciones.

La base de datos debe de cumplir con las especificaciones descritas en la tabla 1.3.

Tabla 1.3 Especificaciones de la Base de Datos

Especificaciones	Descripción de la especificación
Empleo del modelo relacional	Se emplea el modelo relacional con el fin de relacionar datos, y facilitar la consulta.
Disponibilidad de la base de datos	La base de datos funciona las 24 horas durante todos los días, permitiendo a los usuarios consumir el servicio en todo momento.
Accesibilidad a los datos	La información de la base de datos tiene que funcionar en todo momento igual que la disponibilidad de la base de datos.
Implementación de seguridad	La base de datos cuenta con un sistema de seguridad para evitar accesos no reconocidos a través de monitoreo y alertas.

Alojamiento remoto de la base de datos	La base de datos que se utiliza está alojado de manera remota, para permitir el acceso a través de la red.
Almacenamiento de la cuenta de usuario	El sistema es capaz de autenticar al usuario a través de un identificador único y contraseña, el sistema no permite 2 usuarios con el mismo identificador.
Almacenamiento de la información de proyectos del usuario	Todos los proyectos que genere el usuario en su cuenta, son almacenados en la base de datos del usuario para que posteriormente pueda consultar y modificar los proyectos generados.
Administración de la base de datos	La base de datos debe de poder ser administrada por cualquier escenario a través de una plataforma.
Interfaz para la base de datos	Ventana o canal lógico que permite la conexión de todas las aplicaciones hacia la base de datos.

Todas las especificaciones descritas en la tabla anterior, son de apoyo para realizar el diseño de la base de datos en el próximo capítulo y para escoger la plataforma o herramienta adecuada para la implementación de la base de datos.

Con esto se concluyen las especificaciones y requerimientos del sistema, en el siguiente capítulo se utilizarán estos elementos para la generación de los diseños necesarios del sistema.

1.3 Análisis de Viabilidades

Para el desarrollo de cualquier sistema informático, es necesario determinar la viabilidad del proyecto; dicho análisis debe realizarse antes de comenzar a trabajar en la construcción del proyecto, este análisis se realizó para el presente proyecto con el fin de analizar si realmente es posible concluir con él de forma óptima y con las mejores herramientas al alcance, para así asegurar el desarrollo de un sistema exitoso.

Este análisis de viabilidades es necesario y debe abarcar las tres áreas de viabilidad como lo son la económica, técnica y operativa; buscando como fin que los recursos tanto económicos como el del tiempo asignados al desarrollo del presente proyecto sean los mínimos posibles, además que en los tres análisis en los que se detallan, deben de cumplir con el criterio de elección que es el de que el software a utilizar serán capaces de brindar los elementos necesarios para desarrollar la solución propuesta en el presente trabajo de tesis [11].

El estudio de las viabilidades es de las herramientas a utilizar, el saber si permitirán desarrollar el sistema con el fin de llegar a la conclusión si el sistema es capaz de desarrollarse de forma óptima y con el fin de brindar la mejor solución. Este análisis se utiliza a nivel de dirección técnica de un proyecto, con el fin de poder examinar las bases de toma de decisiones económicas, técnicas y operativas, y que de esta manera se pueda planificar el proyecto de la manera adecuada.

En el caso de realizar un análisis de viabilidades de la manera correcta, se pueden prevenir meses de esfuerzos e inversiones profesionales innecesarias, y por supuesto, evitar costos no previstos. Por lo que se vuelve una solución agradable en cuanto a el desarrollo del software se refiere.

El enfoque de la viabilidad económica es simplemente el de encontrar la solución más económica aceptable, mientras que la viabilidad técnica se realiza con el fin de encontrar las mejores herramientas para el desarrollo del software y para finalizar, la viabilidad operativa consiste en un análisis para asegurar la comodidad de los usuarios, conocer los puntos a fortalecer para el desarrollo del software referente a esta tesis, con el fin de asegurar que dicho software sea utilizado y pueda brindar la solución a la crisis del software y brindar una administración y desarrollo adecuado para los proyectos de software.

A continuación, se detalla el análisis de viabilidades con los tres enfoques mencionados: el técnico, económico y operativo, para así asegurar la viabilidad y cumplir con los objetivos establecidos en el presente proyecto.

1.3.1 Viabilidad Económica

Para un proyecto de desarrollo de software, siempre es necesario contar con una estimación de costo para su elaboración y el beneficio relacionado a la inversión para poder evaluar la rentabilidad del proyecto. En el presente proyecto los criterios de elección para las herramientas utilizadas son: rendimiento, costo y derechos de las mismas.

Cabe destacar que las herramientas se seleccionan por su desempeño, facilidad de uso, documentación, escalables, y como ventaja adicional es que son gratuitas y cumplen con las características necesarias para desarrollar este proyecto como inicialmente se desea. Los factores a considerar en este análisis son los que se muestran en la tabla 1.4. Las cantidades monetarias mencionadas en dicha tabla se calcularán acorde al salario promedio en caso de las personas involucradas y precio promedio en caso del hardware que se presentan en el mercado nacional mexicano.

Tabla 1.4 Recursos Básicos a Considerar para la Viabilidad Económica

Recursos básicos	Análisis
Tiempo para realizar el análisis.	El tiempo establecido para la realización de dicho análisis es de 2 semanas.
Tiempo de desarrollo del software	Primordialmente el tiempo considerado al desarrollo del presente proyecto es de no más de 18 meses. La fecha límite para la conclusión del proyecto se fijó el mes de septiembre de 2017.
Personal involucrado	El personal involucrado es únicamente el de los dos que desean obtener el título de Ingeniero con el desarrollo de la presente tesis y su directora de la misma.
Costo del personal involucrado.	De acuerdo con el Instituto Mexicano para la Competitividad, A.C. (IMCO) el salario mensual promedio para Ingenieros en computación es de $12 229.00 MXN como los desarrolladores y uno de $20 209.00 MXN para la directora de tesis se llega a un total de $268 002.00 MXN concluyendo los 18 meses establecidos para su desarrollo.
Costo estimado del hardware.	El hardware a utilizar consta únicamente de tres computadoras con un precio en mercado de $15 000 MXN aproximadamente, por lo que en hardware el costo asciende a $45 000 MXN.
Costos Fijos	Considerando el precio de gastos adicionales; la luz de acuerdo con CFE para el consumo promedio de $2 102.00 MXN, internet utilizando un paquete comercial por $2 394.00 MXN, renta de $42 000.00 MXN, todas las cantidades mencionadas son en un periodo de 6 meses.
Costo estimado total por el desarrollo del software.	Considerando los gastos anteriormente vistos se llega a la conclusión de un costo total de $359 498.00 MXN considerando precio por hora empleada en el proyecto.

El precio o costo establecido para el desarrollo de este software es puramente neto, no es el precio comercial que se definiría por el software en sí, sino únicamente de producción y a su vez es considerado aceptable ya que los costos pueden ser cubiertos con el fin de lograr la

conclusión de la presente tesis con el motivo de obtener el título como Ingeniero en Computación.

1.3.2 Viabilidad Técnica

Fue necesario realizar un análisis de la viabilidad técnica que engloba las herramientas a utilizar, con el fin de averiguar si los recursos técnicos seleccionados son capaces de satisfacer los requerimientos del proyecto bajo consideración, dicho análisis se hizo tomando en cuenta las capacidades que presenta cada herramienta de software a utilizar con el fin de seleccionar las mejores.

En la tabla 1.5, se muestran todas las tecnologías y herramientas seleccionadas a utilizar en el proceso de construcción del sistema.

El servicio de base de datos que ofrece *firebase* cumple con las especificaciones mencionadas anteriormente de la base detos, a excepción del modelo relacional, que se resuelve a través de una adaptación del modelo relacional al modelo *NoSQL*.

Tabla 1.5 Tecnologías y herramientas que serán implementadas

Tecnología/Herramienta/Plataforma	Funcionalidad correspondiente
Amazon Web Services	Plataforma web para alojamiento de la aplicación web.
Angular	Marco de trabajo para la creación de la aplicación web (aplicación base).
Bootstrap	Marco de trabajo para el diseño gráfico.
Visual Studio Code	IDE para el desarrollo de aplicaciones.
NodeJS	Intérprete de JavaScript del lado del servidor.
Nginx	Tecnología para la creación de servidor de alojamiento de la palicación web.
TypeScript	Lenguaje de programación para manejar programación orientada a objetos en JavaScript.
CSS	Lenguaje para el estilado de componentes gráficos.
HTML	Lenguaje para la elaboración delcontenido de las aplicaciones.
Firebase	Plataforma web que ofrece servicios como base de datos en tiempo real, almacenamiento y sistema de autenticación.
Digital Ocean	Plataforma web que ofrece alojamiento en la nube.
GitHub y *Git*	Plaforma y sistema de control de versiones.

1.3.3 Viabilidad Operativa

El análisis de viabilidad operativa consiste en si dadas las características del sistema a implementar; si el personal asignado al desarrollo del mismo es capaz de completar dicho fin. Además de la aceptación de los usuarios con este proyecto, si realmente soluciona el problema presente en la crisis y dará un paso para unificar los procesos en los que se desarrolla software.

Para la aplicación web los conocimientos en lenguajes como, HTML, CSS; son considerados aceptables para su desarrollo y en lo que concierne a la base de datos el servidor web los conocimientos necesarios son suficientes para su construcción.

De acuerdo con el ranking realizado por IEEE para el año 2015 como se muestra en la figura 1.2, los puntos en la figura van desde el cero al cien, asignando cien puntos al lenguaje más utilizado y cero al menos utilizado, para fines de esta tesis sólo se mostrarán los diez lenguajes más utilizados; de esta manera establecemos que los lenguajes objetivo a soportar como lo son el lenguaje Java con 100 puntos y el lenguaje C con 99.9 puntos, siguiéndole C++ con 99.6, Python con 95.8, C# con 91.8 [12]; Java es el más utilizado, es por eso que se seleccionó este lenguaje para soportarlo en el sistema, para garantizar un mayor alcance y mejor aceptación del presente trabajo.

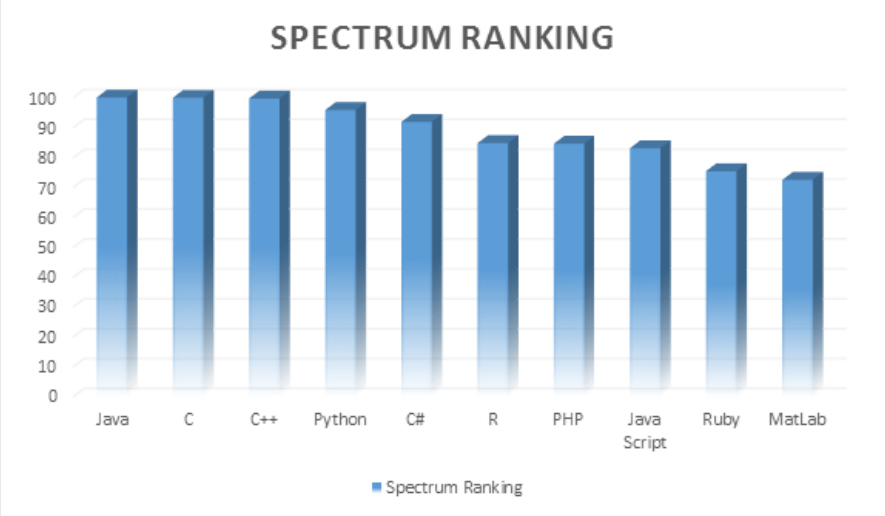

Figura 1.2 Ranking de Lenguajes de Programación en el año 2015 [12]

Capítulo 2

Diseño del Sistema

2.1 Diseño de Base de Datos

Es de gran importancia contemplar el diseño de las bases de datos que van a contener información persistente de los usuarios e información pertinente al sistema. El diseño permite la definición de una estructura adecuada para almacenar todos los datos que circulen dentro del sistema, de no ser así, el sistema no será escalable, dificultando el desarrollo adecuado del sistema.

Para realizar el diseño de las bases de datos necesarias en el sistema, se siguieron los 3 pasos convencionales para el diseño de base de datos, el diseño conceptual, lógico y el físico.

Es importante considerar que la persistencia de datos se dará en la plaforma de *firebase*, por lo cual fue necesario realizar una adaptación en el paso de diseño físico.

2.1.1 Diseño Conceptual

Lo que se busca como resultado final en el diseño conceptual es el *modelo de entidad-relación*, descrito por un diagrama de entidad-relación.

Para la elaboración del diagrama, es indispensable identificar las entidades, y sus respectivos atributos y relaciones entre ellas. En la tabla 2.1 se describen las entidades necesarias y sus atributos correspondientes que se identificaron según los requisitos establecidos en el capítulo anterior.

Tabla 2.1 Entidades del Sistema

Entidad	Atributos	Descripción
Archivo	id_archivo es_main nombre_archivo url_ archivo	Contiene todos los archivos generados por parte del usuario, cada archivo pertenece a un proyecto.
Mensaje	id_mensaje contenido_mensaje	Guarda los mensajes generados por el usuario y el id del propietario para mostrarlo en el área correspondiente de mensajes.
Plantilla	id_plantilla nombre_plantilla contenido_plantilla	Guarda las plantillas de código generadas por el usuario.
Proyecto	id_proyecto nombre_proyecto	Guarda la información del proyecto creado. Cada proyecto tiene un solo propietario.
Usuario	id_usuario nombre_usuario correo_usuario imgurl_usuario	Contiene la información básica del usuario.

El siguiente paso a elaborar es el diagrama de entidad-relación con base a la información en la tabla 2.1 agregando las relaciones entre entidades. La figura 2.1 representa en un diagrama la relación que existe entre las entidades, siendo el resultado del modelo conceptual de las bases de datos del sistema a desarrollar.

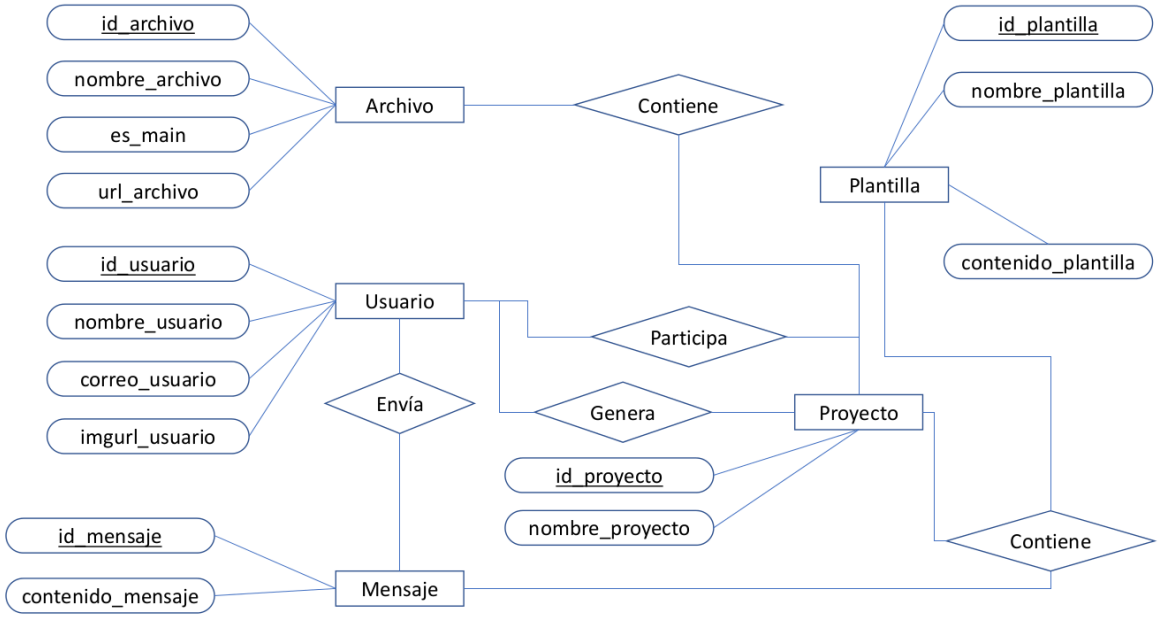

Figura 2.1 Diagrama Entidad-Relación y sus Atributos

Después se realizó un análisis de cardinalidad para indicar el número de entidades con las que puede encontrarse relacionada una entidad. En la figura 2.2 se muestra el diagrama de entidad-relación con sus respectivas cardinalidades.

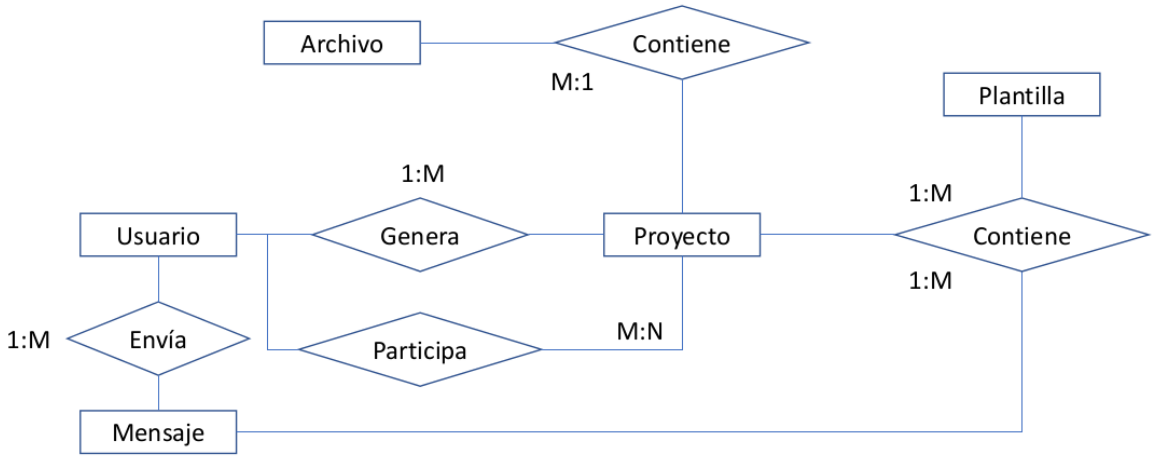

Figura 2.2 Diagrama Entidad-Relación con Cardinalidad

En la tabla 2.2 se describen las cardinalidades correspondientes en las relaciones entre las entidades.

Todas las relaciones poseen las mismas cardinalidades (uno a muchos), excepto la relación "Participa". Es importante resaltar que esta relación existe para representar el compartimiento del proyecto con usaurios no propietarios en la base de datos en una tabla a parte.

Tabla 2.2 Cardinalidad de cada Relación

Relación	Entidades Relacionadas	Cardinalidad
Genera	Usuario-Proyecto	Uno a Muchos
Envía	Usuario-Mensaje	Uno a Muchos
Contiene	Proyecto-Archivo	Uno a Muchos
Contiene	Proyecto -Plantilla	Uno a Muchos
Contiene	Proyecto -Mensaje	Uno a Muchos
Participa	Usuario-Proyecto	Muchos a Muchos

2.1.2 Diseño Lógico

En este paso se busca obtener un modelo relacional a través del diagrama de entidad-relación que se generó en el diseño conceptual. Una vez obtenido el modelo relacional, se busca eliminar anomalías a través de una normalización.

Antes de abstraer información relevante del diagrama de entidad-relación para generar el modelo relacional, es necesario determinar las propiedades de los atributos de cada entidad, cuál es la llave primaria y el tipo de dato correspondiente. En la tabla 2.3 se pueden observar las propiedades de cada atributo de las entidades.

Como se puede observar, a excepción de 1 atributos, todos los restantes son de tipo cadena-texto. También se puede apreciar que, por cada relación uno a muchos existente en el diagrama, hay una llave foránea para su respectiva referencia. En el diagrama se encuentran 5 relaciones de uno a muchos, lo cual indica que existen 5 llaves foráneas.

Tabla 2.3 Propiedades de Atributos

Entidad	Atributo	Llave Primaria	Tipo de Dato
Archivo	id_archivo	Si	cadena-texto
	nombre_archivo	No	cadena-texto
	es_main	No	booleano
	url_ archivo	No	cadena-texto
	id_proyecto_perteneciente	No, Foránea	cadena-texto
Mensaje	id_mensaje	Si	cadena-texto
	contenido_mensaje	No	cadena-texto
	id_usuario_propietario	No, Foránea	cadena-texto
	id_proyecto_perteneciente	No, Foránea	cadena-texto
Plantilla	id_plantilla	Si	cadena-texto
	nombre_plantilla	No	cadena-texto

	contenido_plantilla	No	cadena-texto
	id_proyecto_perteneciente	No, Foránea	cadena-texto
Proyecto	id_proyecto	Si	cadena-texto
	nombre_ proyecto	No	cadena-texto
	id_usuario_propietario	No, Foránea	cadena-texto
Usuario	id_usuario	Si	cadena-texto
	nombre _usuario	No	cadena-texto
	correo_ usuario	No	cadena-texto
	imgurl_ usuario	No	cadena-texto

Una vez conocidos los atributos de cada entidad, se realiza la abstracción del diagrama para traducirlo en el modelo relacional. En la tabla 2.4 se muestra el resultado de la abstracción del diagrama entidad-relación.

Tabla 2.4 Resultado del Modelo Relacional

Modelo Relacional
ARCHIVO(id_archivo varchar, nombre_archivo varchar, es_main boolean, url_archivo varchar, id_proyecto_perteneciente varchar)
MENSAJE(id_mensaje varchar, contenido_mensaje varchar , id_usuario_propietario varchar, id_proyecto_perteneciente varchar)
PLANTILLA(id_plantilla varchar, nombre_plantilla varchar, contenido_plantilla varchar, id_proyecto_perteneciente varchar)
PROYECTO(id_proyecto varchar, nombre_proyecto varchar, id_usuario_ propietario varchar)
USUARIO(id_usuario varchar , nombre_usuario varchar, correo_usuario varchar, imurl_usuario varchar)

Se modelaron 5 relaciones a través del diagrama, la entidad archivo, mensaje, plantilla, proyecto y usuario. Cabe mencionar que una relación en el modelo relacional, hace referencia a una tabla o conjunto de registros a nivel físico.

Para finalizar el diseño lógico, es necesario normalizar el modelo relacional para evitar inconsistencia, redundancia y otros errores que no permitan el almacenamiento correcto de los datos. Para eso, se utilizaron las *reglas formales de Edgar F. Codd*, reglas formales que permiten la normalización de las bases de datos a través de 3 reglas convencionales.

En la primera forma normal de Edgar F. Codd se busca cumplir con los siguientes puntos que aparecen en la tabla 2.5.

Tabla 2.5 Puntos a Considerar para la Primera Forma Normal

Puntos a Cumplir	Descripción
Atomicidad de Datos	El valor que persista en un atributo (columna) no se debe de poder dividir en otros valores.
Llave Primaria Única	En cada relación (tabla) debe de existir una sola llave primaria.
Llave con Valor	Una llave primaria no puede ser de valor nulo.
Integridad de Columnas	En una relación (tabla) no debe de existir una variación de número de columnas.
Dependencia Funcional	Los atributos (columnas) que no son llave primaria se deben de poder identificar a través de una llave primaria.
Independencia de Órden	No importa si el órden de los atributos (columnas) de una tabla cambia, el dato que representa debe de seguir siendo el mismo.

A continuación se busca el cumplimiento de la primera forma normal para cada relación (tabla) modelada, en la tabla 2.6 se puede observar el resultado del cumplimiento de la primera forma normal para cada relación.

Como se puede observar el modelo relacional generado cumple con todos los puntos para considerar que se encuentra en la primera forma normal.

Tabla 2.6 Verificación de Cumplimiento de la Primera Forma Normal

Relación	Concepto a Cumplir	Resultado
ARCHIVO	Atomicidad de Datos	Si cumple
	Llave Primaria Única	Si cumple
	Llave con Valor	Si cumple
	Integridad de Columnas	Si cumple
	Dependencia Funcional	Si cumple
	Independencia de Órden	Si cumple
MENSAJE	Atomicidad de Datos	Si cumple
	Llave Primaria Única	Si cumple
	Llave con Valor	Si cumple
	Integridad de Columnas	Si cumple
	Dependencia Funcional	Si cumple
	Independencia de Órden	Si cumple
PLANTILLA	Atomicidad de Datos	Si cumple
	Llave Primaria Única	Si cumple
	Llave con Valor	Si cumple
	Integridad de Columnas	Si cumple

	Dependencia Funcional	Si cumple
	Independencia de Órden	Si cumple
PROYECTO	Atomicidad de Datos	Si cumple
	Llave Primaria Única	Si cumple
	Llave con Valor	Si cumple
	Integridad de Columnas	Si cumple
	Dependencia Funcional	Si cumple
	Independencia de Órden	Si cumple
USUARIO	Atomicidad de Datos	Si cumple
	Llave Primaria Única	Si cumple
	Llave con Valor	Si cumple
	Integridad de Columnas	Si cumple
	Dependencia Funcional	Si cumple
	Independencia de Órden	Si cumple

Siguiendo los pasos establecidos, restan 2 formas normales para el cumplimiento de la normalización propuesta. En la segunda *forma normal de Edgar F. Codd* se busca una dependencia funcional completa, es decir, que no existan *dependencias parciales*. Para determinar si todos los atributos cumplen con la segunda forma normal, se elaboró en la tabla 2.7 una corroboración para determinar si cada atributo (que no sea llave primaria) cumple con la dependencia funcional completa.

Al mirar la tabla se puede concluir que todos los atributos son únicamente dependientes de una llave primaria.

Al mismo tiempo también se realizó el último paso de normalización, la tercera *forma normal de Edgar F. Codd*. Esta forma normal consiste en evitar una dependencia transitiva. La dependencia transitiva se presenta cuando un atributo no depende inmediatamente de la llave primaria, sino de otro atributo dentro de la relación. En la misma tabla 2.7 se puede observar que cada atributo (no primario) solamente puede depender de la llave primaria.

Tabla 2.7 Corroboración de la Segunda Forma Normal y la Tercera Forma Normal

Relación	Atributo	Dependencia
Archivo	id_archivo	No depende de algún atributo
	nombre_archivo	id_archivo
	es_main	id_archivo
	url_ archivo	id_archivo
	id_proyecto_perteneciente	id_archivo
Mensaje	id_mensaje	No depende de algún atributo
	contenido_mensaje	id_mensaje
	id_usuario_propietario	id_mensaje
	id_proyecto_perteneciente	id_mensaje
Plantilla	id_plantilla	No depende de algún atributo

	nombre_plantilla	id_plantilla
	contenido_plantilla	id_plantilla
	id_proyecto_perteneciente	id_plantilla
Proyecto	id_proyecto	No depende de algún atributo
	nombre_ proyecto	id_proyecto
	id_usuario_propietario	id_proyecto
Usuario	id_usuario	No depende de algún atributo
	nombre _usuario	id_usuario
	correo_ usuario	id_usuario
	imgurl_ usuario	id_usuario

Con esto se concluye la normalización. El modelo relacional está normalizado por las 3 *formas normales de Edgar F. Codd* propuestas.

2.1.3 Diseño Físico

En este paso final para concluir con el diseño de las bases de datos, se realizó el diseño físico adaptando el diseño lógico que se obtuvo a un modelo *NoSQL*. Como se decidió utilizar la plataforma de *firebase* para soportar la base de datos del sistema, es indispensable adaptar el diseño relacional que se elaboró a uno no relacional, que es el formato *JSON*.

En la tabla 2.8 se observa la adaptación que se realizó para mantener la relación diseñada de cada entidad en formato *JSON*.

Ningún atributo de una entidad (objeto) contiene el id del mismo (llave primaria), esto debido a que el mismo registro es la llave primaria y de ahí se puede obtener.

Tabla 2.8 Adaptación del Modelo Relacional a NoSQL

Entidad	Adaptación	Observación

Archivo	"archivos": [
	"archivos_(id_proyecto)": [
	"id_archivo": {
	"nombre_archivo": text,
	"es_main": boolean,
	"url_archivo": text
	}
	]
	]
		El registro “archivos” contiene en un arreglo diferentes colecciones de archivos de proyectos. El (id_proyecto) se sustituye por la llave primaria (key) del proyecto al que pertenece, para así simular la inclusión de la llave foránea en el registro de los archivos.

Mensaje	"mensajes": ["mensajes_(id_proyecto)": ["id_mensaje": { "contenido_mensaje": text, "id_usuario_propietario": text	Se adaptó igual que la entidad archivo, solo que la llave foránea “id_usuario_propietario” se incluyó como atributo y no en

	}]]	el registro. Esto para evitar la complejidad al sustraer los datos.

Plantilla	"plantillas": [
	"plantillas_(id_proyecto)": [
	"id_plantilla": {
	"nombre_plantilla": text,
	"contenido_plantilla": text
	}
	]
	]
		Se adaptó igual que la entindad archivo.

Proyecto	"proyectos": [
	"id_proyecto_(id_usuario)": {
	"integrantes": [
	"id_usuario": {
	"es_propietario" boolean
	}
	],
	"nombre_proyecto": text,
	}
	]
		El registro “proyectos” contiene un arreglo proyectos guardados. Cada proyecto tiene 2 atributos, el atributo “nombre_proyecto” y el atributo “integrantes” que contiene un arreglo de los usuarios. El atributo “integrantes” ayuda a represnetar la relación muchos a muchos.

Usuario	"usuarios": [
	"id_usuario": {
	"nombre_usuario": text,
	"correo_usuario": text,
	"imgurl_usuario": text
	}
	]
		El registro “usuarios” contiene un arreglo de usuarios con el atributo “nombre_usuario”, “correo_usuario” y “imgurl_usuario”.

Con esto se concluye el diseño físico y por consecuente el diseño general de la base de datos para el sistema.

2.2 Diseño de Arquitectura del Sistema

A continuación, en la tabla 2.9, se describen los requerimientos funcionales basados en los atributos de calidad esperados, en cuanto a la prioridad, se maneja una escala del uno al cinco, donde cinco es la prioridad más alta y uno la más baja.

Tabla 2.9 Requerimientos Funcionales

Identificador	Requerimiento	Descripción	Prioridad
RF01	Acceso	El usuario debe ser capaz de iniciar sesión o registrarse desde cualquier ordenador con un navegador de *google chrome*.	5
RF02	Acceso	El sistema debe validar las credenciales del usuario para brindar acceso al sistema, o bien denegarlo.	5
RF03	Usabilidad	El sistema debe mostrar los proyectos que el usuario tenga almacenados.	4
RF04	Almacenamiento	El sistema debe guardar los proyectos del usuario en la nube.	2
RF05	Usabilidad	El usuario debe ser capaz de crear un proyecto nuevo.	5
RF06	Usabilidad	El usuario debe ser capaz de editar un proyecto existente.	5
RF07	Usabilidad	El usuario debe ser capaz de compilar su código.	5
RF08	Usabilidad	El sistema debe retornar el resultado de la compilación del código del usuario.	5
RF09	Usabilidad	El usuario debe ser capaz de ejecutar su proyecto.	5
RF10	Usabilidad	El sistema debe retornar el proceso de ejecución del proyecto del usuario.	5
RF11	Usabilidad	El usuario debe ser capaz de crear plantillas de código.	1
RF12	Usabilidad	El usuario debe ser capaz de agregar hasta 5 usuarios al proyecto de su elección.	4
RF13	Usabilidad	El usuario debe ser capaz de eliminar usuarios del proyecto de su elección.	4
RF14	Usabilidad	El usuario debe ser capaz de enviar mensajes.	3
RF15	Usabilidad	El usuario debe ser capaz de enviar mensajes privados del proyecto.	3

RF16	Usabilidad	El sistema debe realizar el envío y recepción de mensajes a los distintos canales de un proyecto.	3
RF17	Almacenamiento	El sistema debe almacenar las conversaciones en la nube.	3
RF18	Usabilidad	El usuario debe ser capaz de editar sus sus proyectos y almacenarlos en la nube.	4

A continuación, en la tabla 2.10 se muestran los requerimientos no funcionales con los que cuenta el sistema, de la misma manera que en la tabla referente a los requerimientos funcionales, cuenta con una prioridad, siendo 5 la más alta y 1 la más baja.

Tabla 2.10 Requerimientos No Funcionales

Identificador	Requerimiento	Descripción	Prioridad
RNF01	Usabilidad	La interfaz de usuario debe ser sencilla e intuitiva en las diferentes aplicaciones del sistema.	5
RNF02	Desempeño	La compilación y ejecución de un proyeto debe ser a través de un servicio web desarrollado con *nodeJs* y montado en una máquina virtual de *Linux*.	5
RNF03	Multiplataforma	La interacción entre las tres herramientas debe ser la más óptima.	4
RNF04	Desempeño	Los usuarios deben contar con un identificador único (llave primaria) para permitir el esquema entidad relación en una base de datos *NoSQL*.	5
RNF05	Seguridad	La contraseña para la cuenta de usuario debe contener por lo menos una letra, un número y un carácter especial y es administrada por la *API* de *firebase* por cuestión de seguridad.	4
RNF06	Almacenamiento	Los proyectos deben almacenarse en la nube para su fácil acceso desde cualquier navegador de goolge chrome.	2
RNF07	Desempeño	La comunicación entre los usuarios se hace a través del identificador único definido al momento de ser registrados en el sistema.	4
RNF08	Usabilidad	Las interfaces gráficas deben ser responsivas para los diferentes tamaños de pantallas.	3
RNF09	Rendimiento	Los datos deben ser manejados de forma óptima con el enfoque entidad relación,	5

		es decir, evitar redundancia en la base de datos.	
RNF10	Rendimiento	El sistema debe ser capaz de manejar 10 proyectos sin que los componentes de de la aplicación sufran sobrecarga de información y afecte su rendimiento.	5
RNF11	Rendimiento	El tamaño máximo definido para cada clase de un proyecto es de 250 Kb.	3
RNF12	Desempeño	El máximo número de archivos para un proyecto es de 10.	1

Para el diseño de la arquitectura del sistema se utilizó el modelado *SOA* (Arquitectura Orientada a Servicios), esto debido a los requerimientos planteados, de manera que esta arquitectura es la más adecuada para el desarrollo del sistema, los servicios son la base para el modelado de dicha arquitectura; dichos servicios están construidos con base en los requerimientos del sistema.

En la figura 2.3 se muestra la interacción del sistema en términos generales. Muestra cómo es la comunicación entre la aplicación web con el servicio web principal, cabe mencionar que como lo indica la figura 2.3, la comunicación entre estos es bidireccional.

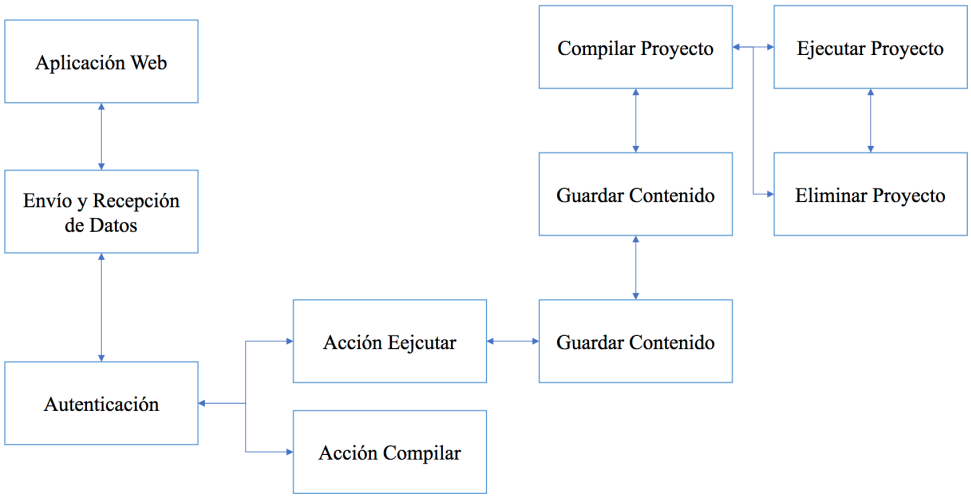

Figura 2.3 Diagrama de Interacción General del Sistema

En la Figura 2.4 se muestra la interacción de la aplicación y servicio web, respecto al almacenamiento de la información.

Como se puede observar en la figura 2.4 el almacenamiento de la información es independiente de la aplicación, pero esta se comunica con la nube que será la encargada de gestionar todos los procesos, entre los que se pueden mencionar enviar o recibir actualizaciones, el almacenamiento en la nube cuenta con todos los proyectos existentes

mientras que la aplicación de escritorio, movil y web solo cuentan con la información del proyecto con el que actualmente se esté trabajando.

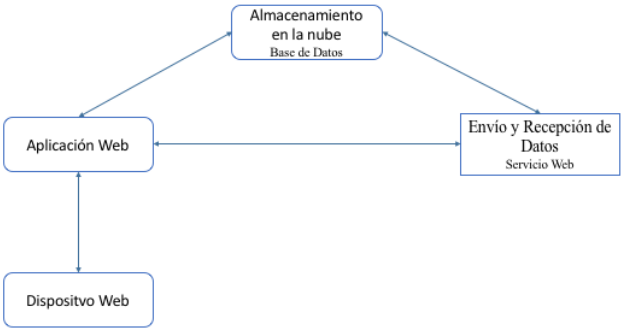

Figura 2.4 Almacenamiento de Información

2.3 Diseño Procedimental

Se elaboraron diseños procedimentales para tener una mejor comprensión de los procesos lógicos del sistema y así facilitar la etapa de desarrollo. Para el diseño procedimental del sistema se utilizaron diagramas de flujo que explican el comportamiento de cada componente del sistema.

En la figura 2.5 se muestra el comportamiento correspondiente al inicio de sesión.

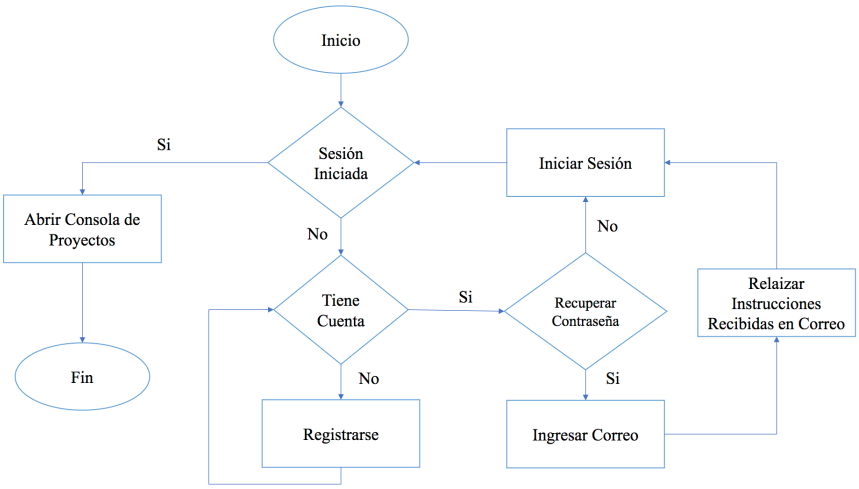

Figura 2.5 Diseño Procedimental Inicio de Sesión

El siguiente diseño como se muestra en la figura 2.6, es el pertinente a la consola de proyectos donde muestra los proyectos pertenecientes del usuario.

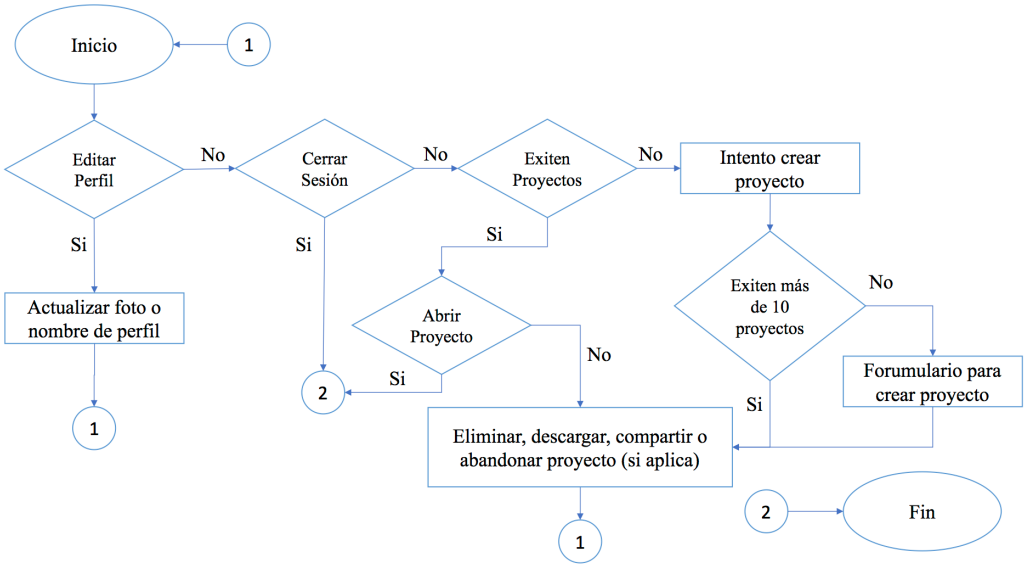

Figura 2.6 Diseño Procedimental Consola de Proyectos

El diseño procedimental para el área de trabajo se muestra en la figura 2.7, en esta figura se presenta el comportamiento de la aplicación para interactuar con el proyecto seleccionado por el usuario.

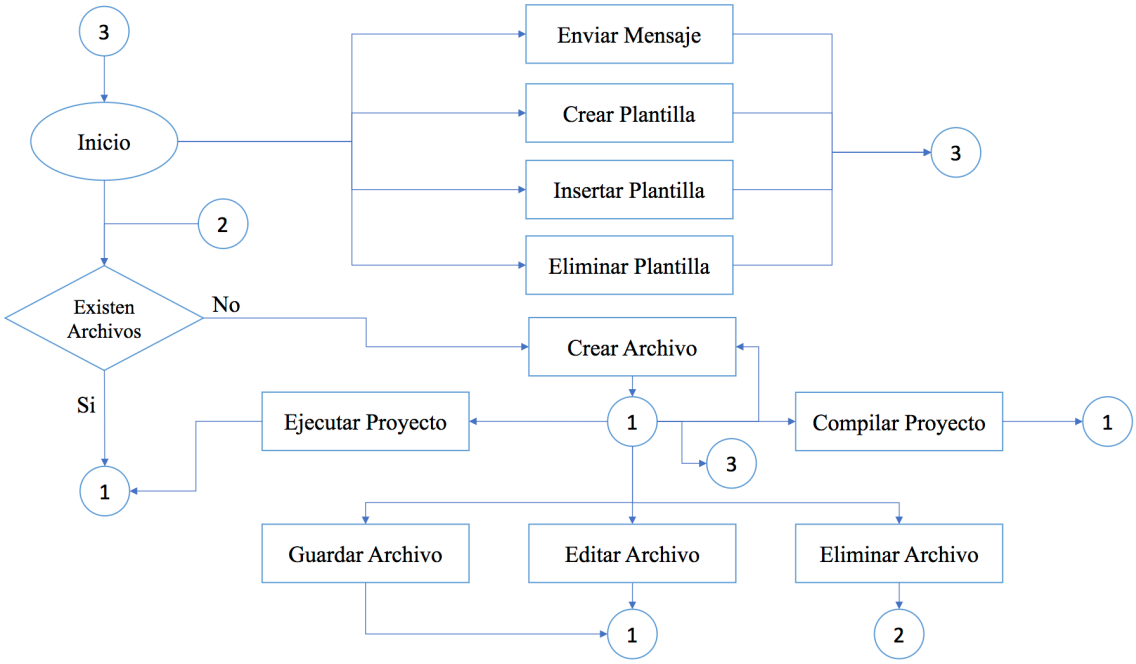

Figura 2.7 Diseño Procedimental Área de Trabajo

Para finalizar el diseño procedimental, en la figura 2.8 se muestra el diseño referente al comportamiento que tiene el servicio web.

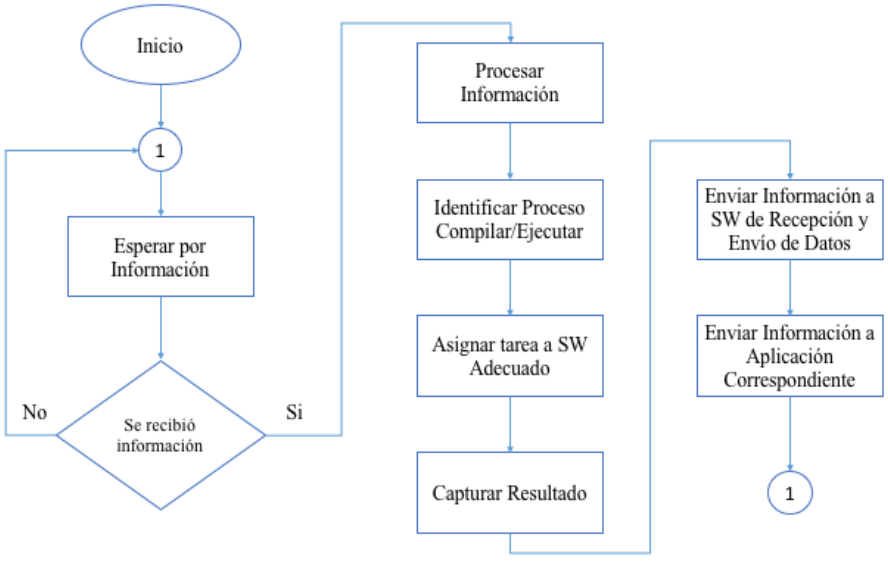

Figura 2.8 Diseño Procedimental Servicio Web

2.4 Diagrama de Caso de Uso

El diagrama de caso de uso que engloba toda la información y todas las operaciones del sistema, a su vez representa la interacción entre el sistema y los usuarios; la forma en que se espera que dichos usuarios utilicen el sistema. En este caso el diagrama de caso de uso de la aplicación web referente al inicio de sesión se muestra en la figura 2.9.

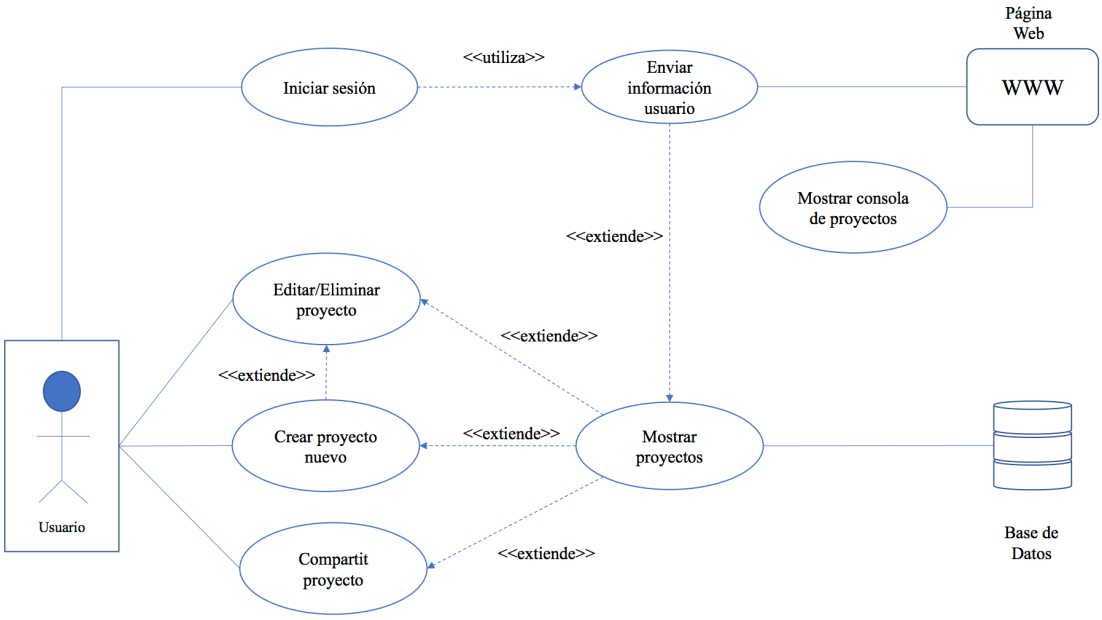

Figura 2.9 Diagrama de caso de uso "Inicio de Sesión"

La descripción del caso de uno "Inicio de sesión" es el que se muestra a continuación en la tabla 2.11.

Tabla 2.11 Descripción de Caso de Uso "Inicio de Sesión"

Caso de Uso: Inicio de Sesión	
Actor: Usuario	
Caso Ideal	Alternativas
1. El usuario inicia sesión en la plataforma	1.1. En caso de no contar con una cuenta se le solicita que cree una cuenta
2. *Firebase* valida la cuenta del usuario	2.1. En caso de no coincidir el nombre de cuenta o contraseña regresará al punto número 1
3. Se muestran una lista con todos los proyectos almacenados en la nube	3.1. En caso de no tener ningún proyecto creado se mostrará una lista en blanco
4. El usuario crea un proyecto	
5. El usuario edita o elimina un proyecto	5.1. Se solicita reintentar la acción

En la figura 2.10 se muestra la interacción del sistema una vez iniciada la sesión del usuario, la cual corresponde a la sección de edición de algún proyecto en específico y su descripción del mismo se muestra en la tabla 2.12.

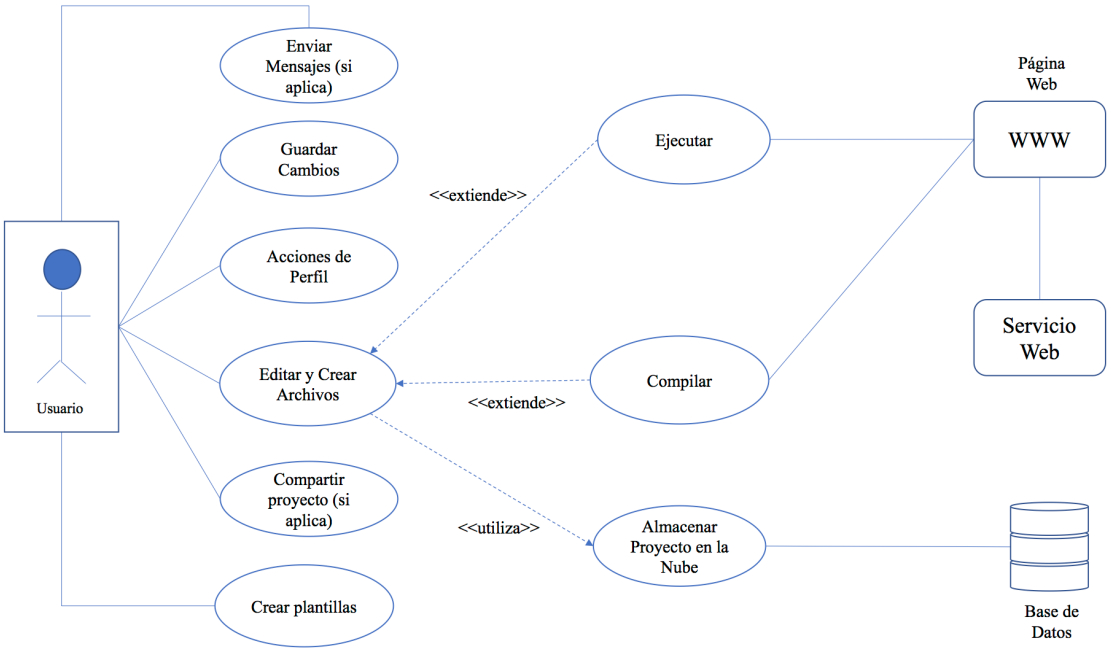

Figura 2.10 Diagrama de Caso de Uso "Edición de Proyecto"

Tabla 2.12 Descripción de Caso de Uso "Edición Proyecto"

Caso de Uso: Edición de Proyecto	
Actor: Usuario	
Caso Ideal	Alternativas
1. El usuario edita el contenido de un proyecto en específico	
2. Los cambios en el contenido del proyecto se cargan en la nube	2.1. En caso de fallar al guardar los cambios se notifica al usuario y se le solicita que intente guardarlos nuevamente.
3. El servicio web se encarga de compilar el código fuente; retorna el resultado de la compilación.	
4. El servicio web se encarga de ejecutar el código fuente	4.1. Si la última compilación del sistema hay errores, no permitirá la ejecución del mismo y se muestra un mensaje para corregir el código. 4.2. Si se presente un error en la compilación se muestra dicho error al usuario.
5. El Usuario puede visualizar, crear y/o eliminar plantillas	5.1. De no tener plantillas creadas se mostrará una lista en blanco.

6. El usuario es capaz de enviar mensajes	6.1. En caso de ser el único integrante del proyecto se le solicitará agregar participantes para poder enviar mensajes
7. EL usuario puede acceder al menu de acciones relacionadas a la cuenta del usuario	
8. El usuario es capaz de guardar los cambios	8.1. Se le solicita salvar los cambios una vez más
9. Rehacer y deshacer acciones	

En la figura 2.11 se muestra la funcionalidad de compartir un proyecto y la tabla 2.13 corresponde a la descripción del mismo.

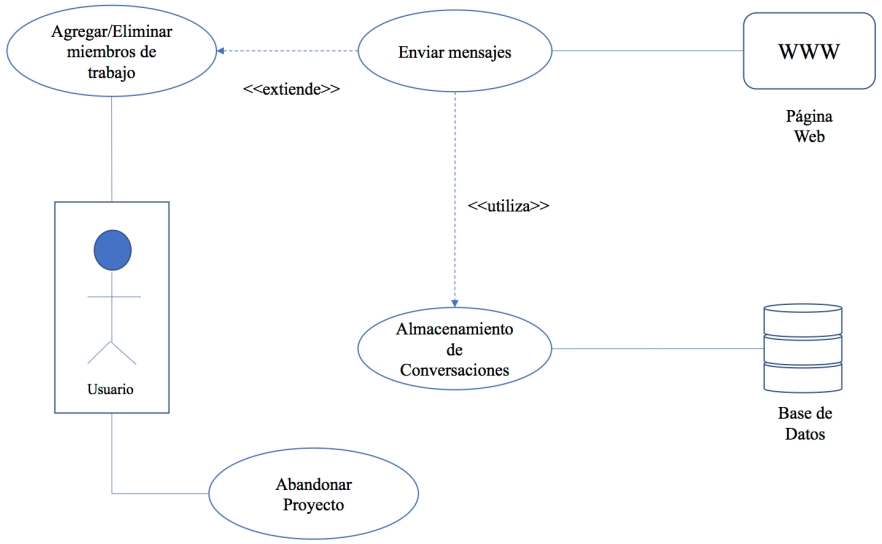

Figura 2.11 Diagrama de Caso de Uso "Compartir Proyecto"

Tabla 2.13 Descripción de Caso de Uso "Compartir Proyecto"

Caso de Uso: Compartir Proyecto	
Actor: Usuario	
Caso Ideal	Alternativas
1. El usuario puede eliminar o agregar miembro de trabajo	1.1. Al intentar agregar a un sexto miembro se cancela la acción
2. El usuario puede enviar mensajes a los integrantes del proyecto	

3. El almacenamiento de todas las conversaciones es de forma interna y en la nube	3.1. En caso de no contar con una conexión estable, los mensajes no podrán ser enviados
4. El usuario puede abandonar un proyecto si el no es el creador del mismo	4.1. Se notifica que la acción no pudo ser realizada y que se intente de nuevo

2.5 Diseño Gráfico de la Aplicación

A continuación, se detalla el diesño de la aplicación web para una mejor comprensión del sistema planteado.

La aplicación comienza con la pantalla de inicio de sesión, cuyo objetivo es autenticar el usuario. La figura 2.12 muestra el diseño de la pantalla para autenticar al usuario. En el caso de no tener una cuenta, el usuario es capaz de crearla con el botón "Registrarse". El texto "Recuperar Contraseña" es un botón que navega al usuario hacia la pantalla de recuperación de contraseña.

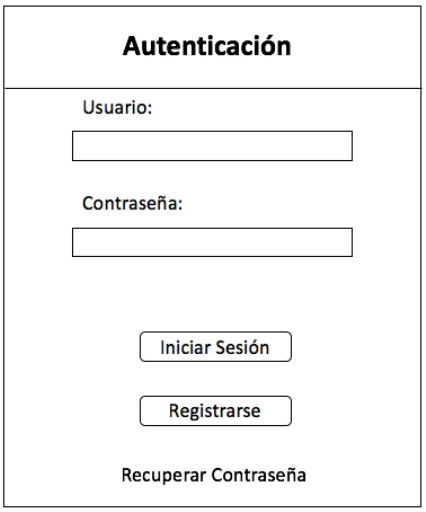

Figura 2.12 Diseño Autenticación de Usuario

Al tratar crear una cuenta, la aplicación le solicita al usuario llenar 4 campos para la creación de la cuenta como se muestra en la figura 2.13. El campo de usuario, correo, contraseña y la confirmación de la contraseña. Con el botón "Crear Cuenta" la aplicación solicita una petición de creación de usuario al servicio de *firebase* para crear la cuenta con los datos introducidos. Al tener éxito con la creación del usuario, el usuario navega directamente hacia la pantalla de consola de proyectos.

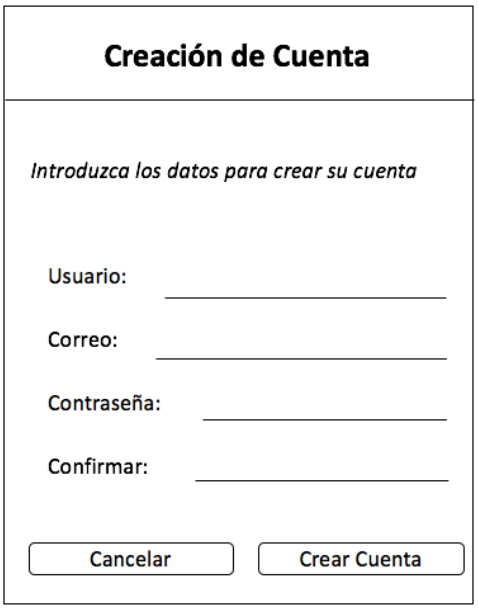

Figura 2.13 Diseño Pantalla para Registrar de Usuario

En la figura 2.14 se muestra el diseño de la pantalla para recuperar la contraseña. El usuario introduce su correo electrónico para establecer de nuevo la contraseña a través del correo enviado del servicio de *firebase*. El texto "Iniciar Sesión" es un botón que regresa el usuario hacia la pantalla de iniciar sesión.

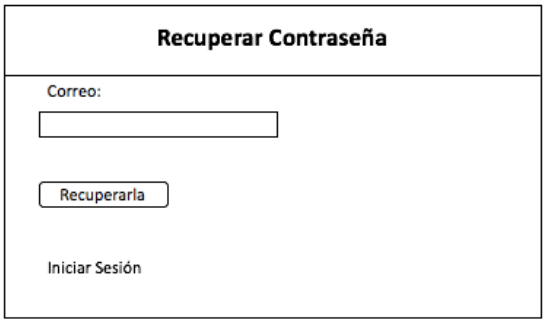

Figura 2.14 Diseño Pantalla Recuperar la Contraseña

Al iniciar sesión o registrar el usuario, la aplicación debe de navegar hacia la consola de proyectos. En la figura 2.15 se observa el diseño de la consola. Tiene un botón "Crear Proyecto" para llamar un modal y permitir al usuario crear proyectos nuevos. La foto de perfil y el nombre de usuario proporciona un menú sencillo. Los contenedores de proyectos existen para que los usuarios puedan interactuar con el proyecto (eliminar, agregar integrantes, etcétera).

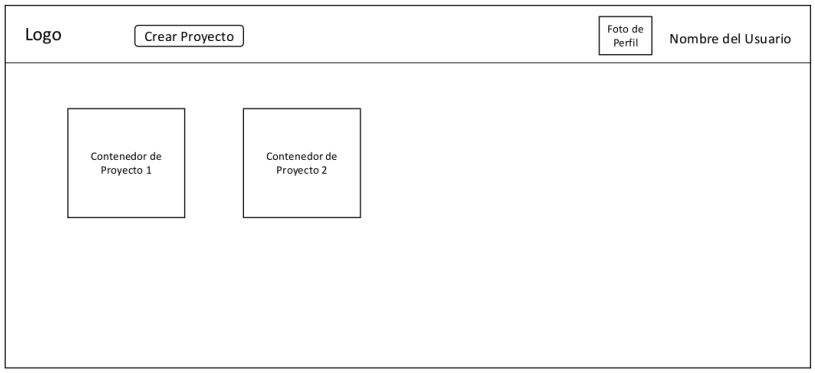

Figura 2.15 Diseño para Consola de Proyectos

El menú que contiene el área de foto de perfil, contiene la opción "Edición de Perfil" y "Cerrar Sesión". Al escoger la primera opción, se abre un modal como se muestra en la figura 2.16.

Este modal brinda al usuario en total 3 opciones diferentes. Editar el nombre del usuario, escoger una nueva foto de perfil y cambiar la contraseña. Al presionar el botón "Escoger Nueva Foto" abre un selector de recursos para que el usuario sea capaz de actualizar su foto. El botón "Cambiar Contraseña" debe de enviar un correo al usuario para que con la liga incluida actualice la contraseña de inicio de sesión.

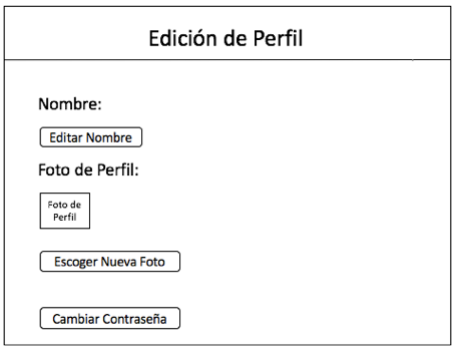

Figura 2.16 Diseño Modal Edición de Perfil

El modal para crear proyectos se diseño como se muestra en la figura 2.17. Tiene un campo de texto para el nombre del proyecto y otro para dar una descripción opcional. Con el botón "Cancelar" se descarta el modal y con el botón "Aceptar" se crea el proyecto.

Figura 2.17 Diseño Modal Crear Proyectos

Al crear un proyecto, se genera un panel de proyecto que debe de tener el diseño que se muestra en la figura 2.18. Tiene el botón de "Opciones" que abre un menú para que el usuario interactúe con el proyecto, y otro de "Abrir Proyecto" para navegar hacia el área de trabajo. También cuenta con una etiqueta de "Propietario" o "Participante" para clasificar los proyectos. Si el proyecto tiene una descripción, lo muestra debajo de la etiqueta.

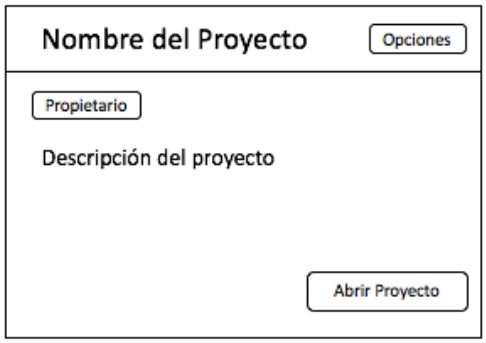

Figura 2.18 Diseño Panel Proyecto

Existen 2 casos de menú, si el usuario es el propietario del proyecto, el menú contiene las opciones que se muestran en la tabla 2.14.

Tabla 2.14 Menú Propietario Panel de Proyecto

Opciones	Acción que Realiza
Descargar Proyecto	Descarga todo el proyecto en un archivo zip a través del navegador.
Eliminar Proyecto	Abre un modal para confirmar la eliminación del proyecto para remover todos los registros relacionados con el proyecto.
Compartir Proyecto	Abre un modal para la introducción del correo electrónico del integrante a agregar.
Eliminar Integrante	Abre un modal para la selección de integrantes a remover del proyecto.

De manera contraria, el menú contiene las opciones de la tabla 2.15.

Tabla 2.15 Menú Participante Panel de Proyecto

Opciones	Acción que Realiza
Descargar Proyecto	Descarga todo el proyecto en un archivo zip a través del navegador.
Abandonar Proyecto	Elimina el registro del participante de la base de datos.

Al seleccionar un proyecto, la aplicación navega hacia la pantalla de área de trabajo. En la figura 2.19 se muestra el diseño de las diferentes secciones del área de trabajo. El área de trabajo se conforma por la barra de menú, área de proyecto, editor de texto, área multifuncional y área de ejecución.

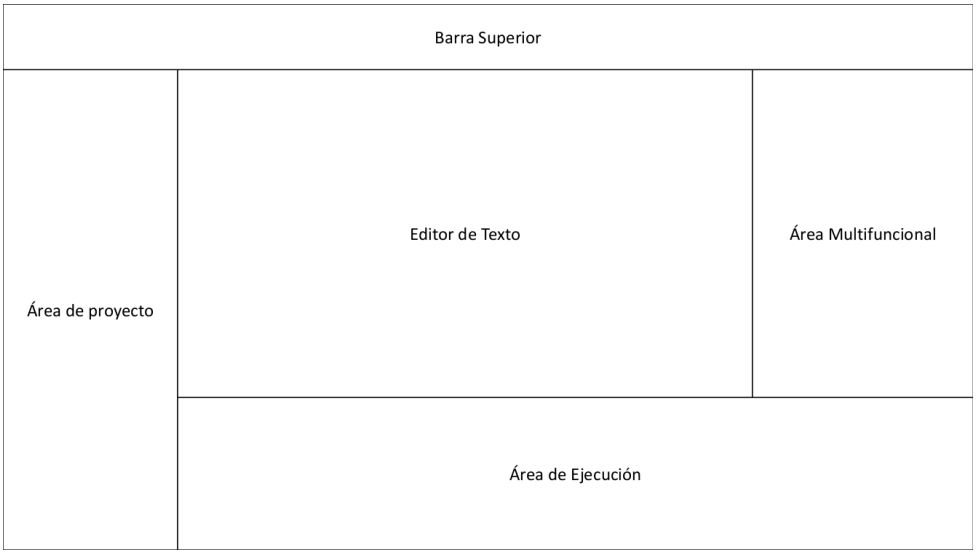

Figura 2.19 Diferentes Secciones del Área de Trabajo

A continuación, se detalla el diseño de las 5 secciones del áreade trabajo.

Barra Superior: Esta área contiene un botón para regresar a la consola de proyectos, menús, botones de rápido acceso y el área de perfil para cerrar sesión como se muestra en la figura 2.20.

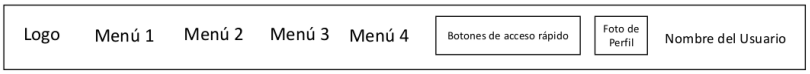

Figura 2.20 Diseño Barra Superior

Área de Proyecto: Esta área se diseñó como se muestra en la figura 2.21. Contiene un botón "Agregar Archivo" que abre un modal para agregar archivos dentro del proyecto. Todos los archivos que se agregan se acomodan en orden alfabético.

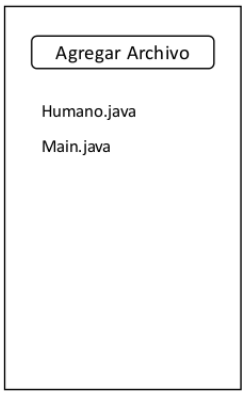

Figura 2.21 Diseño Área de Proyecto

El modal para agregar archivos se diseñó como se observa en la figura 2.22. Este modal tiene un campo de texto para introducir el nombre del archivo y una casilla para crear el archivo con el código main. El botón "Crear" afirma la creación del archivo y el botón "Cancelar" remueve el modal.

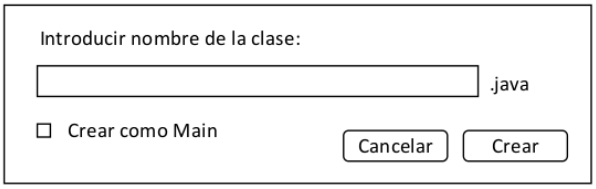

Figura 2.22 Diseño Área de Proyecto

Editor de Texto: El diseño de esta área quedó como se muestra en la figura 2.23. Arriba del área de texto se muestra el nombre del archivo seleccionado y lo acompañan 2 etiquetas que muestran el estado del archivo. Dependiendo el estado de guardado del archivo, cambia el contenido de la etiqueta de estado guardado. Si el archivo es main (contiene el método main), entonces muestra la etiqueta main.

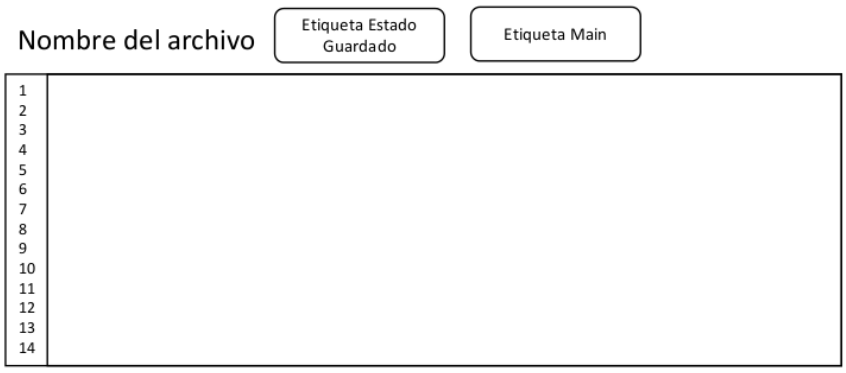

Figura 2.23 Diseño Editor de Texto

Área Multifuncional: Esta área se divide en 2 partes, en el área de mensjaes y de plantillas. Esta área contiene pestañas para cada parte correspondiente.

En la figura 2.24 se muestra el diseño del área multifuncional de mensajes. Cada mensaje viene acompañado de la foto de perfil del usuario y su nombre. El contenedor de mensajes se puede deslizar para ver todos los mensajes existentes del proyecto. Esta área contiene en la parte inferior un área de texto para introducir el contenido del mensaje y un botón para enviar el contenido de mensaje.

Figura 2.24 Diseño Área Multifuncional de Mensajes

En la figura 2.25 se muestra el diseño del área multifuncional de plantillas. Muestra la cantidad de plantillas que tiene el proyecto y su nombre. Contiene un área de texto que muestra el contenido de la plantilla y se puede modificar. El botón "Insertar" agrega el texto mostrado en el área de texto "Contenido de la Plantilla" donde se ubica el cursor en el editor de texto. El botón "Eliminar" abre un modal para confirmar la eliminación de la plantilla y el botón "Crear" abre el modal para agregar plantillas en el proyecto.

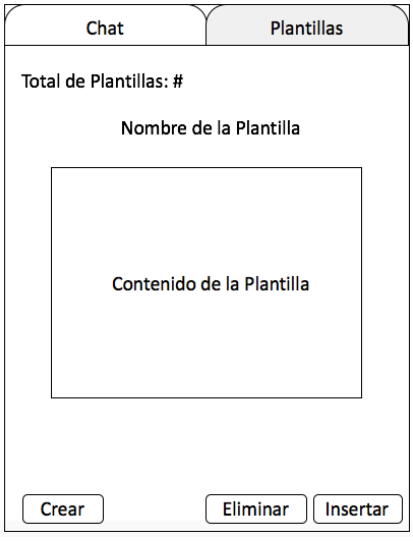

Figura 2.25 Diseño Área Multifuncional de Plantillas

El modal para agregar plantilla se diseñó como se muestra en la figura 2.26. Contiene 2 campos de texto, uno para introducir el nombre y el otro para introducir el contenido de la plantilla. El botón "Aceptar" confirma el registro de la plantilla en el proyecto.

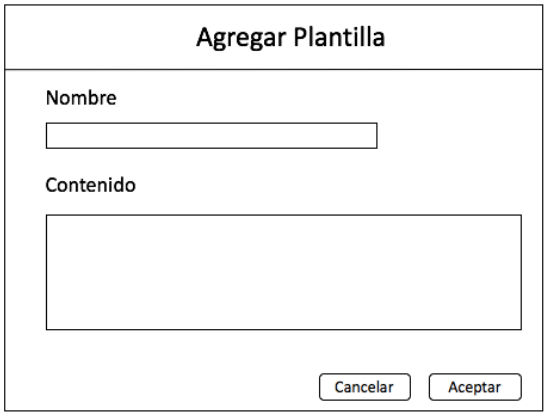

Figura 2.26 Modal para Agregar Plantillas

Área de Ejecución: Esta área muestra el resultado de la ejecución o compilación. En la figura 2.27 se muestra el diseño de esta área. El botón "Cerrar" cierra esta área para que no esté visible en el área de trabajo. Al correr el proyecto, muestra el resultado acompañado de la fecha y hora en el que se realizó la ejecución o compilación.

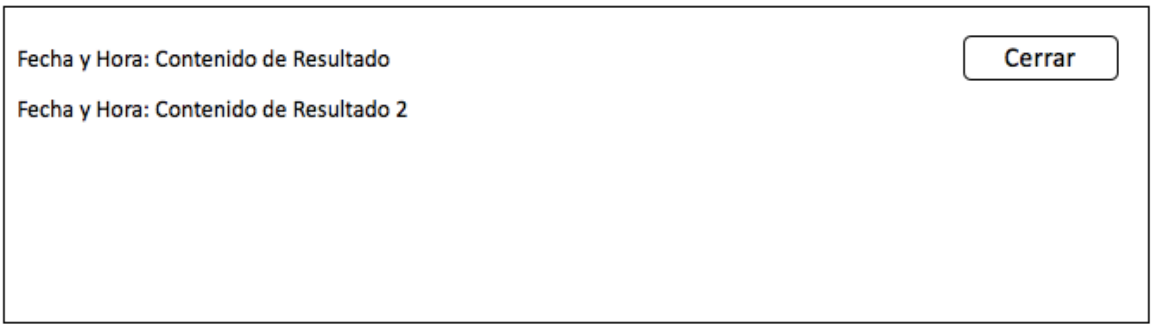

Figura 2.27 Diseño Área de Ejecución

Con esto se concluye el diseño gráfico de la aplicación a desarrollar.

Capítulo 3

Desarrollo del Sistema

3.1 Preparación Inicial

Se utilizó la interfaz de línea de comandos *Angular CLI* para facilitar el desarrollo del proyecto a través de comandos sencillos. Para la creación del proyecto únicamente fue necesario el comando "ng new". En la figura 3.1 se observan los archivos generados de manera automática en el editor de texto Visual Studio Code. *Angular CLI* permite crear plantillas de archivos que utiliza Angular como lo son los componentes, servicios, clases, interfaces y entre otros elementos necesarios para el desarrollo de aplicación a través de comandos (comando ejemplo "ng g plantilla nombre-plantilla"). Por defecto todos los archivos creados de esta manera, se generan dentro de la carpeta "app". Para crear una plantilla en otra ubicación se utiliza el comando "ng g plantilla ubicacion-deseada//nombre-plantilla".

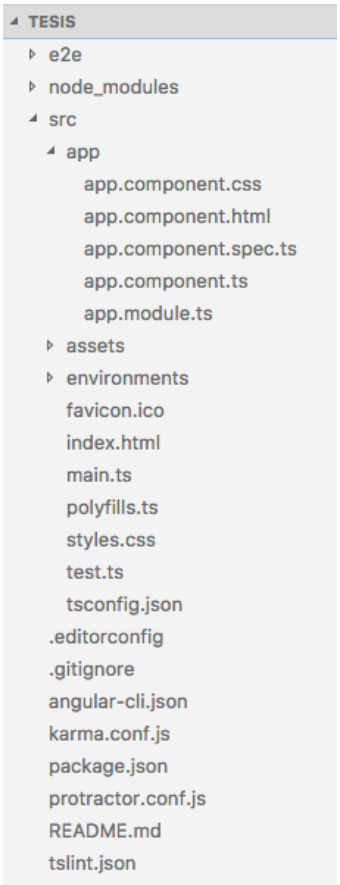

Figura 3.1 Archivos del Proyecto en Visual Studio Code

Los archivos relevantes dentro de la carpeta de proyecto son "index.html" y todos los archivos dentro de la carpeta "src". En el archivo "index.html" se agregaron todas las librerias externas y configuración necesaria para la aplicación. Otro archivo importante es el "app.module.ts", donde se guarda el conjunto de configuración e importación de librerias para poder hacer uso de ellas dentro de la aplicación.

Dentro de la cabecera del index.html, se agregó la etiqueta "<base href="/">" para tener referencia de la ruta por defecto. Como ruta por defecto se estableció la pantalla de inicio de sesión.

Para la interfaz de usuario, se utilizó el marco de trabajo de *Bootstrap 3.3.7* con el fin de facilitar la implementación de componentes gráficos de una manera estandarizada. Para poder hacer uso de las clases de *Bootstrap 3.3.7* en *Angular*, se incluyeron directamente los links de *CDN* en la cabecera de "index.html" (figura 3.2). Es importante tener en cuenta que se agregó el *CDN* de *jQuery* ya que *bootstrap 3.3.7* hace uso de esta librería para llevar a cabo algunas funciones como cuadros de diálogo (modal).

```
<head>

  <!-- jquery -->
  <script
  src="https://code.jquery.com/jquery-3.1.1.min.js"
  integrity="sha256-hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="
  crossorigin="anonymous"></script>

  <!-- Bootstrap 3.3.7-->
  <link rel="stylesheet"
  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
  integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
  crossorigin="anonymous">
```

Figura 3.2 Link de Referencia para Bootstrap 3.3.7 dentro del "index.html"

Para el uso de los servicios de *Firebase* también se colocaron *CDN*s dentro de la cabecera del index.html. En total se consumieron 3 servicios de los que ofrece *Firebase*, autenticación de usuario, base de datos en tiempo real y almacenamiento. Para la autenticación y base de datos fue necesario incluir el *CDN* correspondiente, y además se requirió agregar un *CDN* general y una configuración como se muestra en la figura 3.3.

```
<!-- agregar firebase y sus servicios-->
<script src="https://www.gstatic.com/firebasejs/3.6.8/firebase.js"></script>

<script src="https://www.gstatic.com/firebasejs/3.6.2/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/3.6.2/firebase-database.js"></script>

<script>
  // inicializar firebase
  var config = {
    apiKey: "AIzaSyB90aaOOTpQWqtoC_TGTsgNsyshitkjNLU",
    authDomain: "tesis-4b2ae.firebaseapp.com",
    databaseURL: "https://tesis-4b2ae.firebaseio.com",
    storageBucket: "tesis-4b2ae.appspot.com",
    messagingSenderId: "1080098635986"
  };
  firebase.initializeApp(config);
</script>
```

Figura 3.3 CDNs y Configuración de Firebase dentro del "index.html"

La configuración ayuda a conectar la aplicación web con el proyecto generado en la plataforma de *Firebase*.

Además de la instalación de *Firebase*, se utilizó la librería *Angular Fire 2*. Esta librería permite al usuario consumir el servicio de autenticación y de base de datos en tiempo real de *Firebase* de una manera fácil e interactiva. La librería se instaló a través de *npm* con el comando *npm install angularfire2 firebase --save* sobre la carpeta del proyecto. Una vez instalada la librería se agregó la configuración necesaria en el archivo "app.module.ts" como se puede observar en la figura 3.4 (líneas 10 a la 17) al igual que en el archivo "index.html".

```
6   import { AngularFireModule, AuthProviders, AuthMethods } from 'angularfire2';
7
8   import { AppComponent } from './app.component';
9
10  //configuración de firebase
11  export const firebaseConfig = {
12    apiKey: "AIzaSyB90aaOOTpQWqtoC_TGTsgNsyshitkjNLU",
13    authDomain: "tesis-4b2ae.firebaseapp.com",
14    databaseURL: "https://tesis-4b2ae.firebaseio.com",
15    storageBucket: "tesis-4b2ae.appspot.com",
16    messagingSenderId: "1080098635986"
17  };
18
19  //métodos de autenticación de firebase
20  export const firebaseAuthConfig = {
21    provider: AuthProviders.Password,
22    method: AuthMethods.Password
23  }
24
25  @NgModule({
26    declarations: [
27      AppComponent
28    ],
29    imports: [
30      BrowserModule,
31      FormsModule,
32      HttpModule,
33      AngularFireModule.initializeApp(firebaseConfig, firebaseAuthConfig)
```

Figura 3.4 Instalación de la Librería Angular Fire 2 en el archivo "app.module.ts"

En la línea 6 se hace la importación del módulo principal "AngularFireModule", el módulo de provedoores de autenticación "AuthProviders" y el módulo que define el método de autenticación "AuthMethods". En la línea 11 se exporta la configuración de *Firebase*. Y de la misma forma en la línea 20 se exporta la configuración de autenticación, en esta configuración se establece como proveedor y método "Password", lo cuál significa que se hará uso el sistema de correo y contraseña para registrar y autenticar usuarios dentro de la aplicación. Finalmente se importan ambas configuraciones en el módulo (línea 33), habilitando el uso de la librería en todos los componentes que pertenezcan en "AppModule".

3.2 Desarrollo del Módulo Raíz

Como en cualquier estructura de proyecto informático, es importante tener separado cada módulo según su funcionalidad, ya que tener separados los módulos en *Angular*, ayuda a crear un proyecto ordenado, estructurado, fácil de mantener y escalable. Para una mejor comprensión del proyecto, se utilizaron las nomenclaturas que aparecen en la tabla 3.1 para la creación de archivos dentro del proyecto.

Tabla 3.1 Nomenclaturas Empleadas para los Componentes

Tipo de Archivo	Nomenclatura
Módulo	nombre-del-archivo.module.ts
Componete	nombre-del-archivo.component.ts
Servicio	nombre-del-archivo.service.ts
HTML	nombre-del-archivo.component.html
CSS	nombre-del-archivo.component.css
Modelo	nombre-del-archivo.ts

Para esta aplicación, se diseñaron 4 módulos principales: raiz, login (inicio de sesión), consola de proyectos y área de trabajo. El módulo de raiz es fundamental ya que funciona para interconectar los diferentes componentes del proyecto. Este módulo contiene únicamente un componente, el de aplicación, que se encarga de arrancar el sistema y administrar todos los recursos necesarios para llevar a cabo un funcionamiento adecuado de la aplicación. El componente princiapl componente app se definió como se muestra en la figura 3.5.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
}
```

Figura 3.5 Definición del Componente App

Es una gran ventaja hacer uso de servicios en *Angular*, un modelo lógico que permite proporcionar funciones e información desde cualquier parte de la aplicación e incluso de servicios externos (desde servidores y/o servicios web). Para consumir los servicios externos, se crearon servicios que permiten hacer el consumo de ellos a través del protocolo http. Para hacer uso de http en *Angular*, es necesario importar el módulo “HttpModule”, como se puede ver en la figura 3.6 (línea 4 y 17). Todos los módulos que se requieran utilizar en *Angular* se tienen que importar dentro del decorador “@NgModule” en el arreglo de “imports”, así habilitando el uso de los módulos (en este caso “HttpModule”) en todo el módulo “AppModule”.

En el módulo raíz se creó un servicio denominado “DatosService” como se puede observar en las líneas 8 y 20 dentro del archivo “app.module.ts”, el cuál se encarga de mantener información persistente como la del usuario y proyecto seleccionado a través de la aplicación. Todos los servicios declarados en el proyecto se importan colocándolos dentro del decorador “@NgModule” en el arreglo de “providers”. Todos los servicios del proyecto se crean con el comando “ng g service nombre-servicio” con *Angular CLI*.

```
1   import { BrowserModule } from '@angular/platform-browser';
2   import { NgModule } from '@angular/core';
3   import { FormsModule } from '@angular/forms';
4   import { HttpModule } from '@angular/http';
5
6   import { AppComponent } from './app.component';
7
8   import { DatosService } from './datos.service';
9
10  @NgModule({
11    declarations: [
12      AppComponent
13    ],
14    imports: [
15      BrowserModule,
16      FormsModule,
17      HttpModule
18    ],
19    providers: [
20      DatosService
21    ],
22    bootstrap: [AppComponent]
23  })
24  export class AppModule { }
```

Figura 3.6 Importación de la Librería “HttpModule” en “app.module.ts”

Además de los servicios, otra ventaja que proporciona el marco de trabajo *Angular*, es el concepto de routing, un modelo lógico que ayuda a implementar la navegación entre distintos componentes (pantallas o interfaces gráficas) dentro de la aplicación. La definición del routing permite hacer el uso de los componentes definidos de una manera sencilla. Para este proyecto la configuración del routing se definió como aparece en la figura 3.7 dentro del archivo “app.module.ts”.

Se definieron en total seis rutas diferentes, cinco que corresponden a una pantalla diferente y la ruta por defecto que dirige al usuario a la pantalla de inicio de sesión. Las pantallas de consola de proyectos y área de trabajo tienen una protección de navegación “canActivate” del servicio “ProtegerLoginService”, lo cual significa que no permite navegaciones hacia esas pantallas si el usuario no se encuentra autenticado. Este punto se cubrirá en el módulo de login con detalle.

```
const Rutas: Routes = [
  { path: 'login', component: LoginInicioComponent },
  { path: 'registro', component: LoginRegistroComponent },
  { path: 'login/recuperar', component: LoginRecuperarComponent },
  { path: 'consola', component: ConsolaProyectosComponent, canActivate:[ProtegerLoginService] },
  { path: 'area-trabajo', component: AreaTrabajoComponent, canActivate:[ProtegerLoginService] },
  { path: '', redirectTo: '/login',pathMatch: 'full' }
];
```

Figura 3.7 Definición del routing dentro de “app.module.ts”

En la tabla 3.2 se pueden observar las direcciones con su pantalla correspondiente (La dirección *http://localhost:4200* solamente es de ejemplo).

Tabla 3.2 Nomenclaturas Empleadas para las Rutas

Ruta	Pantalla correspondiente
http://localhost:4200/login	Pantalla de inicio de sesión
http://localhost:4200/registro	Pantalla de registro de nuevo usuario
http://localhost:4200/login/recuperar	Pantalla de recuperación de contraseña
http://localhost:4200/consola	Pantalla de consola de proyectos
http://localhost:4200/area-trabajo	Pantalla de área de trabajo
http://localhost:4200/ (Ruta por defecto)	Pantalla de inicio de sesión

Para poder cargar las rutas de manera propia dentro de la aplicación, es necesario hacer el uso de la etiqueta “<router-outlet></router-outlet>”. El archivo “app.component.html” contiene únicamente la etiquta “<router-outlet></router-outlet>”. *Angular* carga la ruta necesaria dentro de la etiqueta. De manera inicial (cuando arranca la aplicación), se carga la ruta por defecto (pantalla de inicio se sesión).

3.3 Desarrollo del Módulo de Login

Este módulo se conforma de 3 componentes: el de inicio de sesión que se encarga de la autenticación del usuario, registro de usuario que permite dar de alta a un usuario nuevo y recuperación de contraseña que permite cambiar la contrasñea al usuario que no recuerde su contraseña; y 2 servicios, los cuales reciben el estado de autenticación. El servicio de login se encarga de estar escuchando el estado de autenticación constantemente para saber si mantener al usuario dentro de la aplicación, y otro servicio que protege el acceso por usuarios no autenticados. Cada componente y servicio se nombró como se muestra en la tabla 3.3.

Tabla 3.3 Componentes y Servicios del Módulo de Login

Función	Nombre del Archivo
Interfaz gráfica de autenticación del usuario	login-inicio.component.ts
Interfaz gráfica de registro de un nuevo usuario	login-registro.component.ts
Interfaz gráfica de recuperación de la contraseña	login-recuperar.component.ts
Servicio que guarda el estado de autenticación	login.service.ts
Servicio que protege la navegación de un usuario no autenticado	proteger-login.service.ts

Cada componente se llama desde el componente app como se muestra en la figura 3.8. El usuario únicamente navega hacia los diferentes componentes a través del router.

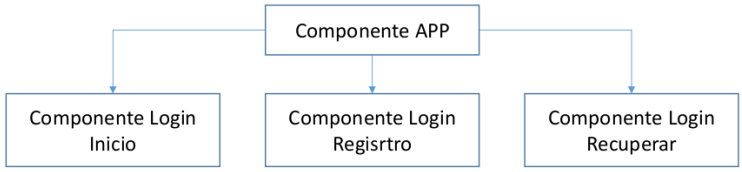

Figura 3.8 Relación de los Componentes del Módulo Login

3.3.1 Desarrollo del Componente Login Inicio

El primer componente que se creó es el componente login inicio. El contenido gráfico de este componente se encuentra en un archivo externo llamado “login-inicio.component.html”. La presente tesis omite el código que no se considere relevante de los archivos html que aparezcan.

3.3.1.1 Desarrollo Lógico del Componente Login Inicio

Cuando se crea un componente en *Angular*, se define un selector correspondiente al componente creado para poder hacer uso del mismo como una etiqueta de html. En el caso de este componente, el selector se definió como “<app-login-inicio></ app-login-inicio >”. Esto significa que cualquier componente de la aplicación puede hacer el uso del selector (etiqueta) “<app-login-inicio></ app-login-inicio >”.

Como se puede observar en la figura 3.9, se muestran las importaciones de los módulos necesarios desde la línea 1 a la línea 4. En la línea 7 se encuentra la definición del selector y en la línea 8 la definición del archivo html con el que se asocia ("login-inicio.component.html").

El módulo "Router" permite al componente obtener el estado de navegación y navegar hacia otros componentes. El servicio "AngularFire" se utiliza para crear una instancia y poder utilizar métodos de autenticación del servicio de autenticación de *Firebase*. Para poder usar ambos servicios ("Router" y "AngularFire") dentro de este componente, se declararon dos variables en el constructor del componente, las variables "router" y "af" como aparece en la línea 17 y 18 respectivamente.

De la línea 14 a la línea 18 se pueden encontrar las tres variables que maneja el componente. La variable "autenticacion_fallida" guarda un valor booleano para mostrar un mensaje de error. Las variables "valor_correo" y "valor_contrasena" guardan lo que el usuario introduce en el campo de texto de inicio de sesión. En la línea 26 se suscribe el estado de autenticación, lo cuál significa que en cuanto cambie el estado de autenticación del usuario (aprobado o no) se ejecuta el código de la línea 27 a la línea 31. Si el usuario entra a la aplicación por primera vez, lo mantiene dentro de la pantalla de inicio de sesión.

```
1   import { Component, OnInit } from '@angular/core';
2   import { Router } from '@angular/router';
3
4   import { AngularFire } from 'angularfire2';
5
6   @Component({
7       selector: 'app-login-inicio',
8       templateUrl: './login-inicio.component.html',
9       styleUrls: ['./login-inicio.component.css']
10  })
11
12  export class LoginInicioComponent implements OnInit {
13
14      autenticacion_fallida = false;
15
16      //para ngModel
17      valor_correo = null;
18      valor_contrasena = null;
19
20      constructor(
21          private router: Router,
22          private af: AngularFire
23      ) {
24
25          //monitorea el cambio de estado de autenticación
26          this.af.auth.subscribe(auth => {
27              if(auth) {
28                 this.navegar_consola();
29              } else {
30
31              }
32          });
33
34       }
```

Figura 3.9 Importaciones, Variables y Constructor en "login-inicio.component.ts"

Este componente tiene dos métodos como se muestra en la figura 3.10, "autenticar_usuario()" que verifica si el usuario existe y "navegar_consola()" que navega hacia la pantalla de consola de proyectos (componente consola proyectos).

Dentro del método "autenticar_usuario()" se usa la variable "af" para usar el método "auth.login()", este método recibe como parámetros la variable "volor_correo" y "valor_contrasena", de esta manera el servicio de *Firebase* se encarga en corroborar la existencia del usuario y la contraseña dentro de su servicio. En el caso de tener éxito con la autenticación, se ejecuta el método "navegar_consola()" (línea 46), de no ser así se guarda un falso en la variable "autenticacion_fallida" (línea 51), para mostrar un mensaje de error al usuario.

```
40      autenticar_usuario() {
41
42          //autenticacion con el servicio de angular fire
43          this.af.auth.login({email: this.valor_correo,password: this.valor_contrasena}).then(
44              (success) => {
45
46              this.navegar_consola();
47
48          }).catch(
49              (err) => {
50
51              this.autenticacion_fallida = true;
52
53          });
54
55      }
56
57      navegar_consola() {
58          this.router.navigateByUrl('/consola');
59      }
```

Figura 3.10 Métodos de "login-inicio.component.ts"

3.3.1.2 Desarrollo Gráfico del Componente Login Inicio

A continuación se explica la conexión del archivo "login-inicio.component.ts" y "login-inicio.component.html". Es importante tener presente que *Angular* ofrece la dinámica que todos los elementos que se encuentran dentro del archivo html pueden utilizar las variables y métodos que están definidos en su componente asociado (archivo con la extensión "ts").

Como se había mencionado, el componente login inicio ("login-inicio.component.ts") tiene asociado el archivo "login-inicio.component.html" para la elaboración del contenido gráfico (pantalla o interfaz de usuario). El resultado del archivo es lo que se muestra en la figura 3.11.

Esta pantalla tiene un formulario que contiene dos campos de texto para la introducción del usuario y contraseña. Lo que el usuario introduzca se guarda en la variable "valor_correo" y "valor_contrasena" respectivamente. También tiene tres botones, el botón "Recuperar Contraseña" que permite navegar hacia el componente login recuperar; el botón "Iniciar Sesión" que ejecuta el método "autenticar_usuario()" y el botón "Registrarse" que permite navegar hacia la pantalla de registro de usuario (componente login inicio).

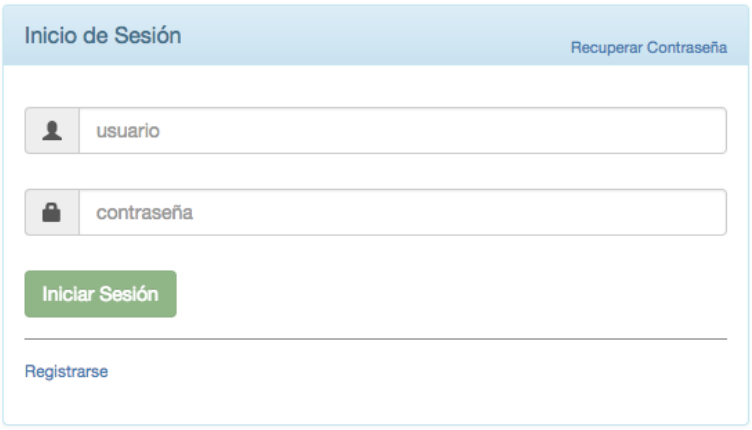

Figura 3.11 Apariencia de la Pantalla de Inicio de Sesión

En la figura 3.12 se muestra cómo se definió el botón de "Recuperar Contraseña". Dentro de la etiqueta de la liga "<a>" se utilizó la directiva "routerLink" para permitir navegar hacia la ruta "/login/recuperar", la cuál está asociada con el componente login recuperar (pantalla de recuperar contraseña).

```
10      <div class="panel-heading">
11          <div class="panel-title">Inicio de Sesión</div>
12          <div style="float:right; font-size: 80%; position: relative; top:-10px">
13              <a routerLink="/login/recuperar">Recuperar Contraseña</a>
14          </div>
15      </div>
```

Figura 3.12 Definición del Botón "Recuperar Contraseña" en "login-inicio.component.html"

Los dos campos de texto se definieron como se muestra en la figura 3.13. A través de la directiva "ngModel" se vincula la variable "valor_correo" como se puede ver en la línea 33, lo cuál significa que cualquier introducción de texto por parte del usuario en el campo "usuario" se almacena en la variable "valor_correo" de manera síncrona. Para el campo "contraseña" también se empleó la misma lógica, la variable "valor_contrasena" se vinculó a través de la directiva "ngModel" de la misma manera (línea 40).

```
29  <!-- campo de texto usuario -->
30  <div style="margin-bottom: 25px" class="input-group">
31      <span class="input-group-addon"><i class="glyphicon glyphicon-user"></i></span>
32      <input id="login-username" type="text" class="form-control" placeholder="usuario"
33      required name="usuario" [(ngModel)]="valor_correo">
34  </div>
35
36  <!-- campo de texto contrasena -->
37  <div style="margin-bottom: 5px" class="input-group">
38      <span class="input-group-addon"><i class="glyphicon glyphicon-lock"></i></span>
39      <input id="login-password" type="password" class="form-control" placeholder="contraseña"
40      required name="contrasena" [(ngModel)]="valor_contrasena">
41  </div>
```

Figura 3.13 Definición del Campo de Texto "usuario" y "contraseña" en "login-inicio.component.html"

El botón de "Iniciar Sesión" llama el método "autenticar_usuario()" mencionado anteriormente como aparece en la línea 49 de la figura 3.14.

La directiva "disabled" (línea 48) funciona como control de validación para no permitir la llamada del método "autenticar_usuario()" con valores nulos en la variable "valor_correo" y "valor_contrasena". En el caso de detectar valores nulos en las variables, el botón se deshabilita (línea 48).

```
43              <!-- botón de inicio de sesión -->
44              <div style="margin-top:25px" class="form-group">
45
46                  <div class="col-sm-12 controls">
47                      <button
48                      [disabled]="valor_correo == null || valor_contrasena == null"
49                      (click)="autenticar_usuario()"
50                      id="btn-login" class="btn btn-success">
51                      Iniciar Sesión
52                      </button>
53                  </div>
54              </div>
```

Figura 3.14 Definición del Botón "Iniciar Sesión" en "login-inicio.component.html"

El último botón "Registrarse", se definió como aparece en la figura 3.15.

Al igual que el botón "Recuperar Contraseña", en la liga de "<a>" contiene la directiva "routerLink" para permitir al usuario navegar hacia la ruta "/registro" que está asociada con el componente login registro.

```
56    <div class="form-group">
57        <div class="col-md-12 control">
58            <div style="border-top: 1px solid #888; padding-top:15px; font-size:85%" >
59                <a routerLink="/registro">Registrarse</a>
60            </div>
61        </div>
62    </div>
```

Figura 3.15 Definición del Botón "Registrarse" en "login-inicio.component.html"

Como último elemento del componente se encuentra el mensaje de error. Que tiene la siguiente apariencia como se muestra en la figura 3.16.

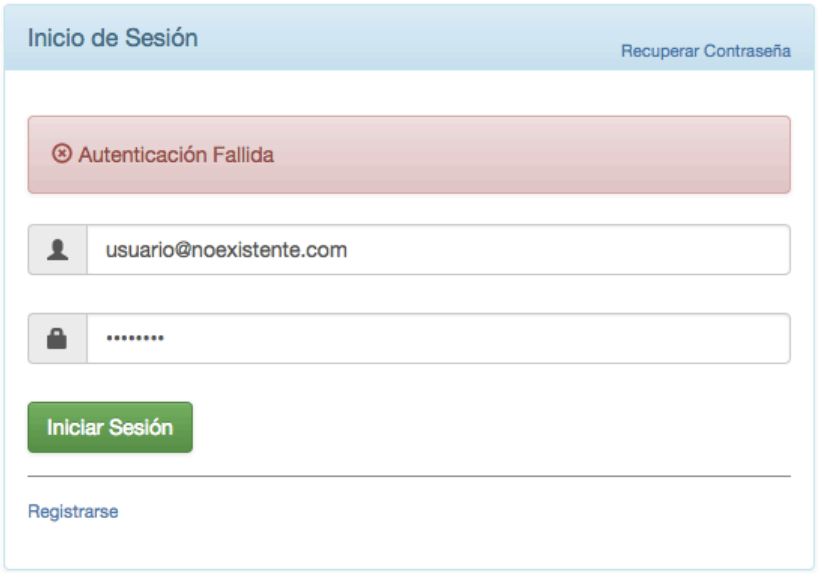

Figura 3.16 Apariencia de la Pantalla de Inicio de Sesión con Mensaje de Error

El mensaje de error se definió como aparece en la figura 3.17. Como se menció anteriormente a la variable "autenticacion_fallida" se le asigna un valor falso en el caso de que el método "autenticar_usuario()" no tenga éxito. La directiva que aparece en la línea 24 "ngIf" comprueba el valor de la variable "autenticacion_fallida" de manera constante para validar la aparición del mensaje de error "Autenticación Fallida".

```
24    <div class="alert alert-danger"  *ngIf="autenticacion_fallida">
25        <span class="glyphicon glyphicon-remove-circle" aria-hidden="true"></span>
26        Autenticación Fallida
27    </div>
```

Figura 3.17 Definición del Mensaje de Error "Autenticación Fallida"

Con esto se concluye el desarrollo del componente login

3.3.2 Desarrollo del Componente Login Registro

A continuación se detalla la construcción del componente login registro. Este componente se encarga de proporcionar un formulario al usuario para que pueda generar una nueva cuenta, generando un nuevo registro de usuario en la base de datos de *firebase*.

3.3.2.1 Desarrollo Lógico del Componente Login Registro

En la figura 3.18 se pueden observar las importaciones y variables que contiene el componente. Para este componente se definió como selector "app-login-registro" (línea 6) y "login-registro.component.html" (línea 7) para el archivo asociado de html.

```
1  import { Component, OnInit } from '@angular/core';
2  import { AngularFire,  FirebaseObjectObservable } from 'angularfire2';
3  import { Usuario } from '../../modelos/usuario';
4
5  @Component({
6    selector: 'app-login-registro',
7    templateUrl: './login-registro.component.html',
8    styleUrls: ['./login-registro.component.css']
9  })
10 export class LoginRegistroComponent implements OnInit {
11
12    usuario: Usuario = null;
13    referencia: FirebaseObjectObservable<any>;
14
15    //valores a enviar
16   valor_correo = null;
17   valor_nombre = null;
18   valor_contrasena = null;
19   valor_contrasena_confirmacion = null;
20
21   //valores booleanos
22   coinciden_contrasenas = false;
23   usuario_existente = false;
24   correo_invalido = false;
25   usuario_registrado = false;
26
27   constructor(
28     private af:AngularFire
29   ) { }
```

Figura 3.18 Importaciones y Variables en "login-registro.component.ts"

De la línea 1 a la línea 3 se importaron las clases necesarias en el componente. Es importante señalar que en la línea 2 se importó la clase "FirebaseObjectObservable", que se emplea para guardar el objecto resultado de la consulta de la base de datos de *Firebase*. La clase "Usuario" (línea 3) fue importado para crear la instancia del objeto a insertar dentro de la base de datos. La clase "Usuario" es una clase personalizada como se puede observar en la figura 3.19.

```
1  export class Usuario {
2
3      public nombre: string;
4      public correo: string;
5      public id: string;
6      public imgurl: string;
7
8  }
```

Figura 3.19 Definición de la Clase "Usuario" en "usuario.ts"

El usuario tiene 4 atributos; el atributo id que funciona como llave primaria en la base de datos, el correo que es la llave primaria candidata que permite asociar la introducción del usuario con el objeto usuario, el nombre que se muestra en la consola de proyectos y el imgurl que guerda la liga de foto de perfil del usuario. Al crearse un objeto usuario, se le asigna una liga de imagen por defecto.

En total se declararon diez variables dentro del componente. Variable "usuario" (línea 12) para guardar el objeto "Usuario"; variable "referencia" (línea 13) que guarda la consulta del

objeto de la base de datos; cuatro variables de "valor_campo" que guardan el contenido que introduce el usuario y cuatro variables que guardan valores booleanos que representan el estado de registro del usuario para mostrar ciertas alertas en la pantalla.

El componente login registro tiene dos métodos, uno es "verificar_coincidencia()" (figura 3.20). Este método se activa cada vez que el usuario introduce un carácter en el campo de texto para la contraseña y confirmación de la contraseña. Compara si la variable "valor_contrasena" y "valor_contrasena_confirmacion" tienen el mismo valor, de ser así, la variable "coinciden_contrasenas" recibe el valor de verdadero.

```
34      verificar_coincidencia() {
35
36        if (this.valor_contrasena === this.valor_contrasena_confirmacion) {
37          this.coinciden_contrasenas = true;
38        }
39        else {
40          this.coinciden_contrasenas = false;
41        }
42
43      }
```

Figura 3.20 Método "verificar_coincidencia()" en "login-registro.component.ts"

El método restante del componente es el método "registrar_usuario()" (figura 3.21). Utiliza el método integrado de la librería "AngularFire", para crear usuarios. En la línea 47 se puede observar el método integrado "auth.createUser()", este método recibe un objeto que contiene dos campos, el campo de correo (email) y contraseña (password).

Con el método "then()" (línea 47) se puede obtener el resultado del usuario creado o el error que retorna el servidor de *Firebase*.

En el caso de tener éxito con la creación del usuario, se obtiene el id que le asigna *Firebase* a la variable "user" para agregarlo en la variable "usuario" (línea 49). Además del id se le asigna el contenido de la variable "valor_nombre" y "valor_correo" como se puede ver en la línea 50 y línea 51 respectivamente. De manera predeterminada se asigna el valor "vacio" en el campo de "imgurl".

Una vez guardados los campos de la variable "usuario", se guarda la referencia "/users/user.id (id del usuario que se acaba de crear)" de la base de datos en la variable "referencia" (línea 55). Después se inserta la variable "usuario" (instnacia del objeto "Usuario") en la base de datos en la ubicación que apunta la variable "refernecia" con el método "set()" (línea 56). Finalmente en la variable "usuario_existente" guarda un falso y en la variable "usuario_registrado" guarda un verdadero. Estas dos variables sirven para ocultar y mostrar alertas que contiene el componente.

En el caso de detectar un error al tratar de crear un usuario, se ejecuta el código de las líneas 62 a la 73. *Firebase* retorna dos tipos de errores principalmente al tratar de crear un usuario, con un correo ya registrado en *Firebase* o hacer uso de un correo no válido (no contiene @ y/o .com).

La variable de error que retorna *Firebase* tiene un campo "code" que contiene el código de error detectado en el servidor. Para poder leer ese código y comparar el contenido, es necesario convertir la variable de error en un objeto JSON con el método "JSON.parse()" y "JSON.stringify()".

En la línea 65 se convierte la variable "err" en un objeto JSON para poder comparar el campo "code" con el contenido "auth/email-already-in-use", este código representa que el error fue a causa de que el correo ya está en uso. Se guarda un verdadero en la variable "usuario_existente" para mostrar su alerta correspondiente.

Para detectar el error restante se compara el campo "code" con el contenido "auth/invalid-email" (línea 69), este código significa que el correo utilizado para crear el usuario es inválido. Se guarda un verdadero en la variable "correo_invalido" para mostrar su alerta correspondiente.

```
45    registrar_usuario() {
46
47        this.af.auth.createUser({ email: this.valor_correo, password: this.valor_contrasena }).then(
48            (user) => {
49                this.usuario = {
50                  nombre: this.valor_nombre,
51                  correo: this.valor_correo,
52                  id: user.uid,
53                  imgurl: "vacio"
54                }
55                this.referencia = this.af.database.object('/users/' + user.uid);
56                this.referencia.set(this.usuario);
57
58                this.usuario_existente = false;
59                this.usuario_registrado = true;
60
61        }).catch(
62            (err) => {
63
64              console.log(err);
65              if(JSON.parse(JSON.stringify(err)).code === 'auth/email-already-in-use') {
66                this.usuario_existente = true;
67              }
68
69              if(JSON.parse(JSON.stringify(err)).code === 'auth/invalid-email') {
70                this.correo_invalido = true;
71              }
72
73        });
74
75    }
```

Figura 3.21 Método "registrar_usuario()" en "login-registro.component.ts"

3.3.2.2 Desarrollo Gráfico del Componente Login Registro

A continuación se explica el desarrollo del contenido gráfico del componente login registro.

El resultado del archivo "login-registro.component.html" tiene la apariencia que se muestra en la figura 3.22.

Esta pantalla tiene un formulario con cuatro campos de texto y tres botones (uno está escondido por defecto).

Figura 3.22 Apariencia del Componente Registro

El primer botón que se situa en la cabecera del panel del formulario, se definió como se muestra en la figura 3.23. El elemento contiene la directiva "routerLink" que permite al usuario navegar hacia el componente login inicio (pantalla de inicio de sesión).

```
7     <!-- panel -->
8     <div class="panel panel-info">
9         <div class="panel-heading">
10            <div class="panel-title">Registro de usuario</div>
11            <div style="float:right; font-size: 85%; position: relative; top:-10px">
12                <a id="signinlink" routerLink="/login">Iniciar sesión</a>
13            </div>
14        </div>
```

Figura 3.23 Definición de botón "Iniciar sesión" en "login-registro.component.html"

En la figura 3.24 se muestra la definición del campo "Correo". En la línea 65 se define la variable local "correo", esta variable se vincula con la variable "valor_correo" definida en el archivo "login-registro.component.ts". El atributo "required" permite establecer un control sobre este elemento (input). Si este elemento está vacío, el atributo "correo.errors" regresa en valor booleano falso.

```
60    <div class="form-group">
61        <label for="correo" class="col-md-3 control-label">Correo</label>
62        <div class="col-md-9">
63            <input
64            type="text" class="form-control" name="correo" id="correo" placeholder="Correo"
65            required  [(ngModel)]="valor_correo" #correo="ngModel">
66        </div>
67    </div>
68
```

Figura 3.24 Definición del Campo de Texto "Correo" en "login-registro.component.html"

El campo de texto "Nombre" se definió casi de la misma manera (figura 3.25). Solo que además de la vinculación y establecimiento del atributo "required", se colocó el atributo "minlength" (línea 74). Este atributo agrega una regla más en el control del formulario, como

el valor del atributo “minlength” para este elemento es de 3, si el usuario no introduce más de 3 caracteres en el campo de texto, el atributo “nombre.errors” regresa un valor booleano falso.

```
69    <div class="form-group">
70        <label for="nombre" class="col-md-3 control-label">Nombre</label>
71        <div class="col-md-9">
72            <input
73            type="text" class="form-control"  name="nombre" id="nombre" placeholder="Nombre Completo"
74            required minlength="3" [(ngModel)]="valor_nombre" #nombre="ngModel">
75        </div>
76    </div>
```

Figura 3.25 Definición del Campo de Texto “Nombre” en “login-registro.component.html”

Para el campo de texto de “Contraseña”, se utilizó el tipo “password” para no permitir visualizar la introducción del usuario (figura 3.26). Este elemento también está vinculado con su variable correspondiente y tiene 2 controles de formulario, además tiene asociada un método llamado “varificar_coincidencia()” que se llama cada vez que el usuario introduce un carácter en el campo de texto. Como se había mencionado, este método ayuda a corroborar la coincidencia de la variable “valor_contrasena” y “valor_contrasena_confirmacion” .

```
78    <div class="form-group">
79        <label for="contrasena" class="col-md-3 control-label">Contraseña</label>
80        <div class="col-md-9">
81            <input
82            type="password" class="form-control" name="contrasena" id="contrasena" placeholder="Contraseña"
83            required minlength="6" [(ngModel)]="valor_contrasena" #contrasena="ngModel"
84            (keyup)="verificar_coincidencia()"
85            >
86        </div>
87    </div>
```

Figura 3.26 Definición del Campo de Texto “Contraseña” en “login-registro.component.html”

El último campo es el de “Confirmación Contraseña”, este campo se definió casi de la misma manera que la del campo de “Contraseña” (figura 3.27). La única diferencia que tiene es el nombre de la variable local, que se definió como “contrasena2”.

```
89    <div class="form-group">
90        <label for="contrasena" class="col-md-3 control-label">Confirmar</label>
91        <div class="col-md-9">
92            <input
93            type="password" class="form-control" name="contrasena2" id="contrasena2" placeholder="Contraseña"
94            required minlength="6" [(ngModel)]="valor_contrasena_confirmacion" #contrasena2="ngModel"
95            (keyup)="verificar_coincidencia()"
96            >
97        </div>
98    </div>
```

Figura 3.27 Definición del Campo de Texto “Contraseña Confirmación” en “login-registro.component.html”

La pantalla en total tiene 7 alertas (mensajes de error) definidas en “login-registro.component.html”. A continuación, se explica a detalle la lógica que se empleo para cada alerta y su definición.

La primera alerta que se definió es la del campo de "Correo" como se muestra en la figura 3.28. En la línea 21 se utilizó la directiva "ngIf" para agregar el elemento en el *DOM* en caso de cumplir con alguna de las 2 condiciones establecidas. La primera condición a comparar es si el campo de texto tiene algún error, esto se compara con el atributo "correo.errors". La segunda condición se cumple si el campo de texto ha sido modificado ("correo.dirty") o si el campo de texto ha sido visitado por el usuario ("correo.touched").

```
19    <div
20    id="alerta1"
21    *ngIf="correo.errors && (correo.dirty || correo.touched)"
22    class="alert alert-danger">
23        <div [hidden]="!correo.errors.required">
24        Introduce un correo
25        </div>
26    </div>
```

Figura 3.28 Definición de la Alerta 1 en "login-registro.component.html"

Una vez que se cumpla la condición de la línea 21, el mensaje con el contenido "Introduce un Correo" está escondido si el atributo "correo.errors.required" tiene un valor booleano falso (línea 23). La alerta tiene la apariencia de la figura 3.29.

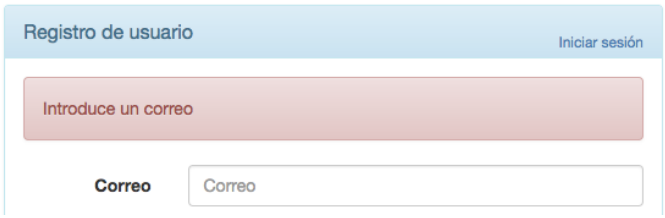

Figura 3.29 Apariencia de la Alerta 1

La segunda alerta está definida como se muestra en la figura 3.30. La definición es similar a la primera alerta, pero con 2 opciones de mensaje. El mensaje "Introducir un nombre" aparece si el campo de texto está vacío (línea 32) y el otro mensaje aparece si el campo de texto no detecta más de 3 caracteres (línea35). No pueden aparecer los 2 mensajes al mismo tiempo ya que el atributo "minlength" depende del atributo "requiered".

```
28    <div
29    id="alerta2"
30    *ngIf="nombre.errors && (nombre.dirty || nombre.touched)"
31    class="alert alert-danger">
32        <div [hidden]="!nombre.errors.required">
33        Introducir un nombre
34        </div>
35        <div [hidden]="!nombre.errors.minlength">
36        El nombre tiene que ser de mínimo 3 caracteres
37        </div>
38    </div>
```

Figura 3.30 Definición de la Alerta 2 en login-registro.component.html

La tercera y cuarta alerta se definieron como se muestra en la figura 3.31. La cuarta alerta no compara los controladres integrados de *Angular*, compara únicamente si las contraseñas introducidas coinciden con la variable "coinciden_contrasenas" (línea 55).

```
40      <div
41      id="alerta3"
42      *ngIf="contrasena.errors && (contrasena.dirty || contrasena.touched)"
43      class="alert alert-danger">
44          <div [hidden]="!contrasena.errors.required">
45          Introducir una contraseña
46          </div>
47          <div [hidden]="!contrasena.errors.minlength">
48          La contraseña tiene que ser de mínimo 6 caracteres
49          </div>
50      </div>
51
52      <div
53      id="alerta4"
54      *ngIf="(contrasena2.dirty || contrasena2.touched)">
55          <div *ngIf="!coinciden_contrasenas" class="alert alert-danger">
56          Las contraseñas no coinciden
57          </div>
58      </div>
```

Figura 3.31 Definición de la Alerta 3 y 4 en "login-registro.component.html"

De la quinta a la séptima alerta se definieron como se puede observar en la figura 3.32.

```
113   <div
114   id="alerta5"
115   *ngIf="usuario_existente"
116   class="alert alert-danger">
117       Usuario ya existente, utiliza otro correo
118   </div>
119
120   <div
121   id="alerta6"
122   *ngIf="correo_invalido"
123   class="alert alert-danger">
124       Correo inválido, utilzia un correo válido
125   </div>
126
127   <div
128   id="alerta7"
129   *ngIf="usuario_registrado"
130   class="alert alert-success">
131       Usuario registrado exitosamente
132   </div>
```

Figura 3.32 Definición de la Alerta 5, 6 y 7 en "login-registro.component.html"

Estas alertas comparan únicamente el estado de las variables booleanas definidas en "login-registro.component.ts" (línea 115, 122 y 129). Estas alertas se situan de bajo del botón de "Registrarse" (figura 3.33).

Figura 3.33 Apariencia de la Alerta 5 en "login-registro.component.html"

El tercer botón se definió como aparece en la figura 3.34. Este botón únicamente aparece si la variable "usuario_registrado" tiene un valor booleano verdadero (línea 135).

```
134        <!-- botón de navegación -->
135        <div class="form-group" *ngIf="usuario_registrado" >
136            <div class="col-md-offset-3 col-md-9">
137                <button id="btn-navegar-consola" type="button" class="btn btn-success" routerLink="/consola">
138                 Navegar hacia la consola
139                </button>
140            </div>
141        </div>
```

Figura 3.34 Definición del Botón "Nevagr hacia la consola" en "login-registro.component.html"

Tiene la directiva "routerLink" para navegar hacia el componente consola proyectos (línea 137). Este botón siempre aparece junto con la séptima alerta debido a que ambas comparan el estado de la variable "usuario_registrado" (figura 3.35).

Figura 3.35 Apariencia del Botón "Nevagr hacia la consola" y Alerta 7

3.3.3 Desarrollo del Componente Login Recuperar

El último componente que pertenece a este módulo es el componente login recuperar. Este componente proporciona una interfaz gráfica al usuario para que pueda recuperar la contraseña olvidada a través de su correo. Al usuario se le envía un correo para restablecer una nueva contraseña.

3.3.3.1 Desarrollo Lógico del Componente Login Recuperar

El componente loging recuperar se definió como se muestra en la figura 3.36.

Como selector se difinió "app-login-recuperar" y "login-component.html" como el archivo de html (línea 4 y 5).

Este componente contiene 2 variables y 2 métodos. La variable "valor_correo" almacena el correo que introdujo el usuario de manera síncrona con la directiva "ngModel" desde la plantilla de html. La otra variable "es_correo" guarda un estado booleano para determinar si el contenido de la variable "valor_correo" cumple con un patrón con el método "validar_correo()".

El método "validar_correo()" prueba la regla establecida en la línea 34 para corroborar si el contenido de la variable cumple con el patrón de "cadena@cadena.cadena", lo cuál es un patrón de cadena para un correo electrónico.

El método "recuperar_contraseña()" se llama cuando el usuario presiona el botón "Restablecer Contraseña". El método utiliza el *API* de *firebase* que es el método

"firebase.auth().sendPasswordResetEmail()" (línea 24) que proporciona *Firebae* que toma como argumento un correo para enviar una liga de recuperación de contraseña al usuario a su bandeja de correo.

```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-login-recuperar',
5    templateUrl: './login-recuperar.component.html',
6    styleUrls: ['./login-recuperar.component.css']
7  })
8
9  export class LoginRecuperarComponent implements OnInit {
10
11   valor_correo = null;
12   es_correo = false;
13
14   constructor( ) { }
15
16   ngOnInit() {
17   }
18
19   recuperar_contrasena() {
20     //envía correo de restablecimiento de contrasena
21     firebase.auth().sendPasswordResetEmail(this.valor_correo).then(function() {
22       console.log("correo enviado");
23     }, function(error) {
24       console.log("ocurrió un error");
25     });
26   }
27
28   validar_correo()
29   {
30     var re = /\S+@\S+\.\S+/;
31     this.es_correo = re.test(this.valor_correo);
32   }
33
34 }
```

Figura 3.36 Definición del Componente Login Recuperar

El correo de recuperación de contraseña tiene la apariencia como se muestra en la figura 3.37.

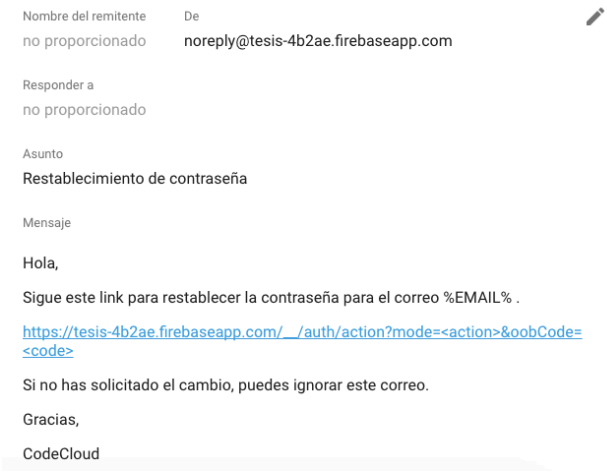

Figura 3.37 Contenido del Correo para Restablecer la Contraseña

El cuerpo del correo puede ser editado desde la consola de proyectos de *firebase*. Una vez que el usuario abra la liga, aparece un cuadro como se muestra en la figura 3.38.

El cuadro para restablecer la constraseña es un servicio que ofrece firebase de manera gratuita para poder darle la opción a los usuarios de volver a establecer su contraseña sin que el administrador pueda saber el contenido de la contraseña de los usuarios registrados.

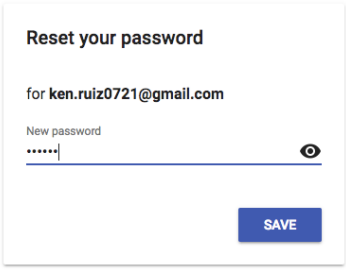

Figura 3.38 Cuadro para Restablecer Contraseña

3.3.3.2 Desarrollo Gráfico del Componente Login Recuperar

La pantalla de este componente tiene la apariencia que se muestra en la figura 3.39.

Esta pantalla contiene un formulario con un campo de texto, 3 botones y un *modal* (un botón está dentro del *modal*). El botón de "Iniciar sesión" tiene la directiva "routerLink" con valor "/login" para navegar hacia la pantalla de inicio de sesión.

Figura 3.39 Resultado del archivo "login-recuperar.component.html"

El campo de texto conecta el contenido introducido por el usuario con la variable "valor_correo" como se puede ver en la línea 53 (figura 3.40). Para comparar el contenido del campo de texto cada vez que el usuario inserta una cadena, se llama el método "validar_correo()" con la llamada "(keyup)" (llama el método cada vez que el usuario presiona una tecla) en la línea 53.

```
49    <div style="margin-bottom: 25px" class="input-group">
50        <span class="input-group-addon"><i class="glyphicon glyphicon-envelope"></i></span>
51        <input id="login-recuperar-correo" type="text" class="form-control"
52        name="username" value="" placeholder="correo"
53        (keyup)="validar_correo()" [(ngModel)]="valor_correo">
54    </div>
```

Figura 3.40 Definición del Campo de Texto en "login-registro.component.html"

En la línea 59 el botón de "Restablecer Contraseña" está desactivado hasta que el usuario introduzca una cadena con un patrón de correo (figura 3.41). En la línea 61 se define el *modal* a activar cuando se presiona el botón.

```
59    <button [disabled]="!es_correo"
60    type="button" class="btn btn-success" (click)="recuperar_contrasena() "
61    data-toggle="modal" data-target="#myModal">Restablecer Contraseña</button>
```

Figura 3.41 Definición del Botón "Restablecer Contraseña" en "login-recuperar.component.html"

El *modal* tiene la apariencia como se muestra en la figura 3.42.

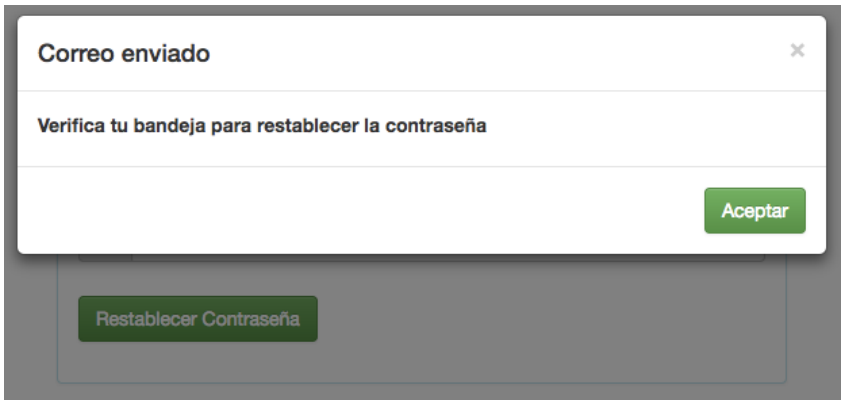

Figura 3.42 Apariencia del Modal de Confirmación en "login-registro.component.html"

El botón del *modal* se definió como se puede ver en la figura 3.43. Al presionar el botón, navega hacia la pantalla de inicio, además por el atributo "data-dismiss" con el valor de "modal" (línea 18) causa que se cierre el *modal*.

```
17    <div class="modal-footer">
18      <button type="button" class="btn btn-success" routerLink="/login" data-dismiss="modal">
19      Aceptar</button>
20    </div>
```

Figura 3.43 Definición del botón "Aceptar" del Modal en "login-registro.component.html"

3.3.4 Desarrollo de los Servicios del Módulo Login

Acontinuación se explica la definición de los 2 servicios pertenecientes al módulo de login. Estos 2 servicios tienen la funcionalidad de proteger la autenticación del usuario en conjunto.

El servicio "LoginService" se definió como se muestra en la figura 3.44. Los servicios en *Angular* se tienen que declarar como un objeto inyectable (línea 2 y 4), lo cuál significa que cada instancia de este servicio a través de toda la aplicación hará uso del mismo y único

servicio, así asegurando una proporción de datos consistentes para todos los clientes que consuman el servicio.

En este servicio se vigila el estado de autenticación del usuario desde su creación (línea 10 a la línea 19) ya que se susscribe la autenticación del servicio de *Firebase* (línea 12). En el momento que cambie el estado de autenticación guarda un valor booleando en la variable "usuario_autenticado". El método "esAutenticado()" proporciona el valor de la variable "usuario_autenticado" para comunicar al exterior del servicio el estado de autenticación del usuario. Este método es consumido por el servicio "ProtegerLoginService" para determinar si el usuario es capaz de navegar hacia el contenido dinámico de la aplicación.

```
1   import { Injectable }    from '@angular/core';
2   import { AngularFire } from 'angularfire2';
3
4   @Injectable()
5
6   export class LoginService {
7
8       private usuario_autenticado = false;
9
10      constructor(private af:AngularFire) {
11          //monitorea el cambio de estado
12          this.af.auth.subscribe(auth => {
13              if(auth) {
14                  this.usuario_autenticado = true;
15              } else {
16                  this.usuario_autenticado = false;
17              }
18          });
19      }
20
21      esAutenticado() {
22         return this.usuario_autenticado;
23      }
24
25  }
```

Figura 3.44 Definición del Servicio "LoginService"

El servicio proteger login se definió como se muestra en la figura 3.45. Una importación a destacar es el módulo "CanActivate" que se implementó para la declaración de este servicio (línea 2 y 6). Este módulo contiene el método "canActivate" que es utilizado en la definición del routing como se había mencionado. El método tiene que regresar un valor booleano, es por eso que consume el método "esAutenticado()" del servicio "LoginService" que retorna el estado de autenticación de forma booleana (línea 13).

```
1   import { Injectable } from '@angular/core';
2   import { CanActivate } from '@angular/router';
3   import { LoginService } from './login.service';
4
5   @Injectable()
6   export class ProtegerLoginService implements CanActivate {
7
8     constructor(
9       private servicio: LoginService
10      ) { }
11
12    canActivate() {
13      return this.servicio.esAutenticado();
14    }
15
16  }
```

Figura 3.45 Definición del Servicio "ProtegerLoginService"

Con esto se concluye el desarrollo de los 3 componentes y 2 servicios que conforman el módulo de login.

3.4 Desarrollo del Módulo de Consola de Proytectos

Al igual que el módulo de login, este módulo se conforma de 3 componentes. El componente consola de proyectos que permite al usuario crear, visualizar y escoger proyectos. El componente panel de proyecto que es una representación gráfica de cada proyecto para que el usuario sea capaz de interactuar con el proyecto (descargar, compartir, eliminar y abrir), este componente se encuentra situado dentro del componente consola de proyectos. Y finalmente el componente integrante que representa a los colaboradores agregados por parte del usaurio, este componente se sitúa dentro del componente panel proyecto. Como se muestra en la figura 3.46, los componentes panel proyecto e integrante son llamados por sus correspondientes componentes padres.

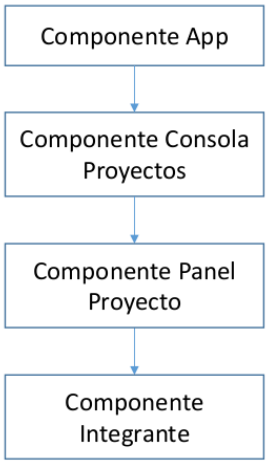

Figura 3.46 Relación de los Componentes del Módulo Consola de Proyectos

La interfaz gráfica de este módulo lo conforman los 3 componentes mencionados, se acomodan como se muestra en la figura 3.47 para formar la interfaz gráfica.

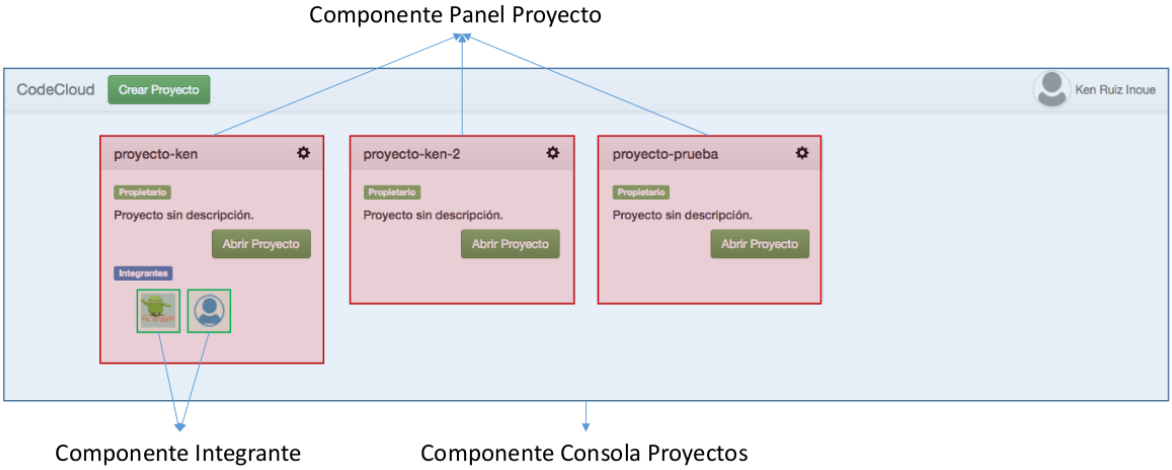

Figura 3.47 Componentes Acomodados del Módulo de Consola de Proyectos

Todo lo rodeado por el marco azul es el resultado del componente consola proyectos. Lo rodeado por el marco rojo es la instancia de cada proyecto del usuario en el componente panel proyecto. Y lo verde es la instancia de cada integrante (sin incluir al propietario) en el componente integrante.

Cada componente correspondiente al módulo de consola de proyectos se nombró como se puede ver en la tabla.

Tabla 3.4 Componentes del Módulo de Consola de Proyectos

Función	Nombre del Archivo
Interfaz gráfica de visualización y administración de los proyectos	consola-proyectos.component.ts
Interfaz gráfica de interacción con los proyecotos	panel-proyecto.component.ts
Interfaz gráfica de visualización de los integrantes del proyecto	integrante.component.ts

3.4.1 Desarrollo del Componente Consola Proyectos

El primer componente que se desarrolló fue el componente consola proyectos. Este componente es el más importante del módulo ya que recibe toda la información necesaria de la base de datos para representarla de manera gráfica. En este componente el usuario puede interacturar con 3 tipos de objetos, modal para crear proyectos nuevos, menú del usuario y proyectos propietarios del usuario instanciados en el componente panel proyecto.

3.4.1.1 Desarrollo Lógico del Componente Consola Proyectos

En la figura 3.48 se pueden ver las importaciones definidas para el componente consola proyectos.

Se importaron las clases de “Component”, “OnInit” y “AfterViewInit” de la librería de *angular*. La clase “AngularFire”, “FirebaseListObservable” y “FirebaseObjectObservable” de la librería *angularfire 2*. También se importaron 3 clases personalizadas “Proyecto”, “NodoArbol” y “Usuario”. Como se puede observar en la línea 10, se declaró la variable “$” para habilitar el uso de *jQuery*. Se seleccionó como selector “app-consola-proyectos” (línea 13).

“AfterViewInit” (línea 1) es una clase que permite utilizar el ciclo de vida del componente de *Angular* para ejecutar líneas de código después que los componentes visuales hayan inicializado. Esta clase se utilizó para disparar funciones de *bootstrap* que emplean *jQuery*.

La clase “FirebaseListObservable” (línea 4) también guarda la consulta de base de datos, pero a diferencia de “FirebaseObjectObservable”, guarda una lista de objetos.

```
1  import { Component, OnInit, AfterViewInit } from '@angular/core';
2  import { Router } from '@angular/router';
3
4  import { AngularFire, FirebaseListObservable, FirebaseObjectObservable } from 'angularfire2';
5
6  import { Proyecto } from '../modelos/proyecto';
7  import { NodoArbol } from '../modelos/nodoArbol';
8  import { Usuario } from '../modelos/usuario';
9
10 declare var $: any;
11
12 @Component({
13   selector: 'app-consola-proyectos',
14   templateUrl: './consola-proyectos.component.html',
15   styleUrls: ['./consola-proyectos.component.css']
16 })
```

Figura 3.48 Importaciones en "consola-proyectos.component.ts"

La clase "Proyecto" es una clase que guarda propiedades de un proyecto. Se definió como se puede observar en la figura 3.49.

Esta clase tiene 5 atributos y 1 constructor. El atributo "descripcion" es opcional debido a que el usuario puede escoger no darle una. El atributo integrantes guarda como mínimo al propietario y los posibles usuarios que se lleguen a integrar al proyecto. Al momento de instanciar un objeto de esta clase, es necesario proporcionar 3 entradas, nombre del proyecto, descripción del proyecto y la llave primaria del usuario. El id de esta clase se genera a través del id del usuario y nombre del proyecto (línea 10), lo cuál significa que el usuario no puede tener 2 proyectos con el mismo nombre en su pertenencia.

```
1  export class Proyecto {
2
3      public id: string;
4      public nombre: string;
5      public descripcion?: string;
6      public propietario: string;
7      public integrantes: Array<any>;
8
9      constructor(nombre: string, descripcion: string, id_usuario: string) {
10         this.id = id_usuario + "_" + nombre;
11         this.nombre = nombre;
12         this.descripcion = descripcion;
13         this.propietario = id_usuario;
14     }
15 }
```

Figura 3.49 Definición de la Clase "Proyecto" en "proyecto.ts"

Otra clase personalizada es la clase "NodoArbol" (figura 3.50), esta clase es fundamental para la construcción del proyecto seleccionado dentro de la aplicación. Al instanciar un objeto de esta clase, el usuario es capaz de interactuar con el proyecto para modificar y crear carpetas o archivos dentro del proyecto en el área de trabajo (editor de texto).

La clase tiene en total 13 atributos para representar el estado de un proyecto. Cada atributo o grupo de atributos tiene su explicación (comentario) en la figura 3.50 de manera sencilla. Pero cabe resaltar que el atributo "id" (línea 4) es un atributo opcional debido a que se tiene

que crear el registro dentro de la base de datos de *firebase* y posteriormente se obtiene la llave del registro para actualizarlo con el identificador correspondiente dentro del atributo "id".

Otro atributo importante es el de "hijos", este atributo permite guardar arreglos de su misma clase "NodoArbol", así permitiendo la generación de una estructura de árbol para representar la carpeta de proyecto del sistema.

```
1   export class NodoArbol {
2
3       //despues de realizar el push se actualiza el id con el push-id generado
4       public id?: string;
5       public nombre: string;
6       //todos los nodos deben de tener este atributo
7       public proyecto_perteneciente: string;
8
9       //lo tienen todos los archivos y carpetas que no sean el nodo inicial
10      public nodo_padre?: string;
11
12      //después de realizar el push se actualiza la referencia_padre e hijo con el push-id generado removiendo el -
13      public referencia_padre?: string;
14      public referencia_hijo?: string;
15
16      //sirve para guardar el path del archivo en el almacen de firebase
17      public url?: string;
18
19      //atributos para manejar estados
20      public es_archivo: boolean;
21      public es_carpeta: boolean;
22      public carpeta_abierta?: boolean;
23      public archivo_abierto?: boolean;
24      public es_seleccionado: boolean;
25
26      //solamente si es carpeta puede tener hijos
27      public hijos?: Array<NodoArbol>;
28
29      constructor(nombre: string, proyecto_perteneciente: string) {
30          this.carpeta_abierta = false;
31          this.es_archivo = false;
32          this.es_carpeta = true;
33          this.nombre = nombre;
34          this.proyecto_perteneciente = proyecto_perteneciente;
35          this.es_seleccionado = false;
36      }
37
38  }
```

Figura 3.50 Definición de la Clase "NodoArbol" en "nodoArbol.ts"

En la figura 3.51 se pueden ver los pasos que se siguen en cada sistema (aplicación y *firebase*) para la creación de un nuevo registro. Se optó utilizar el identificador que genera *firebase* de manera automática al crear un nuevo registro, así para evitar crear un algoritmo que genere un identificador único. En el paso 4 se genera el identificiador y se le retorna a la aplicación para que así se pueda actualizar el nodo con el identificador creado.

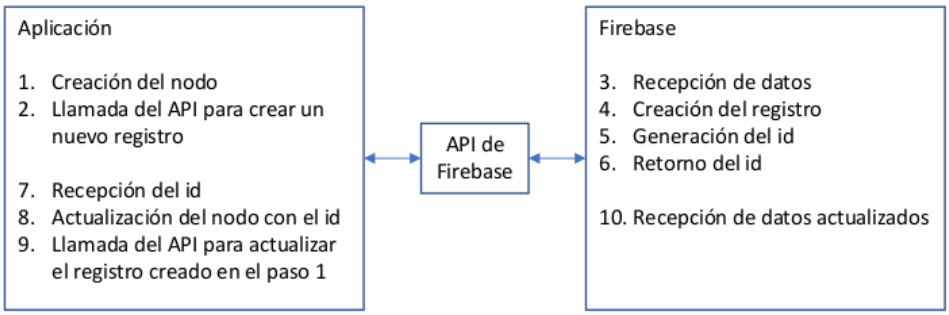

Figura 3.51 Comunicación de la Aplicación con Firebase en la Creación de un Reigstro

En la siguiente figura 3.52 se pueden observar las variables que se utilizan dentro del componente consola proyectos y su constructor.

En total se declararon 14 variables golabales dentro del componente. En el constructor se declararon 2 variables que hacen el uso de la clase "Router" y la librería "AngularFire" al igual que otros compoentes generados anteriormente. Además, implementa la clase "OnInit" y "AfterViewInit" (línea 18).

```
18  export class ConsolaProyectosComponent implements OnInit, AfterViewInit {
19
20    referencia_almacen = firebase.storage().ref();
21    usuario: Usuario;
22    proyectos_descargados: Array<Proyecto>;
23    proyectos_descargados_ajenos: Array<Proyecto>;
24    consulta_usuario: FirebaseObjectObservable<any>;
25    consulta_proyectos: FirebaseListObservable<any>;
26    consulta_proyecto_crear: FirebaseObjectObservable<any>;
27
28    nombre_usuario;
29    url_imagen;
30    valor_nombre;
31    valor_descripcion = null;
32
33    editando_nuevo_nombre = false;
34    formulario_aprobado = false;
35    nombre_unico = true;
36
37    constructor(
38      private router: Router,
39      private af: AngularFire
40    ) { }
```

Figura 3.52 Variables y Constructor en "consola-proyectos.component.ts"

En la tabla 3.5 se describe la funcionalidad y objetivo de cada variable perteneciente al componente consola proyectos.

Tabla 3.5 Variables Globales del Componente Consola Proyectos

Variable	Función
referencia_almacen	Guarda la referencia del almacenamiento de firebase
usuario	Guarda la información del usuario autenticado
proyectos_descargados	Guerda todos los proyectos del usuario

proyectos_descargados_ajenos	Guarda todos los proyectos en el que participa el usuario
consulta_usuario	Apunta la consulta de la base de datos de firebase del usuario autenticado
consulta_proyectos	Apunta la consulta de la base de datos de firebase de los proyectos del usuario
consulta_proyecto_crear	Apunta la consulta de la base de datos de firebase a la que se desea crear
nombre_usuario	Guarda el nombre del usuario para mostrarlo en la interfaz gráfica
url_imagen	Guarda la url de la foto de perfil del usuario para mostrarla en la interfaz gráfica
valor_nombre	Guarda el valor del nombre del proyecto introducido en el formulario para la creación de un nuevo proyecto
valor_descripcion	Guarda el valor de la descripción del proyecto introducido en el formulario para la creación de un nuevo proyecto
editando_nuevo_nombre	Variable boolena para habilitar el botón "Guardar" que se encuentra en el formulario para crear un nuevo proyecto
formulario_aprobado	Variable booleana que aprueba la validez del formulario
nombre_unico	Variable booleana que aprueba si el nombre introducido no está en uso

A continuación se explican los 11 métodos que contiene el componente consola proyectos.

El primer método a llamarse es "ngOnInit()". Este método se declaró debido a que el componente implementa la clase "OnInit", método que llama la aplicación después de haber vinculado las variables con la plantilla de html. En la figura 3.53 se puede mostrar una sencilla abreviación del método. En el nivel más alto del método suscribe la autenticación del usuario, lo cuál significa que cada vez que se perciba un cambio en el estado de autenticación del usuario, se ejecuta un estado if else. Si el usuario sigue existiendo (usuario autenticado), se ejecuta el código de la línea 46 a la 85, de manera contraria (usuario no autenticado), se ejecuta la línea 89 que nevega hacia la pantalla de inicio de sesión.

```
42    ngOnInit() {
43
44      this.af.auth.subscribe(usuario => {
45        if(usuario) {…
86        } else {
87          this.router.navigateByUrl('/login');
88        }
89      });
90    }
```

Figura 3.53 Contenido Breve del Método "ngOnInit()"

En la siguiente figura 3.54 se muestra el contenido del código en el caso de que el usuario se encuentre autenticado.

Este método realiza en total 3 consultas diferentes para mantener actualizados los datos remotos de los usuarios y proyectos. De esta manera la aplicación mantiene la interfaz gráfica actualizada en tiempo real.

La primera consulta que se realiza en el código es la del usuario, para eso se guarda la referencia de la consulta en la variable "consulta_usuario" (línea 48). Se suscribe la consulta para vigilar de manera continua el cambio de alguna de las propiedades del usuario. En el caso de que se llegue a percibir un cambio, se vuelven a asignar los valores actualizados en las variables "usuario", "nombre_usuario" y "url_imagen".

Después sigue la consulta de los proyectos que le pertenecen al usuario. Se guarda la consulta en la variable "consulta_proyectos" (línea 58 a la 64). Para esta consulta, se utilizó un *query* que retorna una lista de objetos con el campo "propietario" que coincidan con el valor del identificador del usuario autenticado (línea 62) dentro de la base de datos de *firebase*. Se suscribe la consulta para cada vez que se perciba un cambion (eliminar, crear o actualizar) en la ubicación de la consulta, se vuelvan a descargar los proyectos para asignarlos en la variable "proyectos_descargados".

La última consulta es la de los proyectos ajenos, es decir, proyectos que no son pertenencia del usuario pero si como integrante (línea 73 a la 84). La consulta se guarda en la variable "consulta_proyectos_ajenos" y también se suscribe. Dentro de la suscripción, se renicializa el arreglo "proyectos_descargados_ajenos" ya que en este código no se asigna directamente en el arreglo la colección que retorna la base de datos de *firebase*, sino se realiza un filtro para agregar el elemento válido dentro del arreglo con el método push(). Se recorren todos los proyectos con el estado "for" (línea 76) y además se recorren todos los integrantes en cada proyecto de manera anidada (línea 77), en el segundo recorrido es donde se aplica el filtrado. El filtrado consiste en 2 condiciones; la primera condición es que el id del usuario debe de encontrarse en la propiedad de "integrantes[index].id" del proyecto (línea 79) y la segunda condición consiste en que la propiedad "integrantes[index].propietarioProyecto" no contenga un valor booleano verdadero (línea 80).

```
47        //obtener datos del usuario almacenados en la base de datos de firebase
48        this.consulta_usuario = this.af.database.object('/users/' + usuario.uid);
49        this.consulta_usuario.subscribe(usuario_retornado => {
50
51            this.usuario = usuario_retornado;//guarda el objeto retornado en la variable usuario
52            this.nombre_usuario = this.usuario.nombre;
53            this.url_imagen = this.usuario.imgurl;
54
55        });
56
57        //obtener datos de proyectos almacenados en la base de datos de firebase
58        this.consulta_proyectos = this.af.database.list('/proyectos',
59          {
60            query: {
61              orderByChild: 'propietario',
62              equalTo: usuario.uid
63            }
64          });
65
66        this.consulta_proyectos.subscribe(proyectos => {
67          this.proyectos_descargados = proyectos;
68        });
69
70        //obtener datos de proyectos almacenados en la base de datos de firebase
71        let consulta_proyectos_ajenos = this.af.database.list('/proyectos');
72
73        consulta_proyectos_ajenos.subscribe(proyectos => {
74          this.proyectos_descargados_ajenos = [];//inicializar variable para evitar duplicidad
75
76          for(let proyecto of proyectos) {
77            for(let index in proyecto.integrantes) {
78              if((proyecto.integrantes[index].id == usuario.uid) &&
79              (proyecto.integrantes[index].propietarioProyecto != true)) {
80                this.proyectos_descargados_ajenos.push(proyecto);
81              }
82            }
83          }
84        });
```

Figura 3.54 Contenido del Código del Método "ngOnInit()" Detallado

Una vez que se haya llamado el método "ngOnInit()", la aplicación llama el método "ngAfterViewInit()", en la figura 3.55 se muestra el contenido del método.

Se declaró una variable llamada "ref_this" (línea 93) para guardar la referencia del "this", que contiene todas las referencias de las variables y métodos definidos de manera global en el componente. En el lenguaje de *Typescript* la referencia "this" pierde el alcance dentro de un método interno. El método "ngAfterViewInit" define 2 métodos internos, es por eso que se declaró esta referencia para no perder de alcance el "this" dentro de los métodos.

Los métodos declarados son funcionalidades de *jQuery* que permiten detectar eventos de los modals. En este caso, el evento que se tomó en cuenta es el de esconder (línea 95 y 98). Cuando el usuario hace click afuera del cuadro del modal, se esconde y reinicia las variables que contiene la información del formulario, para así evitar que la introducción del usuario permanezca en el formulario.

```
92      ngAfterViewInit() {
93          var ref_this = this;
94          //reacciona ante los evento hidden
95          $('#modalPerfil').on('hidden.bs.modal', function (e) {
96            ref_this.terminar_cambio_nombre();
97          });
98          $('#modalCrearProyecto').on('hidden.bs.modal', function (e) {
99             ref_this.terminar_crear_proyecto();
100         });
101     }
```

Figura 3.55 Contenido del Método "ngAfterViewInit()"

El método "crear_proyecto()" se encarga de construir proyectos nuevos a través de la información que introdujo el usuario en el formulario para crear proyectos. En la figura 3.56 se puede observar el contenido del método "crear_proyecto()".

Al entrar al método, lo primero que se realiza es la construcción de un nuevo proyecto haciendo uso de la clase "Proyecto" para asignarlo en la variable "proyecto". Utiliza el constructor propio recibiendo como entrada la variable "valor_nombre", "valor_descripcion" y "usuario.id". Después se consulta la ubicación en la que se desea guardar el nuevo proyecto creado (línea 107) y se genera el registro en la base de datos (línea 108).

Una vez creado el proyecto, se utiliza la clase "NodoArbol" para crear el nodo padre del proyecto (línea 111). Se inserta el registro en la base de datos con el método "push()" y llama el método "then()" para poder acceder al indentificador que generó *firebase* al insertar el nuevo registro (línea 114).

Se genera una consulta con el identificador retornado (línea 117 y 119). Con el identificador único, se actualiza la propiedad "id", "referencia_padre" y "referencia_hijo". El identificador que genera firebase tiene al inicio de la cadena un '-', por eso se utiliza el método "substring(1)" para eliminar ese carácter de la cadena (línea 120 y 121). La propiedad "referencia_padre" requiere tener un '#' al inicio, por lo cual se concatena con el identificador proporcionado por *firebase* (línea 120). La propiedad "referencia_hijo" únicamente guarda el identificador sin el carácter '-'. Finalmente se llama la petición de actualización del registro (línea 122) y se le asigna el valor "undefined" en la variable "valor_nombre" y el valor nulo en la variable "valor_descripcion" para limpiar el contenido del formulario (línea 124 y 127). En el módulo de área de trabajo se explicará con más detalle cómo se utilizan las propiedades "referencia_padre" y "referencia_hijo".

Al final del método, se genera una consulta para crear el registro del integrante propietario (línea 129 y 130). En el registro del integrante tiene 4 propiedades, "id", "nombre", "imgurl" y "propietarioProyecto" que guarda un booleano para representar si es el propietario o no del proyecto (línea 131 a la 136).

```
crear_proyecto(){
  //generacion del proyecto
  var proyecto: Proyecto = new Proyecto(this.valor_nombre, this.valor_descripcion, this.usuario.id);

  this.consulta_proyecto_crear = this.af.database.object('/proyectos/' + this.usuario.id + "_" + this.valor_nombre);
  this.consulta_proyecto_crear.set(proyecto);

  //generacion del nodo_padre
  var estructura: NodoArbol = new NodoArbol(this.valor_nombre, this.usuario.id + "_" + this.valor_nombre);

  const ref_estructura = this.af.database.list('/nodos_proyecto_' + this.usuario.id + "_" + this.valor_nombre);
  ref_estructura.push(estructura).then((objeto) => {

     //actualizar el proyecto para que tenga el id y referencia_padre que se genera a través del push id de firebase
     const ref_objeto = this.af.database.object(
       '/nodos_proyecto_'  + this.usuario.id + "_" + this.valor_nombre + '/' + objeto.key);
     estructura.id = objeto.key;
     estructura.referencia_padre = "#" + objeto.key.substring(1);
     estructura.referencia_hijo = objeto.key.substring(1);
     ref_objeto.update(estructura);

     this.valor_nombre= undefined;
    });

  this.valor_descripcion = null;

  let consulta_proyecto = this.af.database.object(
    'proyectos/' + this.usuario.id + "_" + this.valor_nombre + '/integrantes/' + this.usuario.id);
        consulta_proyecto.set({
          nombre: this.usuario.nombre,
          imgurl: this.usuario.imgurl,
          id: this.usuario.id,
          propietarioProyecto: true
        });

}
```

Figura 3.56 Contenido del Método "crear_proyecto()"

Como se puede observar en la figura 3.57, cada proyecte tiene como mínimo un integrante, el propietario del proyecto. Un proyecto puede tener n integrantes, pero únicamente un propietario (en el caso de la figura el integrante 2 es el propietario).

Esta relación entre el proyecto y los integrantes, permite dar a conocer a la aplicación los integrantes a mostrar, y saber cuales proyectos le pertenecen al usuario. Cuando se explique acerca del componente panel proyecto, se explicará con más detalle la importancia de la relación.

Si el usuario desea crear un nuevo proyecto necesita introducir un nombre para el proyecto en el formulario como se había mencionado. Para validar ese campo obligatorio se creó el método "validar_nombre()".

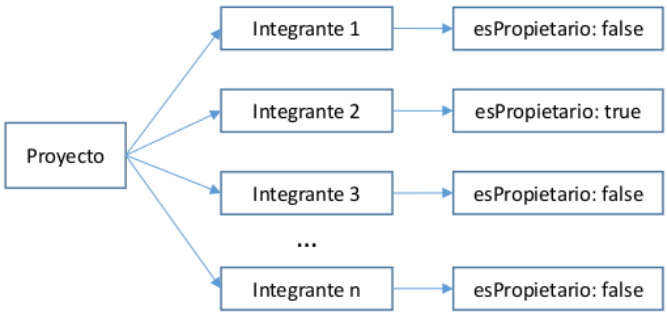

Figura 3.57 Relación Proyecto Integrante

Como se puede observar en la figura 3.58, el método busca que la variable "valor_nombre" tenga una longitud de cadena mayor a 5 para establecer un valor booleano verdadero en la variable "formulario_aprobado" (línea 142 a la 146) y así aprobar el formulario. También compara el nombre de todos los proyectos descargados del usuario para notificar a la aplicación si es único el nombre que el usuario introdujo con la variable booleana "nombre_unico" (línea 149 a la 154).

```
140     validar_nombre(){
141       this.nombre_unico = true;
142       if(this.valor_nombre.length > 5) {
143         this.formulario_aprobado = true;
144       } else {
145         this.formulario_aprobado = false;
146       }
147
148       //no permite la introducción de un nombre existente
149       for (let proyecto of this.proyectos_descargados) {
150         if(proyecto.nombre == this.valor_nombre){
151           this.nombre_unico = false;
152           return
153         }
154       }
155
156     }
```

Figura 3.58 Contenido del Método "validar_nombre()"

Para comunicar a la aplicación que el usuario ha abandonado o terminado la creación de un nuevo proyecto, se emplea el método "terminar_crear_proyecto()". En la figura 3.59 se puede observar el contenido del método.

Lo único que hace este método es reiniciar los valores de las variables vinculadas al formulario (línea 141 y 142). Cada variable se reinicia de manera diferente ya que el valor "undefined" es fácil de comparar en html, pero no para la *API* de *firebase*. El contenido de la variable "valor_descripcion" se envía con el *API* de *firebase*, la cuál no sabe interpretar el valor "undefined".

```
140     terminar_crear_proyecto() {
141       this.valor_nombre = undefined;
142       this.valor_descripcion = null;
143     }
```

Figura 3.59 Contenido del Método "terminar_crear_proyecto()"

Además del modal para crear proyectos, el usuario puede interacturar con el menú de usuario. El menú contiene 2 opciones, “Edición de Perfil” y “Cerrar Sesión”. La primera opción abre un modal para cambiar nombre, foto de perfil y contraseña del usuario. La segunda opción cierra la sesión del usuario.

Al presionar la opción “Edición de Perfil” el usuario se encuentra con un modal con 3 opciones. La primera opción utiliza los métodos “iniciar_cambio_nombre()”, “actualizar_nombre_usuario()” y “terminar_cambio_nombre()” para permitir al usuario cambiar su nombre (figura 3.60).

Estos métodos cambian el estado de la variable booleana “editando_nuevo_nombre”(línea 164 y 174) y hace una petición de actualización en el registro del usuario solicitado (línea 168 y 169).

```
163     iniciar_cambio_nombre() {
164       this.editando_nuevo_nombre = true;
165     }
166
167     actualizar_nombre_usuario(nuevo_nombre) {
168       let consulta_usuario = this.af.database.object('/users/' + this.usuario.id);//obtener referencia del usuario
169       consulta_usuario.update({nombre: nuevo_nombre});//actualizar la propiedad "nombre" del usuario en la BD
170       this.terminar_cambio_nombre();
171     }
172
173     terminar_cambio_nombre() {
174       this.editando_nuevo_nombre = false;
175     }
```

Figura 3.60 Métodos para Cambiar Nombre del Usuario

La segunda opción del modal para editar el perfil, permite cargar al usuario una nueva foto para actualizar la foto de perfil. Para llevar a cabo esta acción, se llama el método “cargar_imagen()” (figura 3.61).

Para realizar la carga de imagen, se colocó una etiqueta “<input/>” de tipo “file”. Cuando el usuario escoge la imagen, se manda toda la información del evento como entrada de la función (línea 177). Se extraen los archivos y se asigan a la variable “files” (línea 179). Corrobora que el tipo de la imagen sea “jpeg” o “png” para seguir con el algoritmo (línea 182), de manera contraria manda una alerta (línea 220). Se crea una referencia de carga en el almacen de *firebase* (línea 184 y 185) y se inicia la carga con el método “on()” (línea 187). Una vez terminada la carga, se crea una referencia del usuario para actualizar la propiedad de “imgurl” con la liga de la nueva imagen (línea 215 y 216).

```
177     cargar_imagen(event) {
178       var ref_this = this;
179       var files = event.srcElement.files;
180
181       //solamente permite realizar cargas de imágenes jpeg o png
182       if(files[0].type == "image/jpeg" || files[0].type == "image/png") {
183         //construcción del nombre de archivo
184         var uploadTask = this.referencia_almacen.child('fotos/' + this.usuario.id + '.' +
185         files[0].type.substring(6)).put(files[0]);
186
187         uploadTask.on(firebase.storage.TaskEvent.STATE_CHANGED,
188           function(snapshot) {…
197           },
198           function(error){…
212           },
213           function() {
214             //console.log("carga completa");
215             let consulta_usuario = ref_this.af.database.object('/users/' + ref_this.usuario.id);
216             consulta_usuario.update({imgurl: uploadTask.snapshot.downloadURL});
217           }
218         );
219       } else {
220         alert("No es una imagen");
221       }
222     }
```

Figura 3.61 Contenido del Método "cargar_imagen()"

La tercera y última opción del modal para editar el perfil, le da la opción al usuario de modificar la contraseña enviándole una liga a su correo con el método "enviar_link()" (figura 3.62) a través de la *API* de *firebase* (al igual que le hace el componente login recuperar).

```
227     enviar_link() {
228       firebase.auth().sendPasswordResetEmail(this.usuario.correo).then(function() {
229         alert("Se envió un link para establecer una nueva contraseña");
230       }, function(error) {
231         alert("Ocurrió un error, intente de nuevo");
232       });
233     }
```

Figura 3.62 Contenido del Método "enviar_link()"

La segunda opción del menú de usuario, permite al usuario cerrar la sesión con el método "cerrar_sesion()" (figura 3.63).

```
235     cerrar_sesion(){
236       this.af.auth.logout();
237     }
```

Figura 3.63 Contenido del Método "cerrar_sesion()"

3.4.1.2 Desarrollo Gráfico del Componente Consola Proyectos

De los 11 métodos que se acaban de explicar, algunos son llamados directamente de la plantilla de html correspondiente de este componente. A continuación se explica el desarrollo de la interfaz gráfica.

Lo primero que se encuentra definido en el archivo "consola-proyectos.component.html" son 2 modals. El modal para crear proyectos (figura 3.64) y el modal para editar el perfil (figura 3.65).

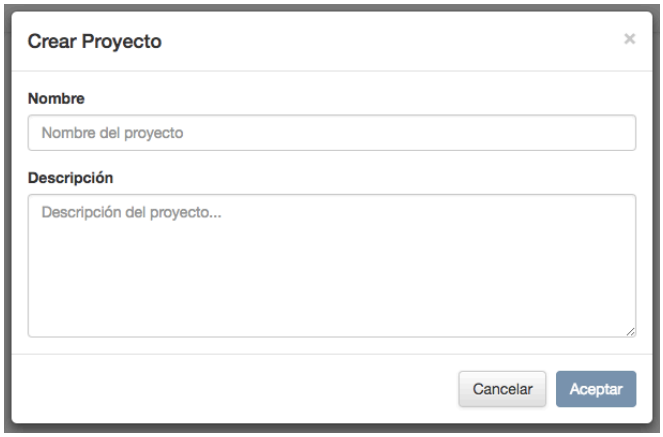

Figura 3.64 Apareicnia Modal para Crear Proyectos

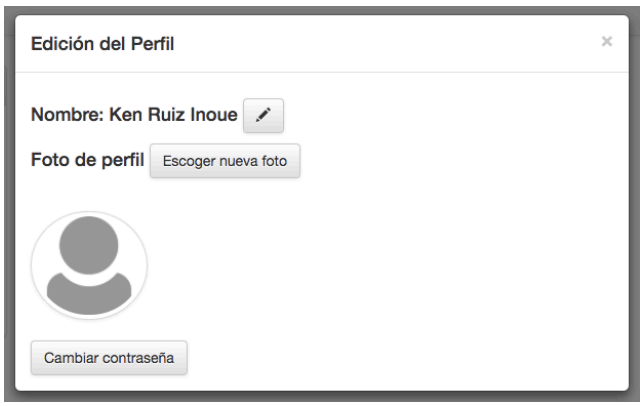

Figura 3.65 Apareicnia Modal para Editar Perfil

Los modals tienen que declararse fuera del cuerpo para evitar que se pierda al llamarlo. En la figura 3.66 se puede ver la definición del modal para crear proyectos.

Tiene como identificador “modalCrearProyecto” (línea 2), este identificador permite que otro elemento pueda activar el modal.

El modal tiene un formulario con 2 campos de introducción como se había mencionado. Cada campo tiene una variable vinculada a través de la directiva “ngModel”. Las variables vinculadas son “valor_nombre” (línea 12) y “valor_descripcion” (línea 24).

Este modal utiliza 3 métodos definidos en “conosola-proyectos.component.ts”. El método “terminar_crear_proyecto()”, “validar_nombre()” y “crear_proyecto()”.

El método “terminar_crear_proyecto()” se llama de 3 maneras diferentes. Cuando el usuario desea cerrar el modal con el botón “x” (línea 7), cuando desea cancelar el modal con el botón “Canelar” (línea 30) y cuando el usuario hace click fuera del modal (explicado en la figura 3.55).

Cada vez que el usuario introduce algún carácter en el campo de introducción “Nombre”, se manda a llamar el método “validar_nombre()” para cambiar el estado de las variables “formulario_aprobado” y “nombre_unico” (línea 12).

El método “crear_proyecto()” se llama cuando el usuario llena el formulario del modal para generar un nuevo proyecto y presiona el botón “Aceptar”. El botón “Aceptar” se deshabilita si alguna de las variables “formulario_aprobado” o “nombre_unico” contiene un valor boolenado falso (línea 32).

```
1  <!-- modal crear proyecto -->
2  <div class="modal fade" id="modalCrearProyecto" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">
3    <div class="modal-dialog" role="document">
4      <div class="modal-content">
5        <div class="modal-header">
6          <button type="button" class="close" data-dismiss="modal" aria-label="Close" (
7              click)="terminar_crear_proyecto()"><span aria-hidden="true">&times;</span></button>
8          <h4 class="modal-title" id="myModalLabel">Crear Proyecto</h4>
9        </div>
10       <div class="modal-body">
11         <label for="entradaNombreProyecto">Nombre</label>
12         <input type="text" class="form-control" placeholder="Nombre del proyecto"  [(ngModel)]="valor_nombre" (keyup)="validar_nombre()">
13
14         <div style="margin-bottom:15px" *ngIf="!nombre_unico" class="alert alert-danger">
15             <span class="glyphicon glyphicon-remove-circle" aria-hidden="true"></span>
16             Ya existe el proyecto: {{valor_nombre}}
17         </div>
18
19         <div style="padding-top:15px">
20             <label for="entradaNombreProyecto">Descripción</label>
21             <div class="row">
22                 <div class="col-lg-12 col-md-12 col-sm-12">
23                     <textarea style="resize:vertical;" class="form-control" placeholder="Descripción del proyecto..."
24                     rows="6" name="comment" [(ngModel)]="valor_descripcion" ></textarea>
25                 </div>
26             </div>
27         </div>
28       </div>
29       <div class="modal-footer">
30         <button type="button" class="btn btn-default" data-dismiss="modal" (click)="terminar_crear_proyecto()">Cancelar</button>
31         <button type="button" class="btn btn-primary" (click)="crear_proyecto()"
32         [disabled]="!formulario_aprobado || !nombre_unico" data-dismiss="modal">Aceptar</button>
33       </div>
34     </div>
35   </div>
36 </div>
```

Figura 3.66 Definición del Modal para Crear Proyectos en “consola-proyectos.component.html”

Si la plantilla encuentra un valor booleano falso en la variable “nombre_unico”, muestra un mensaje de alerta para dar a entender que ya existe un proyecto con el mismo nombre (línea 14 a la 17, figura 3.67).

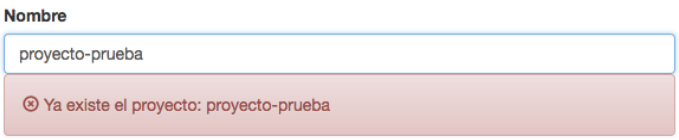

Figura 3.67 Mensaje de Alerta en el Modal para Crear Proyectos

A continuación, se muestra la primeta parte de la definición del modal para editar perfil en la figura 3.68. Tiene como identificador “modalPerfil” (línea 40).

Al igual que el modal para crear proyectos, se puede observar que se utilizó la misma lógica para mandar a llamar el método “terminar_cambio_nombre()” para comunicar a la apliación la conclusión de la edición del nombre (línea 45).

Obtiene el nombre de la variable "nombre_usuario" (línea 51) para que el usuario pueda observar su propio nombre. Al presionar el botón con el ícono de edición (figura 3.64), se llama el método "iniciar_cambio_nombre()" (línea 53).

```
39  <!-- modal edición de perfil -->
40  <div class="modal fade" id="modalPerfil" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">
41    <div class="modal-dialog" role="document">
42      <div class="modal-content">
43        <div class="modal-header">
44          <button type="button" class="close" data-dismiss="modal"
45          aria-label="Close" (click)="terminar_cambio_nombre()"><span aria-hidden="true">&times;</span></button>
46          <h4 class="modal-title" id="myModalLabel">Edición del Perfil</h4>
47        </div>
48        <div class="modal-body">
49
50
51          <h4>Nombre: {{nombre_usuario}}
52              <div class="btn-group">
53                  <button class="btn btn-default" (click)="iniciar_cambio_nombre()">
54                      <span class="glyphicon glyphicon-pencil"></span>
55                  </button>
56              </div>
57          </h4>
```

Figura 3.68 Definición del Modal para Editar Perfil en "consola-proyectos.component.html" Parte 1

Si la variable "editando_nuevo_nombre" (figura 3.69, línea 59) tiene un valor booleano verdadero, entonces muestra un campo para introducir el nuevo nombre que desee el usuario.

```
59          <div *ngIf="editando_nuevo_nombre">
60              <input type="text" class="form-control" placeholder="Nuevo nombre de usuario" #nuevoNombre>
61              <button style="margin-top: 10px;" class="btn btn-success"
62              (click)="actualizar_nombre_usuario(nuevoNombre.value)">Guardar cambio</button>
63          </div>
```

Figura 3.69 Definición del Modal para Editar Perfil en "consola-proyectos.component.html" Parte 2

El campo de texto tiene la apariencia como se puede ver en la figura 3.70, el botón de "Guardar cambio" llama el método "actualizar_nombre_usuario()" (línea 62).

Figura 3.70 Apariencia Campo de Texto y Botón "Guardar cambio" en el Modal para Editar Perfil

La última parte de la definición del modal para editar perfil tiene el contenido como se puede observar en la figura 3.71.

Si se presiona el botón "Escoger nueva foto" (línea 66 a la 69), se activa el elemento "<input/>" (línea 71) para permitir al usuario escoger la foto a actualizar. Una vez activado el elemento, se llama el método "cargar_imagen()" y se le pasa el evento del elemento que contiene la foto selccionada para que se cargue en el almacén de *firebase*. Se actualiza la variable "url_imagen" para mostrar el cambio de la foto de perfil en tiempo real.

El botón de “Cambiar contraseña” únicamente llama el método “enviar_link()” para mandar un correo al usuario (línea 79). El valor del correo lo tiene almacenado la aplicación siempre.

```
65      <h4>Foto de perfil
66          <div class="btn-group">
67              <button class="btn btn-default" onclick="$('#myInput').click();">
68                  Escoger nueva foto
69              </button>
70          </div>
71          <input id="myInput" type="file" style="visibility:hidden" (change)="cargar_imagen($event)"/>
72      </h4>
73
74      <img src={{url_imagen}} class="img-responsive img-thumbnail img-circle" style="width:20%">
75
76
77      <h1></h1>
78      <div class="btn-group">
79          <button class="btn btn-default" (click)="enviar_link()">
80              Cambiar contraseña
81          </button>
82      </div>
```

Figura 3.71 Definición del Modal para Editar Perfil en “consola-proyectos.component.html” Parte 3

A continuación, se detalla el desarrollo de la barra superior del componente. La barra superior tiene 2 elementos, el botón “Crear Proyecto” y el menú de usuario (figura 3.72).

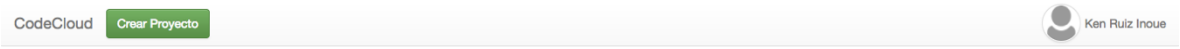

Figura 3.72 Apariencia Barra Superior del Componente Consola Proyectos

El botón de “Crear Proyecto” se definió como aparece en la figura 3.73. En “data-target” (línea 117) contiene el valor del modal para crear proyectos. Al presionarse este botón, se activa el modal correspondiente.

```
116     <button type="button" class="btn btn-success" style="margin-top:8px;"
117     data-toggle="modal" data-target="#modalCrearProyecto">
118       Crear Proyecto
119     </button>
```

Figura 3.73 Definición Botón “Crear Proyecto” en “consola-proyectos.component.html”

El siguiente elemento, el menú de usuario, se definió como un dropdown menú de *bootstrap* (figura 3.74) con 2 opciones, “Edición de Perfil” y “Cerrar Sesión”.

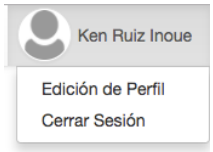

Figura 3.74 Apariencia Menú de Usuario Abierto

El menú de usuario se definió como se puede ver en la figura 3.75. La foto de perfil y el nombre de usuario actuan como un solo botón (línea 127 y 128). Este botón abre las 2 opciones del menú.

La opción de “Edición de Perfil” activa el modal para editar perfil (línea 132). Y la opción “Cerrar Sesión” llama el método “cerrar_sesion()” para notificarle a la aplicación que el usuario quiere abandonar la sesión.

```
123   <ul class="nav navbar-nav navbar-right">
124
125       <li class="dropdown">
126           <a id="user-profile" href="#" class="dropdown-toggle" data-toggle="dropdown">
127               <img src={{url_imagen}} class="img-responsive img-thumbnail img-circle">
128               {{nombre_usuario}}
129           </a>
130           <ul class="dropdown-menu dropdown-block" role="menu">
131               <li>
132                   <a href="javascript:void(0)" data-toggle="modal" data-target="#modalPerfil">
133                       Edición de Perfil
134                   </a>
135               </li>
136               <li><a href="javascript:void(0)" (click)="cerrar_sesion()">Cerrar Sesión</a></li>
137           </ul>
138       </li>
139   </ul>
```

Figura 3.75 Definición Botón “Crear Proyecto” en “consola-proyectos.component.html”

El componente consola proyectos muestra los proyectos del usuario a través de los paneles de proyectos. En la figura 3.76 se muestra la definición del área de proyectos.

Se crean columnas para permitir mostrar hasta 4 paneles por fila (línea 147). Se utiliza el selector “app-panel-proyecto” que le pertenece al componente panel proyecto.

Para los proyectos que le pertenecen al usuario se llama la directiva de la manera como se muestra de la línea 149 a la 151. La directiva “ngFor” se utiliza para recorrer cada elemento almacenado en el arreglo “proyectos_descargados” y crear instancias de cada proyecto con el selector “app-panel-proyecto” (línea 150). Para cada instnacia recorrida, se le pasa al selector 4 entradas; la entrada “proyecto” que contiene la información del proyecto, “correoUsuario”, “usuario” que contiene la información del usuario, y un valor booleano verdadero en la entrada “esPropietario” (línea 149 y 150).

Para los proyectos que el usuario no es propietario, pero si un integrante, se llama el selector “app-panel-proyecto”. También utiliza la directiva “ngFor” para recorrer un arreglo, pero en esta ocasión recorre el contenido del arreglo “proyectos_ajenos_descargados” donde se encuentran almacenados los proyectos ajenos en donde el usuario está como integrante (línea 154). Se pasan las mismas entradas para el selector a excepción de la entrada “esPropietario”, se le pasa un valor booleano falso para esta entrada.

```
147   <div class="row" style="padding-top:30px" >
148     <!-- por cada proyecto que el usuario es propietario, se llama el selector app-panel-proyecto -->
149     <app-panel-proyecto [proyecto]="proyecto"[correoUsuario]="usuario.correo"
150     [usuario]="usuario" [esPropietario]="true" *ngFor="let proyecto of proyectos_descargados">
151     cargando...</app-panel-proyecto>
152     <!-- por cada proyecto en el que se pertenece, se llama el selector app-panel-proyecto -->
153     <app-panel-proyecto [proyecto]="proyecto_ajeno"[correoUsuario]="usuario.correo"
154     [usuario]="usuario" [esPropietario]="false" *ngFor="let proyecto_ajeno of proyectos_descargados_ajenos">
155     cargando...</app-panel-proyecto>
156   </div>
```

Figura 3.76 Definición Área de Proyectos en “consola-proyectos.component.html”

Con esto se concluye la explicación del desarrollo del componente consola proyectos.

3.4.2 Desarrollo del Componente Panel Proyecto

El segundo componente que se desarrolló es el de panel de proyecto. Este componente incluye el contenido del proyecto y permite al usuario realizar acciones sobre él. Por cada proyecto existente, se genera una instancia del componente panel proyecto para representar un único proyecto.

Existen 2 tipos de paneles de proyectos, los que le pertenecen al usuario y los que no le pertenecen pero está participando en ellos.

Si el usuario es propietario del proyecto le aparece una etiqueta de propietario como se muestra en el panel izquierdo de la figura 3.77. De manera contraria, aparece la etiqueta de participante como se puede ver en el panel derecho de la figura 3.77.

También se puede observar en la figura 3.77 en el panel derecho, que existe una etiqueta de integrantes, la cual en su parte inferior muestra la foto de perfil de los integrantes del proyecto.

Figura 3.77 Apariencia de los Panales de Proyecto en la Consola de Proyectos

A continuación se detalla la construcción del componente panel proyecto.

3.4.2.1 Desarrollo Lógico del Componente Panel Proyecto

En la figura 3.78 se observan las importaciones realizadas.

Para manejar métodos del ciclo de vida de la aplicación, se importaron “OnInit” y “AfterViewInit” (línea 1). “Input” funciona para declarar entradas al componente. Se declaran clases de la libería de “AngularFire” en la línea 5. Para poder emplear el servicio “DatosService” se realizó la importación de la misma (línea 10). FileSaver funciona para generar archivos zip (línea 11). Se declararon métodos de la librería Ractive Extension (rxjs) para utilizarlos (línea 13 a la 15). La variable ‘$’ se declaró para permitir el uso de *jQuery*.

```
1   import { Component, OnInit, AfterViewInit, Input } from '@angular/core';
2   import { Router } from '@angular/router';
3   import { Observable } from 'rxjs/Observable';
4
5   import { AngularFire, FirebaseObjectObservable } from 'angularfire2';
6
7   import { Proyecto } from '../../modelos/proyecto';
8   import { NodoArbol } from '../../modelos/nodoArbol';
9
10  import { DatosService } from '../../datos.service';
11  import * as FileSaver from "file-saver";
12
13  import 'rxjs/add/operator/first';
14  import 'rxjs/add/operator/map';
15  import 'rxjs/add/operator/catch';
16
17  declare var $: any;
```

Figura 3.78 Apariencia de los Panales de Proyecto en la Consola de Proyectos

Para este componente se definieron en total 10 variables globales. En la tabla 3.6 se muestran las variables y sus funciones.

Tabla 3.6 Variables Globales del Componente Panel Proyecto

Variable	Función
proyecto	Entrada que guarda información del objeto proyecto
correoUsuario	Entrada que guerda el correo del usuario
usuario	Entrada que guerda información del objeto usuario
esPropietario	Entrada que guarda el estado booleano de propiedad del proyecto
url_archivos	Arreglo que guarda todos los url de los archivos del proyecto
nodos_descargados	Arreglo que almacena todos los nodos del proyecto
almacen	Variable que guarda la referencia del almacen de *firebase*
contenido_id_nodo	Arreglo que guarda el contenido del archivo y su url del proyecto
nombre_integrante_a_eliminar	Variable que guarda el valor del nombre a mostrar en el selector para eliminar integrantes
id_integrante_a_eliminar	Variable que guarda el valor del id del integrante a eliminar
img_url_nombre_integrantes	Arreglo que guarda información de los integrantes, url de la foto de perfil, nombre y el id

A continuación se explican los 16 métodos que contiene el componente panel proyecto.

Método "ngOnInit": Este método al igual que todos los demás componentes, inicializa variables locales y suscripciones. Se puede observar la primera parte del método en la figura 3.79, en este método se descargan todos los nodos del proyecto y se guardan en la variable "nodos_descargados" (línea 61). También se guardan los enlaces de los archivos en la variable "url_archivos" (línea 58).

```
46    ngOnInit() {
47
48      this.url_archivos = [];
49      //obtener la referencia de todos los nodos del proyecto con el id del proyecto
50      let consulta_nodos_proyecto = this.af.database.list('/nodos_proyecto_' + this.proyecto.id);
51
52      //se ejecuta una vez el código debido al método first()
53      consulta_nodos_proyecto.first().subscribe(nodos => {
54
55        for (let nodo of nodos) {
56
57          if(nodo.es_archivo)
58            this.url_archivos.push(nodo.url);
59
60          //guardar todos los nodos del proyecto
61          this.nodos_descargados.push(nodo);
62
63        }
64      });
```

Figura 3.79 Método "ngOnInit" en "panel-proyecto.component.ts" Parte 1

En la segunda parte del método (figura 3.80) se obtienen los proyectos ajenos.

Se consultan todos los integrantes del proyecto (línea 67) para agregarlos en el arreglo "imgurl_nombre_integrante" (línea 76 a la 80). Debido a que se suscribe la consulta (línea 68), cada vez que se agregue o elimine un integrante en el proyecto, se vuelve a ejecutar parte del código (línea 69 a 85) para reflejar los cambios en la interfaz gráfica.

```
66      //obtener la referencia de todos los usuarios con los que se comparte el proyecto
67      let consulta_integrantes_proyecto = this.af.database.list('/proyectos/' + this.proyecto.id + '/integrantes');
68      consulta_integrantes_proyecto.subscribe(integrantes => {//suscribirse al cambio de la base de datos
69
70        //inicializar variables para no duplicar valores
71        this.imgurl_nombre_integrantes = [];
72
73        for(let integrante of integrantes) {
74
75          if(integrante.id != this.usuario.id){//para filtrar la aparición de la img del mismo propietario
76            this.imgurl_nombre_integrantes.push({
77              nombre: integrante.nombre,
78              imgurl: integrante.imgurl,
79              id: integrante.id
80            });
81          }
82
83        }
84
85      });//fin de la suscripción
```

Figura 3.80 Método "ngOnInit" en "panel-proyecto.component.ts" Parte 2

Método 2 "ngAfterViewInit()": En este método se utilizó una función de *jQuery* al igual que el componente consola proyectos para llamar la función "restablecer_menu" cuando el usuario haga click fuera del modal.

```
89      ngAfterViewInit(){
90        let ref_this = this;
91        $('#modalEliminarIntegrante').on('hidden.bs.modal', function (e) {
92          ref_this.restablecer_menu();
93        });
94      }
```

Figura 3.81 Método "ngAfterViewInit" en "panel-proyecto.component.ts"

Método "navegar_area_trabajo(): Cuando el usuario desea trabajar con algún proyecto, se manda a llamar el método "navegar_area_trabajo()" (figura 3.82). Este método utiliza el servicio "DatosService" (línea 98) para guardar el proyecto seleccionado en el servicio. También utiliza el servicio de "Router" (línea 99) para navegar hacia la pantalla de área de trabajo.

```
96      navegar_area_trabajo() {
97        //carga datos del proyecto al servicio DatosService
98        this.ds.proyecto = this.proyecto;
99        this.router.navigateByUrl('/area-trabajo');
100     }
```

Figura 3.82 Método "navegar_area_trabajo" en "panel-proyecto.component.ts"

Método "eliminar_proyecto()": Como se puede ver en la figura 3.83, para eliminar un proyecto, se utiliza el método "eliminar_proyecto()". Este método remueve cada archivo existente dentro del proyecto a través del método "eliminar_archivo()" (línea 104). Elimina la referencia en "/nodos_proyecto_id" (línea 107) y "/proyectos/id" (línea 108) con la *API* de *firebase*.

```
102     eliminar_proyecto() {
103       for (let url of this.url_archivos) {
104         this.eliminar_archivo(url);
105       }
106
107       this.af.database.object('/nodos_proyecto_' + this.proyecto.id).remove();
108       this.af.database.object('/proyectos/' + this.proyecto.id).remove();
109     }
```

Figura 3.83 Método "eliminar_proyecto" en "panel-proyecto.component.ts"

Método "eliminar_archivo()": Una vez que se llama este método, previamente se tuvieron que haber cargado todas las ligas correspondientes del proyecto a eliminar en la variable "url_archivos". En la figura 3.84 se muestra a detalle el contenido del método "eliminar_archivo()". Este método recibe como parámetro la dirección (url) del arcivo a eliminarse. Se obtiene la referencia https con la *API* de *firebase* (línea 112 y 113).

```
111     eliminar_archivo(url) {
112       let httpsReference = this.almacen.refFromURL(url);
113       httpsReference.delete().then(function() {
114         console.log('archivo eliminado');
115       }).catch(function(error) {
116         console.log('ocurrió un error');
117       });
118     }
```

Figura 3.84 Método "eliminar_archivo()" en "panel-proyecto.component.ts"

A continuación se explican las relaciones que tienen las funciones para llevar a cabo el proceso de descarga del proyecto (figura 3.85).

Como se puede observar en la figura 3.85, en total se utilizan 6 métodos para que el usuario pueda recibir un arhivo *zip* que contenga todos las carpetas y archivos del proyecto

seleccionado. Para ofrecer la funcionalidad de descarga al usaurio, fue necesario simular una comunicación síncrona entre los métodos de manera asíncrona.

Por defecto, *Angular* ofrece un marco de trabajo asíncrono, esto para permitir al usuario interactuar con la aplicación de manera continua, de no ser así (síncrono), la interfaz de usuario no respondería ante las acciones del usuario cada vez que la aplicación realiza una petición con http (conexión a internet) y no haya concluido la petición.

Para llevar a cabo el proceso de descarga del proyecto, es indispensable descargar los archivos del proyecto almacenados en el repositorio de *firebase* a través de internet. Una vez descargados los archivos, se construye un archivo *zip* con las carpetas y archivos correspondientes dentro de la aplicación y finalmente inicia la descarga del archivo *zip* para el usuario. El problema que surge ante este proceso es que la aplicación ejecuta el método encargado en descargar los archivos desde el repositorio y de manera inmediata ejecuta el método encargado en construir el archivo *zip*, lo cual significa que sin importar que haya concluido el método que descarga los archivos, la aplicación procede a construir el archivo *zip*.

Para solucionar este problema se utilizó el objeto *Promise* (promesa) que ofrece *JavaScript*. Este objeto permite que métodos asíncronos retornen valores de manera síncrona. Así el método "bucle_promesa()" que se encarga de la descarga de los archivos junto con el método "promesa_descarga()", no llama el método "construir_proyecto()" (generación del archivo *zip*) hasta que termine de descargar todos los archivos pertenecientes del proyecto.

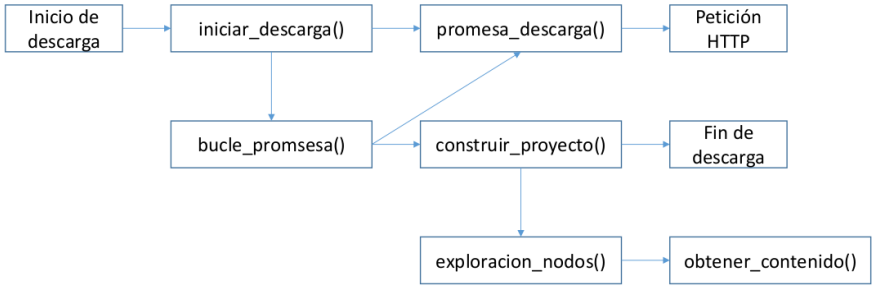

Figura 3.85 Relaciones entre los Métodos para el Proceso de Descarga

Método "iniciar_descarga()": Para comenzar el proceso de descarga se llama este método. A continuación se detalla la primera parte del método (figura 3.86).

Primero se inicializa la variable "contenido_id_nodo" para no encimar valores previamente almacenados (línea 122), se declaran algunas variables auxiliares (línea 122 a la 127) y se comienza a explorar cada nodo descargado para guardar todos los archivos del proyecto en la variable "nodo_archivos" (línea 133) y guardar la cantidad de archivos en la variable "cantidad_archivos" (línea 134).

```
120     iniciar_descarga() {
121
122       this.contenido_id_nodo = [];//iniciarlizar variable
123       var contador = 0;
124       var cantidad_archvios = 0; //determina cuando detener la iteración
125       var nodo_archivos = [];//guardar todos los nodos que sean archivos en esta variable
126
127       var ref_this = this;
128
129       //iterar entre todos los nodos que contenga el proyecto
130       for(let nodo of this.nodos_descargados) {
131         if (nodo.es_archivo == true) {
132           //en el caso de ser un archivo se agrega en el arreglo
133           nodo_archivos.push(nodo);
134           cantidad_archvios ++;
135         }
136       }
```

Figura 3.86 Método "iniciar_descarga()" en "panel-proyecto.component.ts" Parte 1

En la segunda parte del método (figura 3.87), se manda a llamar el método "promesa_descarga()" para iniciar la descarga del primer archivo del proyecto (línea 138).

Una vez que termine de ejecutarse el método "promesa_descarga()", debido a que es un objeto *Promise*, retorna el contenido del archivo que se acaba de descargar en la variable "contenido" (línea 139) y ejecuta el código de la línea 139 a la 145 en caso de tener éxito con la descarga del archivo. Se agrega el contenido del archivo y el id en el arreglo "contenido_id_nodo" (línea 141). Se llama el método recursivo "bucle_promesa()" (línea 143) para seguir con los archivos que restan.

```
138       this.promesa_descarga(nodo_archivos[contador].url).then(
139         function(contenido) {
140
141           ref_this.contenido_id_nodo.push({contenido: contenido, id: nodo_archivos[contador].id});
142           contador++;
143           ref_this.bucle_promesa(nodo_archivos, contador, cantidad_archvios);
144
145         }, function(rechazo) {
146           console.log(rechazo);
147         }
148       );
149
150     }// fin del método
```

Figura 3.87 Método "iniciar_descarga()" en "panel-proyecto.component.ts" Parte 2

Método "promesa_descarga()": El método "promesa_descarga()" retorna un objeto tipo *Promise* para responder en cuanto termine de ejecutarse como se puede observar en la figura 3.89 (línea 158).

Se crea una referencia para la petición de https con el url que recibe como parámetro (línea 160) y se inicia el proceso de descarga con el método "getDownloadURL()" de *firebase*. Una vez iniciada la descarga seguido del método "then()", se obtiene la variable "url" que apunta la dirección del archivo. Para leer el contenido del archivo desde la url retornada se crea una variable de petición llamada "xhr" y se le asigna el tipo blob como tipo de respuesta.

```
156   //promesa que regresa el contenido del archivo desde el url
157   promesa_descarga(url) {
158     return new Promise((resolve, reject) => {
159
160       var referenciaHttps = this.almacen.refFromURL(url);
161
162       referenciaHttps.getDownloadURL().then(function(url) {
163
164         // descargar el archivo y guardarlo en BLOB
165         var xhr = new XMLHttpRequest();
166         xhr.responseType = 'blob';
```

Figura 3.88 Método "promesa_descarga()" en "panel-proyecto.component.ts" Parte 1

A continuación se muestra la definición de la descarga que se le asigna a la variable "xhr" de la línea 168 a la 175 (figura 3.89). Lee el contenido del archivo (línea 171) y retorna el contenido (línea 172). Una vez definido el proceso de descarga, se manda a llamar con el método "send()" (línea 178).

```
168         xhr.onload = function(event) {
169           var blob = xhr.response;
170           var reader = new FileReader();//leer el contenido del BLOB
171           reader.onload = function(event){
172             resolve(reader.result);//regresa el contenido del archivo
173           };
174           reader.readAsText(blob);
175         };
176
177         xhr.open('GET', url);
178         xhr.send();
```

Figura 3.89 Método "promesa_descarga()" en "panel-proyecto.component.ts" Parte 2

Método 8: El método "bucle_promesa()" es un método que se vuelve a llamar a sí mismo las veces que sea necesario hasta que termine de recorrer la lista de archivos encontrados en el proyecto (figura 3.90).

Recorre cada archivo necesario y una vez que termine la descarga, procede con el archivo consecuente, así simulando un proceso síncrono. Una vez que el método "promesa_descarga()" retorna el contenido el archivo, se agrega un objeto tipo JSON en la vairable "contenido_id_nodo" (línea 194). Aumenta el contador para que el método sea capaz de apuntar el siguente archivo a procesar (195).

```
188   //función recursiva para descargar el contenido de manera síncrona
189   bucle_promesa(nodo_archivos, contador, cantidad_archvios) {
190
191     var ref_this = this;
192     this.promesa_descarga(nodo_archivos[contador].url).then(function(contenido) {
193
194       ref_this.contenido_id_nodo.push({contenido: contenido, id: nodo_archivos[contador].id});
195       contador++;
```

Figura 3.90 Método "bucle_promesa()" en "panel-proyecto.component.ts" Parte 1

Si el contador no excede la cantidad de la variable "cantidad_archivos", significa que todavía existen archivos a procesar (figura 3.91).

El método "bucle_promesa" al terminar el ciclo, es decir, cuando el contador llega a ser igual o mayor que la variable "cantidad_archivos" llama el método "construir_proyecto()" para generar el archivo *zip* que contiene los archivos del proyecto.

```
197         if(contador < cantidad_archvios) {
198           ref_this.bucle_promesa(nodo_archivos, contador, cantidad_archvios);
199         } else {//termina ciclo
200           ref_this.construir_proyecto();
201         }
202
203       }, function(rechazo) {
204         console.log(rechazo);
205       });
206     }
```

Figura 3.91 Método "bucle_promesa()" en "panel-proyecto.component.ts" Parte 2

Método "exploracion_nodo()": Este método utiliza 2 librerias, la de "jszip" para generar archivos *zip* y la de "FileSaver" para guardar contenido descargado en el lado del cliente (navegador) y permitir la descarga del contenido generado al usuario. Este método acomoda de manera correcta el contenido descargado de los archivos para generar un solo archivo *zip*.

Método "compartir_proyecto()": El usuario además de descargar o eliminar el proyecto, puede compartirlo con diferentes usuarios, esa acción se puede llevar a cabo con este método (figura 3.92).

El método recibe como parámetro el correo del usuario que se desea agregar en el proyecto. Se realiza una consulta de todos los usuarios existentes y se compara con el parámetro recibido (línea 293), de ser verdadero el caso, se crea una consulta para generar un nuevo registro del nuevo usuario en la ubicación de "…/integrantes" (línea 294) en la base de datos de *firebase* (de la línea 295 a la 300).

```
289       //se ejecuta una vez el código debido al método first()
290       consulta_usuarios.first().subscribe(usuarios => {
291
292         for (let usuario of usuarios) {
293           if(usuario.correo == correo) {
294             let consulta_proyecto = this.af.database.object('proyectos/' + this.proyecto.id+'/integrantes/' + usuario.id);
295             consulta_proyecto.set({
296               nombre: usuario.nombre,
297               imgurl: usuario.imgurl,
298               id: usuario.id,
299               propietarioProyecto: false
300             });
301             alert("Se compartió el proyecto con el usuario de manera existosa");
302             return
303           }
304         }
```

Figura 3.92 Contenido Importante del Método "compartir_proyecto()"

Métodos "seleccionar_integrante", "eliminar_usuario()" y "restablecer_menu()": Una vez compartido el proyecto con otros usuarios, el propietario puede tomar la decisión de dar de baja al usuario que ya no requiera que participe en el proyecto. Para eso se emplea el método "seleccionar_integrante()", el método "eliminar_usuario()" y el método "restablecer_menu()". En la figura 3.93 se observa cómo interactúan los métodos mencionados. El primer método que se llama es el de "seleccionar_integrante()". Este método guarda en la aplicación la información del usuario a eliminar por parte del propietario. En la parte de desarrollo gráfico de este componente, se detalla la interacción del usuario con la interfaz gráfica para explicarlo con más detalle.

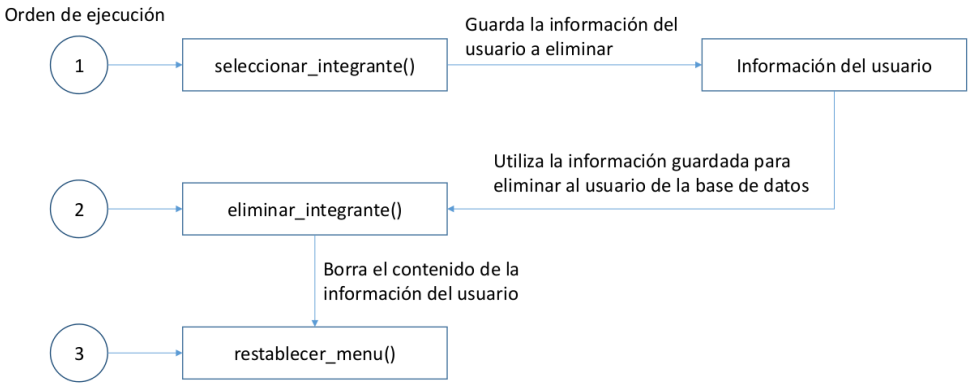

Figura 3.93 Relación de los Métodos para Eliminar un Integrante

Método "eliminar_integrante()": Una vez llamado el método "seleccionar_integrante()", a continuación se llama este método para remover el registro del usuario en la base de datos (figura 3.94).

Se verifica que la información guardada del usuario no contenga valores indefinidos (línea 316). Se crea la referencia en la ruta "/proyectos/{id_proyecto}/integrantes/{id_integrante}" de la base de datos y se llama el método "remove()" para eliminar el registro. Después de eliminar el integrante, se le notifica al usuario con una alerta (línea 320) y se llama el método "restablecer_menu()" (Método 13) para eliminar la información guardada en la aplicación con el método "seleccionar_integrante()".

```
315     eliminar_integrante() {
316       if(this.id_integrante_a_eliminar != undefined) {
317         this.af.database.object('/proyectos/' + this.proyecto.id +'/integrantes/' + this.id_integrante_a_eliminar).remove()
318         .then(() =>
319           {
320             alert('El integrante se dió de baja');
321             this.restablecer_menu();
322           }
323           );
324       }
```

Figura 3.94 Contenido Importante del Método "eliminar_integrante()"

Método "abandonar_proyecto()": Finalmente el usuario puede decidir si quiere abandonar el proyecto en el que está participando a través de este método (figura 3.95). Este método elimina el usuario de la ruta al igual que el método "eliminar_integrante()" (línea 336).

```
335     abandonar_proyecto(){
336       this.af.database.object('/proyectos/' + this.proyecto.id +'/integrantes/' + this.usuario.id).remove()
337         .then(() => alert('Abandonaste el proyecto con éxito'));
338     }
```

Figura 3.95 Método "abandonar_proyecto()" en "panel-proyecto.component.ts"

Con esto se concluye la explicación del desarrollo lógico del componente panel proyecto.

3.4.2.2 Desarrollo Gráfico del Componente Panel Proyecto

Como se había mencionado, el componente panel proyecto proporciona la interfaz gráfica al usuario para que pueda interactuar con el proyecto.

El encabezado del panel de proyecto está definido como aparece en la figura 3.96. Este encabezado obtiene el nombre del proyecto de la variable "proyecto" (línea 78). El botón de configuración ofrece en total 5 opciones para el usuario (línea 86 a la 90). La primera opción "Descargar Proyecto" llama la función "iniciar_descarga()" para comenzar la descarga y ofrecer el archivo *zip* al usuario (línea 86). La segunda opción "Eliminar Proyecto" llama la función "eliminar_proyecto()" para remover el proyecto del registro del usuario (línea 87). La tercera opción "Compartir Proyecto" aparece únicamente si la variable "esPropietario" contiene un valor booleano verdadero, para que únicamente el propietario pueda compartir el proyecto, esta opción activa el modal "modalCompartir" para introducir el correo del usuario a compartir el proyecto (línea 88). La cuarta opción aparece si es el propietario del proyecto al igual que la tercera opción, al presionarse activa el modal "modalEliminarIntegrante" para ofrecer un selector (menú) de los integrantes pertenecen al proyecto y un botón para confirmar la acción (línea 89). La quinta y última opción únicamente aparece si el valor de la variable "esPropietario" es un booleano falso, para permitir abandonar el proyecto al ser un integrante y no el propietario, con la llamada del método "abandonar_proyecto()".

```
76        <!-- contenido del panel -->
77        <h3 class="panel-title">
78          {{proyecto.nombre}}
79
80          <!-- botón -->
81          <div class="btn-group  pull-right">
82            <a class="dropdown-toggle" href="javascript:void(0)" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false" style="color: ■black">
83              <span class="glyphicon glyphicon-cog"></span>
84            </a>
85            <ul class="dropdown-menu">
86              <li><a href="javascript:void(0)" (click)="iniciar_descarga()">Descargar Proyecto</a></li>
87              <li><a href="javascript:void(0)" (click)="eliminar_proyecto()">Eliminar Proyecto</a></li>
88              <li><a href="javascript:void(0)" data-toggle="modal" data-target="#modalCompartir">Compartir Proyecto</a></li>
89              <li *ngIf="esPropietario"><a href="javascript:void(0)" data-toggle="modal" data-target="#modalEliminarIntegrante">Eliminar Integrante</a></li>
90              <li *ngIf="!esPropietario"><a href="javascript:void(0)" (click)="abandonar_proyecto()">Abandonar Proyecto</a></li>
91            </ul>
92          </div>
93
94        </h3>
```

Figura 3.96 Definición del Encabezado Panel Proyecto en "panel-proyecto.component.html"

El encabezado del panel de proyecto tiene la apariencia como se muestra en la figura 3.97 (propietario del lado izquierdo y no propietario del lado derecho).

Como se puede observar, la acción "Descargar Proyecto" y "Eliminar Proyecto" se pueden encontrar en el caso de ser propietario o no. La acción "Compartir Proyecto" y "Eliminar Integrante" únicamente aparecen en el caso de ser el propietario. La acción "Abandonar Proyecto" aparece si no es el propietario del proyecto.

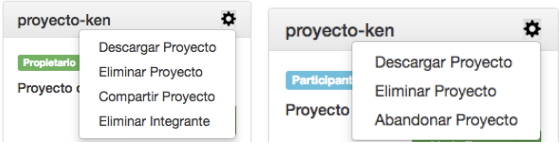

Figura 3.97 Apariencia del Encabezado del Panel de Proyecto

A continuación se detalla la definición del modal “modalCompartir”. El formulario se definió como se muestra en la figura 3.98. Es un elemento tipo “<form></form>” y se definió como variable local “correo” (línea 17).

```
13          <form id="loginform" class="form-horizontal" role="form">
14              <div style="margin-bottom: 25px" class="input-group">
15                  <span class="input-group-addon"><i class="glyphicon glyphicon-envelope"></i></span>
16                  <input id="campo-correo-compartir" type="text" class="form-control"
17                  placeholder="correo" #correo>
18              </div>
19          </form>
```

Figura 3.98 Definición del Formulario del Modal “modalCompartir”

En la figura 3.99 se puede ver la definición del botón de confirmación (línea 23) que llama la función “compartir_proyecto()” pasando como parámetro el valor que contiene el formulario de la figura 3.98.

```
21        <div class="modal-footer">
22          <button type="button" class="btn btn-default" data-dismiss="modal">Cancelar</button>
23          <button type="button" class="btn btn-success" (click)="compartir_proyecto(correo.value)" data-dismiss="modal">Compartir</button>
24        </div>
```

Figura 3.99 Definición del Botón de Confirmación del Modal “modalCompartir”

El modal “modalCompartir” tiene la apariencia como se observa en la figura 3.100.

Figura 3.100 Apariencia del Modal “modalCompartir”

El modal “modalEliminarIntegrante” contiene un selector que recibe como entrada un arreglo de los integrantes que pertenecen en el proyecto para poder ser seleccionados. En la figura 3.101 se observa la definición del selector. En las líneas 47 y 48 se definen las opciones de selección a través de la directiva “ngFor” de la variable “imgurl_nombre_integrantes”. La función “seleccionar_integrante()” pasa como parámetros el nombre y el identificador del usuario que seleccionado. Cada opción aparece con el nombre del integrante (línea 49) a través da la propiedad “nombre”.

```
40        <div class="dropdown">
41            <button class="btn btn-default dropdown-toggle" type="button" id="dropdownIntegrantes"
42            data-toggle="dropdown" aria-haspopup="true" aria-expanded="true">
43                {{nombre_integrante_a_eliminar}}
44                <span class="caret"></span>
45            </button>
46            <ul class="dropdown-menu" aria-labelledby="dropdownIntegrantes">
47                <li *ngFor="let integrante of imgurl_nombre_integrantes"
48                (click)="seleccionar_integrante(integrante.nombre, integrante.id)">
49                  <a  href="javascript:void(0)">{{integrante.nombre}}
50                  </a>
51                </li>
52            </ul>
53        </div>
```

Figura 3.101 Definición del Selector en el Modal "modalEliminarIntegrante"

A continuación se muestra la definición del pie del modal en la figura 3.102. El botón "Cancelar" llama la función "restablecer_menu()" para borrar la información persistente del usuario seleccionado. El botón "Dar de baja" llama el método "eliminar_integrante()" para remover el registro del usuario seleccionado.

```
57        <div class="modal-footer">
58          <button type="button" class="btn btn-default" data-dismiss="modal" (click)="restablecer_menu()">Cancelar</button>
59          <button type="button" class="btn btn-danger" (click)="eliminar_integrante()" data-dismiss="modal">Dar de baja</button>
60        </div>
```

Figura 3.102 Definición del Pie del Modal "modalEliminarIntegrante"

El modal "modalEliminarIntegrante" tiene la apariencia como se muestra en la figura 3.103.

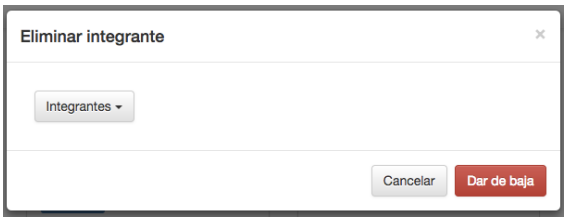

Figura 3.103 Apariencia del Modal "modalEliminarIntegrante"

El cuerpo del panel de proyecto se definió como se muestra en la figura 3.104. El panel dependiendo de si el usuario es propietario o no del proyecto, muestra la etiqueta correspondiente (línea 99 y 100). Si la propiedad "descripcion" de la variable "proyecto" es igual a nula, muestra la descripción por defecto (línea 101 y línea 102), de manera contratia muestra la descripción del proyecto. El botón "Abrir Proyecto" llama la función "navegar_area_trabajo()" para navegar el usuario hacia el módulo de área de trabajo.

```
99    <span class="label label-success" *ngIf="esPropietario">Propietario</span>
100   <span class="label label-info" *ngIf="!esPropietario">Participante</span>
101   <h5 *ngIf="proyecto.descripcion == null">
102   Proyecto sin descripción.
103   </h5>
104   <h5 *ngIf="!(proyecto.descripcion == null)">
105   {{proyecto.descripcion}}
106   </h5>
107   <div class="btn-group pull-right">
108     <button type="button" class="btn btn-success" (click)="navegar_area_trabajo()">Abrir Proyecto</button>
109   </div>
```

Figura 3.104 Definición del Cuerpo del Panel de Proyecto

La apariencia del cuerpo de panel de proyecto es como se observa en la figura 3.105 (propietario del lado izquierdo y no propietario del lado derecho).

Figura 3.105 Apariencia del Cuerpo del Panel de Proyecto

Finalmente, el pie del panel de proyecto se definió como se puede observar en la figura 3.106. Si la variable "imgurl_nombre_integrantes" no es igual a cero, significa que existen participantes en el proyecto y aparece una etiqueta con el contenido "Integrantes" (línea 115). En la línea 119 y 120 se llama el selector "app-integrante" para llamar el componente integrante. Se utiliza la directiva "ngFor" para iterar y generar cada instancia de todos los integrantes que contenga la variable "imgurl_nombre_integrantes" (línea 120). El selector "app-integrante" como parámetro recibe la variable "imgurl_nombre" que tiene como propietad el url de la foto de perfil y el nombre del usuario.

```
115     <span class="label label-primary" *ngIf="imgurl_nombre_integrantes.length != 0">Integrantes</span>
116     <br>
117
118     <div style="margin-top: 10px; margin-left: 12px;">
119       <app-integrante class="col-sm-3" style="margin-right: 5px;" [datosIntegrante]="imgurl_nombre"
120       *ngFor="let imgurl_nombre of imgurl_nombre_integrantes"></app-integrante>
121     </div>
```

Figura 3.106 Definición del Pie del Panel de Proyecto

El pie de panel de proyecto tiene la apariencia como se puede ver en la figura 3.107. Al colocar el cursor encima de la figura (globo terraqueo), aparece una etiqueta con el nombre del usuario.

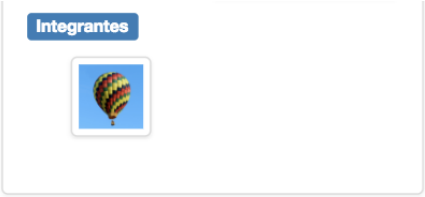

Figura 3.107 Apariencia del Pie del Panel de Proyecto

Con esto se concluye la explicación del desarrollo gráfico del componente panel de proyecto.

3.4.3 Desarrollo del Componente Integrante

El componente integrante es el desarrollo más corto de este módulo. En la figura 3.108 se puede ver la definición lógica de este componente. Este componente recibe como parámetro la variable "datosIntegrante" del componente panel proyecto.

En el método “ngAfterViewInit()” se definió una función para mostrar la etiqueta con el nombre del usuario (línea 20 y 21). Esta función muestra la etiqueta con el identificador “tooltip” cuando percibe el cursor encima del componente.

```
17    @Input() datosIntegrante;
18
19    ngAfterViewInit() {
20      $(function () {
21        $('[data-toggle="tooltip"]').tooltip()
22      })
23    }
```

Figura 3.108 Definición Lógica Importante del Componente Integrante

La definición gráfica del componente integrante contiene únicamente la etiqueta de imagen como se puede ver en la figura 3.109. La etiqueta <img></img> obtiene el url de la foto de perfil y el nombre del usuario de la variable “datosIntegrante” que es el parámetro que recibe este componente. En el atributo “data-toggle” se definió el valor “tooltip” para permitir a la función de la figura 3.108 identificar la etiqueta a mostrar. El valor que muestra la etiqueta se encuentra definido en el atributo “title” (línea 2).

```
1  <img src={{datosIntegrante.imgurl}} class="thumbnail portrait"
2  data-toggle="tooltip" data-placement="bottom" title={{datosIntegrante.nombre}} />
```

Figura 3.109 Definición Gráfica del Componente Integrante

La apariencia de la etiqueta se puede ver en la figura 3.110.

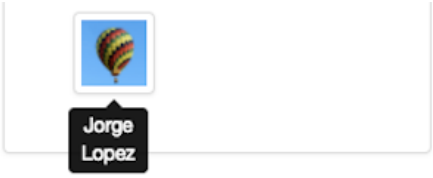

Figura 3.110 Apariencia de la Etiqueta en el Componente Integrante

Con esto se concluye el desarrollo del módulo de consola de proyectos.

3.5 Desarrollo del Módulo de Área de Trabajo

El módulo de área de trabajo es el último módulo del sistma y la principal razón por la cual se decidió utilizar el marco de trabajo de *Angular*. El módulo está diseñado para que el usuario final utilice esta sección más que las demás, ya que es la parte de la aplicación que brinda la capacidad de editar, compilar y ejecutar programas.

Este módulo contiene 3 componentes diferentes. El componente área-trabajo, editor y topbar. Como se observa en la figura 3.111, el componente área-trabajo emplea el componente editor y topbar para conformar el módulo de área de trabajo.

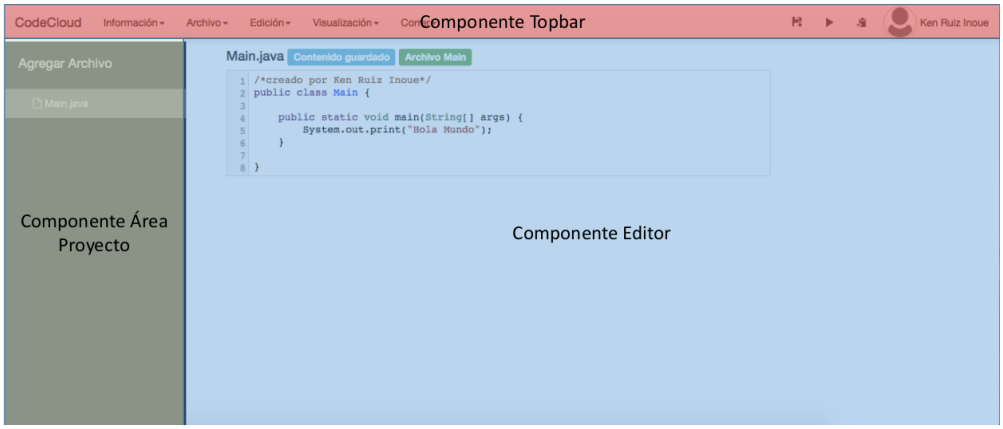

Figura 3.111 Distribución de los Componentes del Módulo de Área de Trabajo

3.5.1 Desarrollo del Componente Área Trabajo

El componente área-trabajo contiene la barra lateral y se comporta como el administrador principal del módulo para comunicar el componente editor y topbar. Además de fungir como el administrador principal del módulo área trabajo, se encarga de activar todos los modals correspondientes a este módulo.

3.5.1.1 Desarrollo Lógico del Componente Área Trabajo

El componente área trabajo utiliza 21 métodos para cumplir con sus funcionalidades. En la tabla 3.7 se muestran y detallan los métodos utilizados.

Tabla 3.7 Métodos del Componente Área Trabajo

Método	Funcionalidad
ngOnInit()	Inicializa variables y obtiene datos del proyecto seleccionado y sus archivos correspondientes.
ngAfterViewInit()	Declara los eventos de los modals y crea una instnacia del editor de texto con la librería de *CodeMirror*.
limpiar_campo_archivo()	Le asigna un valor vacío en el campo de introducción del modal y un valor booleano

	falso en la variable para aprobar el campo de texto.
validar_nombre_archivo()	Comprueba que el campo de texto para crear archivos contenga únicamente caracteres y números sin espacios y que no se introduzca el nombre de un archivo existente.
contiene_letras_numeros_unicamente()	Retorna un valor booleano dependiendo si la entrada cumple o no con la expresión regular.
crear_archivo()	Crea el registro en la base de datos de *firebase* del archivo generado y lo carga en el repositorio de *firebase*.
seleccionar_archivo()	Guarda la referencia del archivo seleccionado en una variable global y comprueba si hay necesidad de mostrar una advertencia en el caso que no se haya guardado un archivo editado.
comunicar_con_editor()	Inicializa algunas variables de control y manda a llamar el método "proporcionar_datos_archivo()" y "descargar_contenido_archivo()".
proporcionar_datos_archivo()	Le proporciona al componente editor los datos del archivo seleccionado.
descargar_contenido_archivo()	Descarga el contenido del archivo en el repositorio a través de la *API* de *firebase* y lo carga en el editor de texto.
toggle_area_proyecto()	Muestra o esconde el área de proyecto según el estado de la variable "si_esconder_area_proyecto".
cambiar_estado_area_ejecucion()	Muestra o esconde el área de ejecución según el estado de la variable "si_esconder_area_ejecucion".
cambiar_estado_checkbox()	Cambia el estado booleano de la variable "estado_checkbox" según el parámetro que recibe.
eliminar_archivo()	Elimina el archivo seleccionado del registro de la base de datos y repositorio de *firebase*.
abrir_modal_crear_archivo()	Abre el modal para crear archivos.
convertir_archivo_en_main()	Obtiene la referencia del archivo seleccionado y actualiza la propiedad "es_main" en la base de datos de *firebase*.
validar_nombre_plantilla()	Comrpueba que en el campo de texto para el nombre en el modal crear plantilla, contenga letras y números sin espacios.
limpiar_campos_plantilla()	Le asigna valores nulos a los campos utilizados para crear plantillas nuevas.

crear_plantilla()	Inserta el registro de la nueva plantilla en la base de datos de *firebase*.
colocar_texto_default_editor()	Limpia el campo de texto para la plantilla e inserta el texto por defecto.
eliminar_plantilla()	Elimina la plantilla seleccionada del registro de la base de datos de *firebase*.

Método "ngOnInit()": Este es el primer método que se manda a llamar. En la figura 3.112 se observa la primera parte de la definición del método. Al entrar al método se esconde el área de proyecto y ejecución (líneas 68 y 70). Se genera una consulta de los archivos del proyecto para monitorear los cambios.

```
63    ngOnInit() {
64
65        //obtener datos del proyecto de DatosServicio
66        this.proyecto = this.ds.proyecto;
67
68        this.ds.si_esconder_area_proyecto = false;
69        //al abrir el proyecto, esconder el área de ejecución
70        this.ds.si_esconder_area_ejecucion = true;
71
72        //obtener los archivos del proyecto
73        const consulta_archivos_proyecto = this.af.database.list('/archivos_proyecto_' + this.proyecto.id);
```

Figura 3.112 Método "ngOnInit()" en "area-trabajo.component.ts" Parte 1

Después se crea una suscripción como se observa en la figura 3.113. Cada vez que se elimina o se genera un nuevo archivo, se acutaliza el arreglo que contiene los archivos para reflejarse en la interfaz gráfica (línea 92). En el caso que exista un archivo main (línea 88), se guarda un valor booleano verdadero en una bandera para avisar al sistema que existe un archivo main en el proyecto. Esta bandera sirve para no permitir al usuario final generar más de un archivos main o mostrarle advertencias respecto al archivo main.

```
75        //cada vez que se modifica la estructura de proyecto se activa esta parte del codigo
76        consulta_archivos_proyecto.subscribe(archivos => {
77
78            //inicializamos la variable para guardar el contenido de los archivos
79            this.ds.archivos_proyecto = [];
80            this.ds.si_existe_main = false;
81            this.ds.si_proyecto_vacio = true;
82
83            //guardar en el arreglo los archivos
84            for (let archivo of archivos) {
85                //si entra 1 vez ya no está vacío el proyecto
86                this.ds.si_proyecto_vacio = false;
87                //verifica que existe el archivo main
88                if(archivo.es_main) {
89                    this.ds.si_existe_main = true;
90                    this.ds.archivo_main = archivo;//guarda el contenido del archivo main para abrirlo
91                }
92                this.ds.archivos_proyecto.push(archivo);
93            }
```

Figura 3.113 Método "ngOnInit()" en "area-trabajo.component.ts" Parte 2

En la última parte del método "ngOnInit()" se ordenan los archivos obtenidos en órden alfabético como aparece en la figura 3.114 (líneas 96 a la 100). Después limpia el arreglo que

contiene los mensajes de ejecución (línea 104) y la bandera que guarda el estado de comunicación con el servidor (línea 105).

```
95          //algotirmo para ordenar de manera alfabética los archivos por la propiedad nombre_archivo
96          this.ds.archivos_proyecto.sort(function(a, b){
97              if(a.nombre_archivo < b.nombre_archivo) return -1;
98              if(a.nombre_archivo > b.nombre_archivo) return 1;
99              return 0;
100         })
101     });
102
103     //vaciar los mensajes si es que existen registros de otro proyecto
104     this.ds.mensajes_resultado_registro = [];
105     this.ds.comunicandose_con_el_servidor = false;
106
107 }
```

Figura 3.114 Método "ngOnInit()" en "area-trabajo.component.ts" Parte 3

Método "ngAfterViewInit()": Después que se ejecuta el método "ngOnInit()", se ejecuta este método. En la figura 3.115 se observa la primera parte de la definición del método. En este método se definen 2 eventos del modal "crearArchivo" y "crearPlantilla" respectivamente. Estos eventos se disparan cada vez que el modal se oculta para limpiar los campos de textos relacionados a los modals.

```
109 ngAfterViewInit() {
110     var ref = this;
111     //reacciona ante los eventos hidden
112     $('#modalCrearArchivo').on('hidden.bs.modal', function (e) {
113         ref.limpiar_campo_archivo();
114     });
115
116     $('#modalCrearPlantilla').on('hidden.bs.modal', function (e) {
117         ref.limpiar_campos_plantilla();
118     });
119
120     $("#wrapper").toggleClass("toggled");
```

Figura 3.115 Método "ngOnInit()" en "area-trabajo.component.ts" Parte 1

En la segunda parte de la definición del método "ngOnInit()" (figura 3.116) se muestra cómo se abre el archivo main (líneas 123 a la 126). y la creación del editor de texto del modal "crearPlantilla" con la librería *CodeMirror* (líneas 128 a la 132).

```
122     //abrir archivo main si es que existe cuando entra por primera vez
123     if(this.ds.si_existe_main && this.entra_por_primera_vez) {
124         this.seleccionar_archivo(this.ds.archivo_main);
125         this.entra_por_primera_vez = false;
126     }
127
128     this.editor = CodeMirror.fromTextArea(document.getElementById("editor-modal-crear-plantilla"), {
129         mode: "text/x-java",
130         styleActiveLine: true,
131         matchBrackets: true
132     });
133     this.colocar_texto_default_editor();
134
135 }
```

Figura 3.116 Método "ngOnInit()" en "area-trabajo.component.ts" Parte 2

A continuación, se detallan métodos relevantes que utiliza el componente área-trabajo para cumplir con sus funcionalidades.

Método "validar_nombre_archivo()": Este método como se observa en la figura 3.117, busca si algún nombre de los archivos existentes coincide con el nombre del archivo a crear para verificar si aprobar o no el campo de texto (líneas 145 a la 150).

```
142     validar_nombre_archivo() {
143         this.nombre_archivo_unico = true;
144
145         for(let archivo of this.ds.archivos_proyecto) {
146             if(this.valor_campo_archivo + '.java' == archivo.nombre_archivo) {
147                 this.nombre_archivo_unico = false;
148                 return
149             }
150         }
151         this.campo_archivo_aprobado = this.contiene_letras_numeros_unicamente(this.valor_campo_archivo);
152     }
```

Figura 3.117 Método "validar_nombre_archivo()" en "area-trabajo.component.ts"

Método "contiene_letras_numeros_unicamente()": Como se muestra en la figura 3.118, este método utiliza una expresión regular para validar si el parámetro recibido cumple o no con la regla. La expresión regular que se encuentra en la línea 155, verifica que exista únicamente caracteres consecutivos alfanuméricos.

```
154     contiene_letras_numeros_unicamente(expresion) {
155         return /^\w+$/.test(expresion);
156     }
```

Figura 3.118 Método "contiene_letras_numeros_unicamente()" en "area-trabajo.component.ts"

Método "crear_archivo()": Este método tiene 3 funciones importantes. La primera función es la de crear el contenido del archivo a cargarse en el repositorio de *firebase*. Como se observa en la figura 3.119, si el usuario desea crear un archivo main, se ejecuta el código de las líneas 165 a 172.

```
158     crear_archivo() {
159         var nombre_archivo = this.valor_campo_archivo;
160         //obtener referencia al this para poder utilizarlo en el siguiente nivel de scope
161         let ref = this;
162         var archivo;
163
164         if(this.estado_checkbox) {//al crear main
165             archivo = new Blob(['/*creado por ' + this.ds.usuario.nombre + '*/\n'
166             + 'public class ' + nombre_archivo + ` {
167
168     public static void main(String[] args) {
169         System.out.print("Hola Mundo");
170     }
171
172 }`], {type: 'text/plain'});
```

Figura 3.119 Método "crear_archivo ()" en "area-trabajo.component.ts" Parte 1

Si el usuario ya ha creado un archivo main o no desea crearlo, se ejecuta la parte del código como se muestra en la figura 3.120. En esta parte del código, no se crea el archivo con el método main.

```
173         } else {//al crear una clase básica
174             archivo = new Blob(['/*creado por ' + this.ds.usuario.nombre + ' */\n'
175             + 'public class ' + nombre_archivo +
176             ` {
177
178 }`
179             ], {type: 'text/plain'});
180         }
```

Figura 3.120 Método "crear_archivo ()" en "area-trabajo.component.ts" Parte 2

La segunda función importante que tiene el método "crear_archivo()" es la de crear la consulta hacia el repositorio de *firebase* e iniciar la carga con la *API* de *firebase*. En la figura 3.121 se observa la definición de la variable "uploadTask" que guarda la referencia de la consula a realizar en la dirección "/id-proyecto/nombre-archivo.java" dentro del repositorio (línea 184). Se inicia la carga y la *API* tiene callbacks (llamas asíncronas) para 3 casos diferentes, mientras carga (línea 189), al ocurrir un error (línea 200) o al terminar la carga (línea 216).

Una vez terminada la carga del archivo generado hacia el repositorio de *firebase*, se crea el registro del archivo en la base de datos. Esta es la tercera y última función importante del método "crear_archivo()".

```
183         //agregar el archivo con el método put() a la referencia
184         var uploadTask = this.referencia_almacen.child(this.proyecto.id + '/' + nombre_archivo + '.java')
185         .put(archivo);
186
187         uploadTask.on(firebase.storage.TaskEvent.STATE_CHANGED,
188
189         function(snapshot) {//mientras carga…
198         },
199
200         function(error){//al ocurrir un error…
214         },
215
216         function() {//al terminar la carga
```

Figura 3.121 Método "crear_archivo ()" en "area-trabajo.component.ts" Parte 3

Como se observa en la figura 3.122, dependiendo si el usuario desea crear el archivo como main o no, se le asiga un valor booleano en la propiedad "es_main" al objeto archivo (líneas 225 y 232). La ruta (link) del archivo se obtiene a través de la propiedad "uploadTask.snapshot.downloadURL".

```
216         function() {//al terminar la carga
217
218             let archivo: Archivo;
219
220             if(ref.estado_checkbox) {
221                 archivo = {
222                             id_archivo: "id",
223                             nombre_archivo: nombre_archivo + '.java',
224                             url_archivo: uploadTask.snapshot.downloadURL,
225                             es_main: true
226                           };
227             } else {
228                 archivo = {
229                             id_archivo: "id",
230                             nombre_archivo: nombre_archivo + '.java',
231                             url_archivo: uploadTask.snapshot.downloadURL,
232                             es_main: false
233                           };
234             }
```

Figura 3.122 Método "crear_archivo ()" en "area-trabajo.component.ts" Parte 4

En la figura 3.123 se observa cómo después de generar el objeto archivo a registrar, se crea la referencia de la consulta a realizar para insertar el objeto en la base de datos (línea 239). Una vez insertado el objeto con el método "push()" (línea 240), se actualiza el id que retorna firebase (líneas 243 a la 246).

```
239    const ref_archivo = ref.af.database.list('/archivos_proyecto_' + ref.proyecto.id);
240    ref_archivo.push(archivo).then((objeto) => {
241
242        //actualizar el registro para que tenga el id que se genera a través del push id de firebase
243        const ref_objeto = ref.af.database.object('/archivos_proyecto_'  +  ref.proyecto.id + '/' + objeto.key);
244        archivo.id_archivo = objeto.key;
245        ref.ds.id_archivo_seleccionado = objeto.key;
246        ref_objeto.update(archivo);
247
248    });
```

Figura 3.123 Método "crear_archivo ()" en "area-trabajo.component.ts" Parte 5

Como se observa en la figura 3.124, al concluir el registro del objeto archivo en la base de datos, se limpia el campo de texto para el nombre del archivo (línea 250). Además, el archivo generado se convierte en el nuevo archivo seleccionado (línea 251). Finalmente se deshabilita el checkbox para no permitir al usuario crear archivos main.

```
250            ref.valor_campo_archivo = "";
251            ref.seleccionar_archivo(archivo);
252
253            $('#checkboxCrearMain').prop('checked', false);
254            ref.estado_checkbox = false;
```

Figura 3.124 Método "crear_archivo ()" en "area-trabajo.component.ts" Parte 6

Método "descargar_contenido_archivo()": Este método como se observa en la figura 3.125, descarga el contenido del archivo seleccionado. Se utiliza el tipo *blob* para leer el contenido del archivo descargado (línea 297). Con el método "xhr.onload()" se defina el proceso de descarga del contenido (línea 298).

```
293        referenciaHttps.getDownloadURL().then(function(url) {
294
295            // descargar el archivo y lo guarda en BLOB
296            var xhr = new XMLHttpRequest();
297            xhr.responseType = 'blob';
298            xhr.onload = function(event) {
299
300                var blob = xhr.response;
```

Figura 3.125 Método "descargar_contenido_archivo ()" en "area-trabajo.component.ts" Parte 1

En la figura 3.126 se observa cómo se define el lector con la librería *Reader* para leer el contenido del archivo blob (línea 303). El contenido leído se lo pasa al componente editor para mostrar en la interfaz gráfica del editor de texto (línea 306). Se utiliza el método "readAsText()" de la librería *Reader* para comenzar la lectura (línea 308).

```
302            //leer el contenido del BLOB
303            var reader = new FileReader();
304            reader.onload = function(event){
305                //carga el contenido del archivo en el editor de texto
306                ref.auxServ.referencia_componente_editor.cargar_contenido(reader.result);
307            };
308            reader.readAsText(blob);
```

Figura 3.126 Método "descargar_contenido_archivo ()" en "area-trabajo.component.ts" Parte 2

Ya que se tiene definido el proceso de descarga, se llama a través del protocolo http get (figura 3.127) con el método "xhr.send" (línea 313).

```
312             xhr.open('GET', url);
313             xhr.send();
314
315         }).catch(function(error) {
316         //manejo de error
317         });
318     }
```

Figura 3.127 Método "descargar_contenido_archivo ()" en "area-trabajo.component.ts" Parte 3

Método "eliminar_archivo()": Como se observa en la figura 3.128, este método utiliza el método "remove()" de la *API* de *firebase* para eliminar el registro de la base de datos (línea 344) y el método "delete()" igual de la *API* de *firebase* para eliminar el archivo guardado en el repositorio (línea 347).

```
343     eliminar_archivo(archivo: Archivo) {
344         this.af.database.object('/archivos_proyecto_' + this.proyecto.id + '/' + archivo.id_archivo).remove();
345
346         let httpsReference = this.almacen.refFromURL(archivo.url_archivo);
347         httpsReference.delete().then(function() {
348             console.log('archivo eliminado');
349         }).catch(function(error) {
350             console.log('ocurrió un error');
351         });
```

Figura 3.128 Método "eliminar _archivo ()" en "area-trabajo.component.ts" Parte 1

En la figura 3.129 se puede observar cómo se establecen al estado por defecto banderas y variables. La variable "si_existe_main" se le asigna un valor booleano falso en el caso que el archivo eliminado haya sido un main (línea 354).

```
353         if(archivo.es_main) {
354             this.ds.si_existe_main = false;
355         }
356
357         //entra si el archivo seleccionado es el que se está eliminando
358         if(this.ds.id_archivo_seleccionado == archivo.id_archivo) {
359             //establece los estados adecuados
360             this.auxServ.referencia_componente_editor.nombre_archivo = "Archivo no seleccionado";
361             this.auxServ.referencia_componente_editor.editor.setValue("");
362             this.auxServ.referencia_componente_editor.es_necesario_guardar = false;
363             this.auxServ.referencia_componente_editor.si_archivo_main_seleccionado = false;
364             this.ds.activar_advertencia = false;
365         }
366     }
```

Figura 3.129 Método "eliminar _archivo ()" en "area-trabajo.component.ts" Parte 2

Método "convertir_archivo_en_main()": En la figura 3.130 se obseva la definición de este método. Si la aplicación detecta que el usuario guardó el contenido del archivo con el método main, y no existe un archivo main en el proyecto, ejecuta esta función. Se actualiza la propiedad "es_main" (líneas 380 y 381) y actualiza las variables relacionadas al control de la existencia de un archivo main en el proyecto. Finalmente, guarda el contenido del archivo editado (línea 384).

```
376     convertir_archivo_en_main() {
377         //actualizar
378         const ref_objeto = this.af.database.object('/archivos_proyecto_'
379         +  this.proyecto.id + '/' + this.archivo_seleccionado.id_archivo);
380         this.archivo_seleccionado.es_main = true;
381         ref_objeto.update(this.archivo_seleccionado);
382         this.auxServ.referencia_componente_editor.si_archivo_main_seleccionado = true;
383
384         this.referencia_topbar.guardar_contenito();
385     }
```

Figura 3.130 Método "convertir _archivo_en_main()" en "area-trabajo.component.ts" Parte 1

Método "crear_plantilla()": Este método lleva a cabo la creación de la plantilla a través del campo de texto del nombre y área de texto del contenido deseado a guardar. En la figura 3.131 se observa cómo el método verifica si no existen más de 10 plantillas. Después de la verificación, se realiza una consulta de registro hacia la base de datos para insertar la nueva plantilla (línea 400).

```
397     crear_plantilla() {
398         if(this.cantidad_plantillas_proyecto < 10) {
399
400             let consulta_plantillas = this.af.database.list('/plantillas/plantillas_proyecto_'
401             + this.ds.proyecto.id);
402             consulta_plantillas.push({
403                 nombre_plantilla: this.valor_campo_plantilla,
404                 contenido_plantilla: this.editor.getValue()
405             });
406         } else {
407             alert("No se pueden tener más de 10 plantillas en este proyecto.");
408         }
409     }
```

Figura 3.131 Método "crear _plantilla ()" en "area-trabajo.component.ts" Parte 1

3.5.1.2 Desarrollo Gráfico del Componente Área Trabajo

Este componente se encarga de algunos modals, área de proyecto y área de ejecución. En la figura 3.132 se observan ambas áreas.

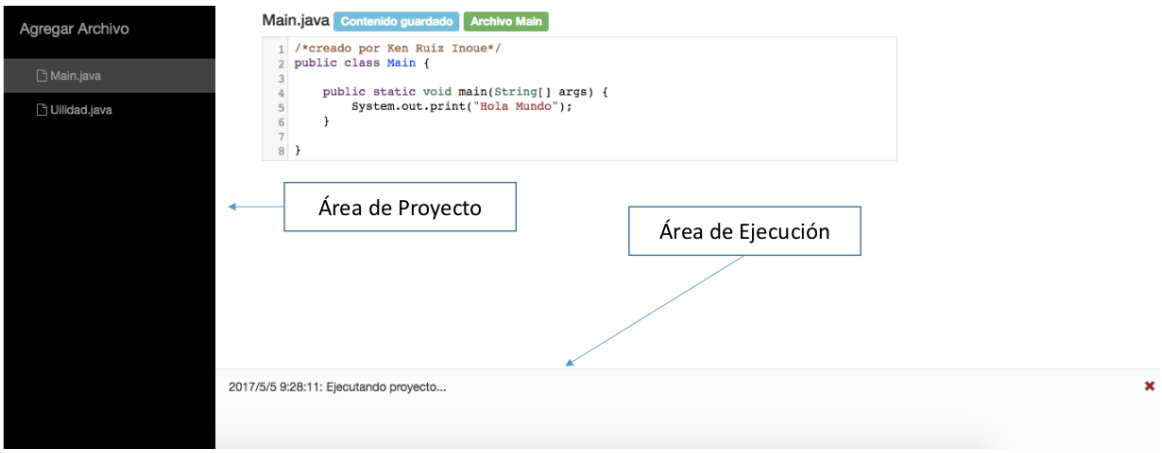

Figura 3.131 Área de Proyecto y Área de Ejecución en el Componente Área Trabajo

El componente área trabajo contiene en total 5 modals diferentes. A continuación, se detalla el desarrollo de cada modal.

Modal Crear Archivo: Este modal proporciona al usuario una interfaz gráfica para crear un nuevo archivo. En la figura 3.132 se muestra el contenido del cuerpo del modal.

El campo de texto (etiqueta input) se vincula con la variable "valor_campo_archivo" (línea 16) para así verificar con esa variable su validez con el método "validar_nombre_archivo()" (línea 16) cada vez que el usuario introduce una tecla.

Si llega a existir algún archivo main, se activa la alerta (línea 20). El checkbox se deshabilita al detectar un archivo main por el "[disabled]" de la línea 26.

```
12      <div class="modal-body">
13          <label for="entradaNombreArchivo">Nombre</label>
14          <div class="input-group">
15              <input placeholder="Nombre del archivo" type="text" id="valorNombreArchivo"
16              class="form-control" [(ngModel)]="valor_campo_archivo" (keyup)="validar_nombre_archivo()">
17              <span class="input-group-addon">.java</span>
18          </div>
19
20          <div class="alert alert-warning" role="alert" *ngIf="ds.si_existe_main" style="margin-top:20px;">
21              Ya existe un archivo main
22          </div>
23          <div class="checkbox">
24              <label><input id="checkboxCrearMain" type="checkbox"
25              (click)="cambiar_estado_checkbox($event.target.checked)"
26              [disabled]="ds.si_existe_main">Crear como main</label>
27          </div>
28      </div>
```

Figura 3.132 Definición del Modal Crear Archivo Parte 1

En la figura 3.133 se observa la definición de las alertas restantes y el pie del modal.

Si la variable "campo_archivo_aprobado" contiene un falso (línea), muestra una alerta para notificarle al usuario que el campo de texto no está aprobado por la aplicación. Si el campo de texto contiene el nombre de otro archivo existente, entonces muestra otra alerta para hacerle saber al usuario que no se permiten archivos con el mismo nombre (líneas 35 a la 38).

El pie del modal contiene 2 botones, uno para cancelar la acción y otro para confirmarla. El botón "Cancelar" llama el método "limpiar_campo_archivo()" para establecer valores iniciales en las variables de este modal. El botón "Aceptar" se habilita si el campo de texto está aprobado por la variable "campo_archivo_aprobado" y verificada la unicidad del nombre del archivo con la variable "nombre_archivo_unico" (línea 44). Al presionar el botón "Aceptar", se ejecuta el método "crear_archivo".

```
30      <div style="margin:15px" *ngIf="!campo_archivo_aprobado" role="alert" class="alert alert-danger">
31          <span class="glyphicon glyphicon-remove-circle" aria-hidden="true"></span>
32          El archivo debe de contener únicamente letras o numeros sin espacios
33      </div>
34
35      <div style="margin:15px" *ngIf="!nombre_archivo_unico" class="alert alert-danger">
36          <span class="glyphicon glyphicon-remove-circle" aria-hidden="true"></span>
37          Ya existe un archivo con el mismo nombre
38      </div>
39
40      <div class="modal-footer">
41          <button type="button" class="btn btn-default" data-dismiss="modal"
42          (click)="limpiar_campo_archivo()">Cancelar</button>
43          <button type="button" class="btn btn-primary" (click)="crear_archivo()"
44          [disabled]="!campo_archivo_aprobado || !nombre_archivo_unico"
45          data-dismiss="modal">Aceptar</button>
46      </div>
```

Figura 3.133 Definición del Modal Crear Archivo Parte 2

El modal para crear archivos tiene la apariencia como se muestra en la figura 3.134. En este caso únicamente está activada la alerta del campo de texto.

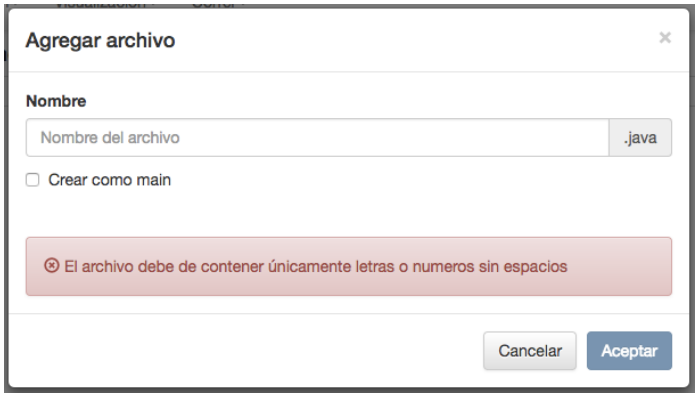

Figura 3.134 Apariencia del Modal Crear Archivo

En la figura 3.135 se muestra el modal para crear archivos con la alerta que notifica al usuario la existencia de un archivo y la alerta que notifica la detección de un nombre de archivo existente. En este caso el checkbox se encuentra deshabilitado, al colocar el cursor encima de él, se bloquea.

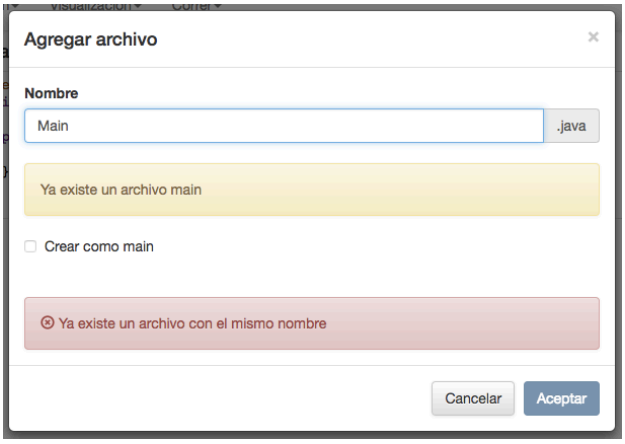

Figura 3.135 Apariencia del Modal Crear Archivo con las Alertas Restantes

Modal Advertencia: Este modal aparece en el caso que el usuario quiera seleccionar otro archivo sin guardar el contenido modificado. Proporciona al usuario una interfaz gráfica para que tome una decisión entre descartar el contenido modificado y cargar el contenido del archivo seleccionado o cancelar la acción de seleccionar otro archivo. En la figura 3.136 se muestra la parte importante de la definición del modal.

El botón para cancelar la acción cierra el modal. El botón para descartar cambios llama a la función "comunicar_con_editor()" (línea 68). Esta función como se había mencionado, carga el contenido del archivo en el editor de texto.

```
62      <p>Si no se guardaron los cambios, se perderá el contenido editado.</p>
63
64  </div>
65  <div class="modal-footer">
66      <button type="button" class="btn btn-default" data-dismiss="modal">Cancelar</button>
67      <button type="button" class="btn btn-primary" data-dismiss="modal"
68      (click)="comunicar_con_editor()">Descartar cambios</button>
```

Figura 3.136 Definición del Modal Advertencia

En la figura 3.137 se muestra la apariencia del modal que muestra la advertencia. Al hacer clic afuera de este modal o presionar el botón de cancelar, se cierra el modal.

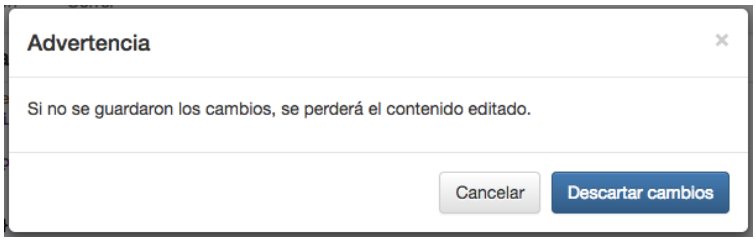

Figura 3.137 Apariencia del Modal Advertencia

Modal Convertir Main: Este modal aparece cuando el usuario trata de guardar el contenido del archivo y la aplicación detecta un método main dentro de ese contenido (siempre y cuando no exista un archivo main) para proporcionarle al usuario la opción de convertir el archivo a ser guardado en main. En la figura 3.138 se muestra la defición del modal.

Al presionar el botón para confirmar la acción, llama el método "convertir_archivo_en_main()" (línea 88) para actualizar la propiedad "es_main" del archivo en la base de datos.

```
82  <div class="modal-body">
83    <p>Se detecto un método Main, ¿quiere convertir este archivo en Main?</p>
84  </div>
85  <div class="modal-footer">
86    <button type="button" class="btn btn-default" data-dismiss="modal">Cancelar</button>
87    <button type="button" class="btn btn-primary" data-dismiss="modal"
88    (click)="convertir_archivo_en_main()">Convertir en Main</button>
```

Figura 3.138 Definición del Modal Convertir Main

En la figura 3.139 se muestra la apariencia del modal convertir main.

Figura 3.139 Apariencia del Modal Convertir Main

Modal Crear Plantilla Código: Este modal permite al usuario crear plantillas de código. En la figura 3.140 se observa la primera parte de la definición del modal.

Si se cierra el modal con el botón de cierre (tache), se llama el método "limpiar_campos_plantilla()" para vaciar los campos de introducción (línea 100).

El campo de introducción para el nombre de la plantilla se conecta con la variable "valor_campo_nombre_plantilla" (línea 109). Cada vez que se introduce un carácter en el campo, se llama el método "validar_nombre_plantilla()". Este método valida que el nombre de la plantilla contenga únicamente caracteres alfanuméricos sin espacios.

Se muestra un mensaje de alerta para notificar al usuario que únicamente se aceptan caracteres alfanuméricos en el campo de nombre. En el caso que la variable "campo_plantilla_aprobado" contenga un valor booleano verdadero, desaparece el mensaje de alerta.

```
97  <div class="modal-content">
98      <div class="modal-header">
99          <button type="button" class="close" data-dismiss="modal" aria-label="Close">
100             <span aria-hidden="true" (click)="limpiar_campos_plantilla()">&times;</span>
101         </button>
102         <h4 class="modal-title" id="myModalLabel">Agregar plantilla</h4>
103     </div>
104
105     <div class="modal-body">
106         <div style="margin:15px">
107             <label for="entradaNombreArchivo">Nombre</label>
108             <input placeholder="Nombre de la plantilla" type="text" id="valorNombrePlantilla"
109             class="form-control" [(ngModel)]="valor_campo_nombre_plantilla" (keyup)="validar_nombre_plantilla()"
110         </div>
111
112         <div style="margin:15px" *ngIf="!campo_plantilla_aprobado" role="alert" class="alert alert-danger">
113             <span class="glyphicon glyphicon-remove-circle" aria-hidden="true"></span>
114                 El nombre de la plantilla debe de contener únicamente letras o numeros sin espacios
115         </div>
```

Figura 3.140 Definición del Modal Crear Plantilla Código Parte 1

En la figura 3.141 se observa el resto de la definición del modal crear plantilla código. El campo de introducción para el contenido de la plantilla, se vincula con la variable "valor_campo_contenido_plantilla" con la directiva "ngModel" (línea 119).

Al presionar el botón cancelar se llama el método "limpiar_campos_plantilla()" (línea 125). El botón para aceptar llama el método "crear_plantilla()", este método utiliza el contenido de las variables "valor_campo_nombre_plantilla" y "valor_campo_contenido_plantilla" para crear un nuevo registro en la base de datos. En el caso que la variable

"campo_plantilla_aprobado" contenga un valor booleano falso, deshabilita el botón para aceptar.

```
117         <div style="margin:15px">
118             <label for="entradaNombreProyecto">Contenido</label>
119             <textarea class="form-control" rows="8" [(ngModel)]="valor_campo_contenido_plantilla"></textarea>
120         </div>
121     </div>
122
123     <div class="modal-footer">
124         <button type="button" class="btn btn-default" data-dismiss="modal"
125         (click)="limpiar_campos_plantilla()">Cancelar</button>
126         <button type="button" class="btn btn-primary" data-dismiss="modal"
127         (click)="crear_plantilla()" [disabled]="!campo_plantilla_aprobado">Aceptar</button>
128     </div>
```

Figura 3.141 Definición del Modal Crear Plantilla Código Parte 2

En la figura 3.142 se puede observar la apariencia del modal crear plantilla código.

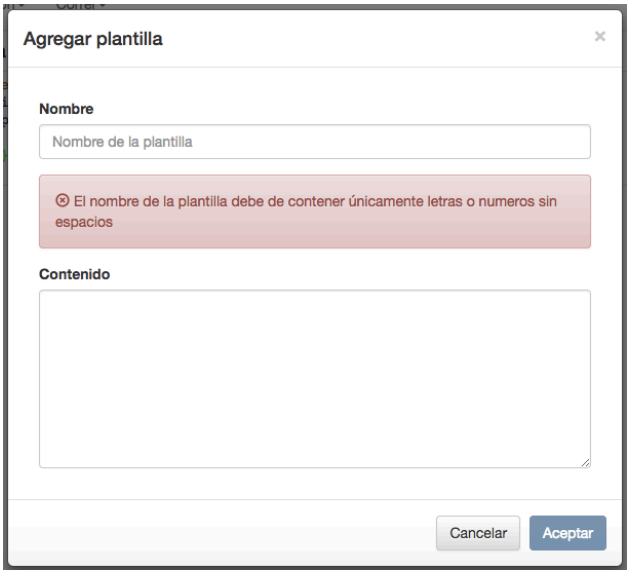

Figura 3.142 Apariencia del Modal Convertir Main

Modal Confirmación Eliminar Plantilla: Este modal confirma la decisión del usuario para eliminar la plantilla seleccionada. En la figura 3.143 se observa la definición del modal.

Al presionar el botón para cancelar cierra el modal (línea 148). El botón para eliminar "eliminar_plantilla" elimina el registro de la plantilla selecciona desde la base de datos (línea 150).

```
142     <div class="modal-body">
143         <div style="margin-bottom:15px" class="alert alert-danger">
144             Se borrará la plantilla de manera permanente, ¿está seguro?
145         </div>
146     </div>
147     <div class="modal-footer">
148       <button type="button" class="btn btn-default" data-dismiss="modal">Cancelar</button>
149       <button type="button" class="btn btn-danger" data-dismiss="modal"
150       (click)="eliminar_plantilla()">Eliminar</button>
```

Figura 3.143 Definición del Modal Confirmación Eliminar Plantilla

En la figura 3.144 se puede observar la apariencia del modal confirmación eliminar plantilla.

Figura 3.144 Apariencia del Modal Confirmación Eliminar Plantilla

A continuación se detalla el desarrollo del área de proyecto. Como se observa en la figura 3.145, se declara una etiqueta para la barra lateral (línea 171). Adentro de la barra lateral, se coloca el botón para agregar archivos nuevos (líneas 174 a la 177). Este botón llama el método "abrir_modal_crear_arhico()" para que el usuario genere archivos (línea 175).

```
169  <div id="wrapper">
170      <!-- Sidebar -->
171      <div id="sidebar-wrapper" style="margin-top:40px;">
172          <ul class="sidebar-nav">
173              <li class="sidebar-brand">
174                  <a class="accordion-toggle collapsed toggle-switch" href="javascript:void(0)"
175                  (click)="abrir_modal_crear_archivo()">
176                      Agregar Archivo
177                  </a>
178              </li>
```

Figura 3.145 Definición del Área de Proyecto Parte 1

En la figura 3.146 se muestra la definición de la representación de los archivos pertenecientes al proyecto.

Se empleó la directiva "ngFor" para generar botones de los archivos a través del arreglo "archivos_proyecto" (línea 180). Dependiendo si el archivo está seleccionado o no, genera el botón con un estilo diferente y sin método para representar el estado de selección. El estado de selección lo determina con la variable "id_archivo_seleccionado" (líneas 182 y 188). El botón que representa el estado de no estar seleccionado llama el método "seleccionar_archivo()" al ser presionado, para realizar la carga del contenido del archivo en el editor de texto (línea 181). Se utilizaron las directivas de "contextMenu" para habilitar el menú personalizado al hacer clic derecho (líneas 183 y 189). Obtiene el nombre del archivo a través de la propiedad "nombre_archivo" (líneas 185 y 191).

```
180     <li *ngFor="let archivo of ds.archivos_proyecto">
181         <a href="javascript:void(0)" (click)="seleccionar_archivo(archivo)"
182         *ngIf="archivo.id_archivo != ds.id_archivo_seleccionado"
183         [contextMenu]="basicMenu" [contextMenuSubject]="archivo">
184             <span class="sidebar-icon"><i class="fa fa-file-o"></i></span>
185             <span class="sidebar-title">{{archivo.nombre_archivo}}</span>
186         </a>
187         <a href="javascript:void(0)" (click)="seleccionar_archivo(archivo)"
188         style="background-color:■#424242;" *ngIf="archivo.id_archivo == ds.id_archivo_seleccionado"
189         [contextMenu]="basicMenu" [contextMenuSubject]="archivo">
190             <span class="sidebar-icon"><i class="fa fa-file-o"></i></span>
191             <span class="sidebar-title">{{archivo.nombre_archivo}}</span>
192         </a>
193     </li>
```

Figura 3.146 Definición del Área de Proyecto Parte 2

El menú personalizado se generó con el objetivo de proporcionar al usuario la opción de eliminar el archivo desde el área de proyecto con el clic derecho. En la figura 3.147 se observa la definición del menú personalizado.

La etiqueta "context-menu" define el contenido del menú y el método a llamar al ser presionado. En esta ocasión se definió un solo elemento para eliminar archivos (líneas 198 a la 200). Esta opción llama el método "eliminar_archivo()" para remover el registro del archivo desde la base de datos.

```
197     <context-menu>
198         <ng-template contextMenuItem (execute)="eliminar_archivo($event.item)">
199             Eliminar archivo
200         </ng-template>
201     </context-menu>
```

Figura 3.147 Definición del Área de Proyecto Parte 3

En la figura 3.148 se muestra la apariencia del área de proyecto. Se observan 2 archivos, uno seleccionado (Main.java) y otro no seleccionado. Con el clic derecho aparece la opción para eliminar el archivo como se muestra en la figura.XXXXXXXX

Figura 3.148 Apariencia del Área de Proyecto

A continuación se muestra la definición del contenido principal (figura 3.149). Como se puede observar, se utiliza el selector de "app-editor" para llamar el componente editor adentro de la etiqueta "main" (líneas 210 a la 212). Se declara una variable local "#editor" para tener la referencia del componente editor dentro del componente área de trabajo (línea 211). Eso implica la capacidad de hacer llamadas de métodos del componente editor de texto desde el componente área de trabajo.

```
205    <!-- contenido principal -->
206    <div id="page-content-wrapper" style="margin-top:10px;">
207        <div class="container-fluid">
208            <div class="row">
209                <div class="col-lg-12">
210                    <main id="page-content-wrapper" role="main">
211                        <app-editor #editor></app-editor>
212                    </main>
```

Figura 3.149 Definición del Área Principal

Para concluir con la explicación del componente de área de trabajo, se detalla la primera parte de la definición del área de ejecución como se muestra en la figura 3.150. Esta área se encarga de mostrar los mensjaes producidos por el servidor.

A través de la directiva "ngFor" carga todos los mensajes guardados en la variable "mensajes_resultado_registro" para mostrarlos en el área de ejecución (línea 232). Dependiendo si la variable "si_esconder_area_proyecto" contiene un verdado o falso booleano, se le agrega un margen hacia la derecha en el estilo para recorrer esta área (líneas 231 y 238).

```
230    <!-- carga los mensajes al DOM -->
231    <div  *ngIf="!ds.si_esconder_area_proyecto" style="margin-left:270px;">
232        <div class="row" *ngFor="let mensaje of ds.mensajes_resultado_registro">
233            {{mensaje}}
234        </div>
235    </div>
236
237    <!-- carga los mensajes al DOM -->
238    <div  *ngIf="ds.si_esconder_area_proyecto">
239        <div class="row" *ngFor="let mensaje of ds.mensajes_resultado_registro">
240            {{mensaje}}
241        </div>
242    </div>
```

Figura 3.150 Definición del Área de Ejecución Parte 1

En la figura 3.151 se muestra el resto de la defición del área de ejecución. En este bloque de código se declara la animación de carga (línea 246) y el botón con símbolo de tache para esconder el área de ejecución (líneas 249 a la 251). Al hacer clic al botón, llama el método "cambiar_estado_area_ejecucion()" para esconder el área de ejecución.

```
246    <div class="loader" *ngIf="ds.mostrar_loader"></div>
247
248    <div class="col-md-1 col-xs-1">
249        <a href="javascript:void(0)" (click)="cambiar_estado_area_ejecucion()"  class="pull-right">
250            <span id="icono-remover-area-compilacion" class="glyphicon glyphicon-remove"></span>
251        </a>
252    </div>
```

Figura 3.151 Definición del Área de Ejecución Parte 2

En la figura 3.152 se muestra la apariencia del área de ejecución y la animación de carga.

Figura 3.152 Apariencia del Área de Ejecución y la Animación de Carga

Con esto concluye la explicación del componente área de trabajo.

3.5.2 Desarrollo del Componente Editor

Este componente se encarga de cargar, crear, actualizar y eliminar archivos dentro del proyecto y del área de multifunciones. Esta área contiene una pestaña para enviar y recibir mensajes entre los participantes del proyecto y otra para crear plantillas a insertar en el editor de texto. Recibe instrucciones como guardar o eliminar archivos desde el componente topbar a través del componente área trabajo. También revisa el contenido de los archivos a guardar para no permitir ciertos contenidos (declaración de la clase Thread, y Scanner entre otros), esto debido a las limitaciones establecidas para el servidor en el capítulo 2.

3.5.2.1 Desarrollo Lógico del Componente Editor

Este componente utiliza 23 métodos para cumplir con sus funcionalidades descritas. En la tabla 3.8 se muestran y detallan los métodos empleados.

Tabla 3.8 Métodos del Componente Editor

Método	Funcionalidad
ngOnInit()	Inicializa variables y obtiene datos del proyecto seleccionado y sus archivos correspondientes.
ngAfterViewInit()	Declara los eventos de los modals y crea una instnacia del editor de texto con la librería de *CodeMirror*.
comprobar_cambio()	Comprueba si es necesario guardar el archivo seleccionado para mostrar una etiqueta y mensaje de advertencia.
cargar_contenido()	Carga en el archivo su contenido descargado en el editor de texto.
obtener_archivo()	Guarda en 3 variables las propiedades del archivo seleccionado.
deshacer()	Carga el estado anterior guardado en el historial del editor de texto.
rehacer()	Carga el estado posterior guardado en el historial del editor de texto.
poner_nombre()	Le asigna el nombre introducido en el campo de texto a la propiedad del archivo.
cambiar_estado_area_multifuncional()	Esconde o muestra el área multifuncional cambiando el estado de la variable booleana “esconder_area_multifuncional”.
intento_guardar_contenido()	Verifica que todas las variables booleanas que guardan el estado del editor de texto, se encuentren en un estado ideal para llamar el método “guardar_contenido()”.
guardar_contenido()	Guarda el contenido del editor de texto en un archivo ubicado en el repositorio de *firebase*.

enviar_mensaje()	Se genera una consulta que apunte la dirección de mensajes del proyecto para crear un registro en la base de datos de *firebase*.
contiene_main()	Retorna un valor booleano verdadero si en la cadana de texto que recibe como parámetro encuentra el patrón “public static void main”.
contiene_scanner()	Retorna un valor booleano verdadero si en la cadana de texto que recibe como parámetro encuentra el patrón “Scanner”.
contiene_buffered_reader()	Retorna un valor booleano verdadero si en la cadana de texto que recibe como parámetro encuentra el patrón “BufferedReader”.
contiene_timer()	Retorna un valor booleano verdadero si en la cadana de texto que recibe como parámetro encuentra el patrón “Timer”.
contiene_thread()	Retorna un valor booleano verdadero si en la cadana de texto que recibe como parámetro encuentra el patrón “Thread”.
contiene_public_class_nombre()	Retorna un valor booleano verdadero si en la cadana de texto que recibe como parámetro encuentra el patrón “class” seguido por el nombre de la clase (archivo sin la extensión *java*).
abrir_modal_crear_plantilla()	Abre el modal para crear plantillas.
seleccionar_plantilla_siguiente()	Actualiza la plantilla seleccionada en el servicio de datos por el siguiente objeto guardado en el arreglo de platillas.
seleccionar_plantilla_anterior()	Actualiza la plantilla seleccionada en el servicio de datos por el anterior objeto guardado en el arreglo de platillas.
abrir_modal_eliminar_plantilla()	Abre el modal para eliminar plantillas.
insertar_plantilla	Agrega un nuevo registro de la plantilla en la base de datos de *firebase* a través de los valores introducidos en los campos de texto.

Método “ngOnInit()”: Este método es el primero en llamarse. Este método suscribe 2 consultas para observar el cambio de los mensajes y plantillas del proyecto.

Como se observa en la figura 3.153 se genera una consula hacia la dirección de los mensajes del proyecto (línea 62). Se suscribe la consulta y cada vez que el registro que apunta la consulte sufre una modificación, se llama el bloque de código de la suscripción e inicializa el arreglo “mensajes_proyecto” (líneas 65 y 68).

```
61      //obtener los mensajes de manera continua con una suscripción
62      let consulta_mensajes = this.af.database.list('/mensajes/mensajes_proyecto_' + this.ds.proyecto.id);
63
64      //cada vez que exista un nuevo mensaje, se dispara esta parte del código
65      consulta_mensajes.subscribe(mensajes => {
66
67          //inicializar arreglo para volver a insertar los mensajes
68          this.mensajes_proyecto = [];
```

Figura 3.153 Método "ngOnInit()" en "editor.component.ts" Parte 1

En la figura 3.154 se observa el bloque de código de la suscripción de los mensajes. Se recorren todos los mensajes del arreglo "mensajes" y de cada mensaje se obtene el nombre del usuario a través de la llave foránea que es la propiedad "propietario_mensaje" con la consulta "consulta_usuario" (líneas 70 y 73). La consulta generada en la línea 73 se utiliza una única vez para evitar crear múltiples suscripciones con el método "first()" (línea 74).

Una vez que se obtenga el nombre del usuario, se agrega al arreglo "mensajes_proyecto" el nombre del usuario, el contenido del mensaje y el url de la foto de perfil (líneas 77 a la 81).

```
70          for(let mensaje of mensajes) {
71
72              //de la llave foranea se obtienen los datos del propietario del mensaje
73              let consulta_usuario = this.af.database.object('/users/' + mensaje.propietario_mensaje);
74              consulta_usuario.first().subscribe(usuario => {
75
76                  //se insertan los datos del mensaje en el arreglo
77                  ref.mensajes_proyecto.push({
78                      contenido_mensaje: mensaje.contenido_mensaje,
79                      propietario_mensaje: usuario.nombre,
80                      url_img_mensaje:  usuario.imgurl
81                  });
82              });
83          }
```

Figura 3.154 Método "ngOnInit()" en "editor.component.ts" Parte 2

Después de suscribir la consulta para mensajes, se genera y suscribe la consulta para las plantillas (figura 3.155). Al entrar al bloque de código de la suscripción, verifica que existan plantillas (línea 92). Se inicializan las variables "plantillas_proyecto" y "contador_plantillas" para guardar el contenido actualizado (líneas 94 y 95). Se guardan todas las plantillas retornadas en el arreglo "plantillas_proyecto".

```
87      //obtener las plantillas de código de manera continua con una suscripción
88      let consulta_plantillas = this.af.database.list('/plantillas/plantillas_proyecto_' + this.ds.proyecto.id);
89
90      consulta_plantillas.subscribe(plantillas => {
91
92          if(plantillas.length != 0) {
93              //inicializar arreglo para volver a insertar los mensajes
94              this.plantillas_proyecto = [];
95              this.contador_plantillas = 0;
96
97              for(let plantilla of plantillas) {
98                  this.plantillas_proyecto.push(plantilla);
99              }
```

Figura 3.155 Método "ngOnInit()" en "editor.component.ts" Parte 3

En la fiugra 3.156 se muestra la última parte de la definición del método “ngOnInit()”. Se le asigna la cantidad de plantillas en la variable “cantidad_plantilas” y la cantidad límite a la que puede alcanzar el recorrido de plantillas en la variable “limite_contador” (líneas 101 y 102). Al actualizar las plantillas se guarda en la variable “plantilla_seleccionada” la primer plantilla guardada en el arreglo “plantillas_proyecto” (línea 104). También se actualiza el contenido de la variable “area_texto_contenido_plantilla” para mostrarlo en la interfaz gráfica (línea 105). Finalmente, si no existe ninguna plantilla en el arreglo retornado, se le asigna un cero en la variable “cantidad_plantillas” y una cadena vacía en la variable “area_texto_contenido_plantilla” (líneas 108 y 109).

```
101             this.cantidad_plantillas = this.plantillas_proyecto.length;
102             this.limite_contador = this.plantillas_proyecto.length - 1;
103
104             this.ds.plantilla_seleccionada = this.plantillas_proyecto[this.contador_plantillas];
105             this.area_texto_contenido_plantilla = this.ds.plantilla_seleccionada.contenido_plantilla;
106         } else {
107             //No hay plantillas o se eliminó la última plantilla
108             this.cantidad_plantillas = 0;
109             this.area_texto_contenido_plantilla = "";
110         }
```

Figura 3.156 Método “ngOnInit()” en “editor.component.ts” Parte 4

Método “ngAfterViewInit()”: Este método se llama después de haber concluido el método “ngOnInit()” para configurar el área de texto con la librería de *codemirror*.

En la figura 3.157 se observa la primera parte de la definición del método. La configuración del área de texto se realiza en este método para obtener la referencia del área de texto “editorTexto” (línea 120) ya una vez concluida la construcción de los elementos html. Se genera la configuración al crear la instancia del objeto “CodeMirror” (líneas 121 a la 125). Después registra un *listener* para monitorizar el cambio del contenido del editor de texto (líneas 127 a la 129).

```
120     this.editor = CodeMirror.fromTextArea(document.getElementById("editorTexto"), {
121         lineNumbers: true,
122         mode: "text/x-java",
123         styleActiveLine: true,
124         matchBrackets: true
125     });
126
127     this.editor.on('change', editor => {//se ejecuta esta parte de código cada vez que hay una modificación
128         this.comprobar_cambio();
129     });
```

Figura 3.157 Método “ngAfterViewInit()” en “editor.component.ts” Parte 1

En la segunda parte del método se definen métodos para utilizar la librería “Clipboard” con el fin de permitir al usuario guardar el contenido del editor de texto en el porta papeles a través de un botón (figura 3.158).

```
131     var btn = document.getElementById('btn');
132     var clipboard = new Clipboard(btn, {
133         text: function(trigger) {
134             return ref.editor.getValue();
135         }
136     });
137
138     clipboard.on('success', function(e) {
139         console.log(e);
140         alert("Contenido copiado en el portapapeles");
141     });
142
143     clipboard.on('error', function(e) {
144         console.log(e);
145     });
```

Figura 3.158 Método "ngAfterViewInit()" en "editor.component.ts" Parte 2

Método "intento_guardar_contenido()": Este método se encarga en verificar si el contenido del editor de texto es apto a ser guardado sin afectar la funcionalidad del servidor. En total tiene 10 casos diferentes.

En la figura 3.159 se observa la primera parte de la definición del método y muestra 4 casos diferentes.

Caso 1: El primer caso se haya si el usuario trata de guardar el contenido del editor de texto cuando la aplicación está comunicándose con el servidor para compilar o ejecutar el proyecto. Activa la alerta para notificar al usaurio lo sucedido (líneas 198 y 199).

Caso 2: Cuando el archivo no contiene la sintaxis "public class nombre_archivo", se le notifica al usuario a través de una alerta. Por ejemplo, si el archivo se llama "Humano.java", entonces el editor de texto debe de tener como contenido "public class Humano" para no activar esta alerta (líneas 200 y 201).

Caso 3: Si el archivo a guardar contiene el método "public static void main(String [] args)" (método main) y ya existe otro archivo que lo contiene, no permite guardar el contenido y se le notifica al usaurio (líneas 203 y 204).

Caso 4: Al contrario al caso anterior, si el archivo a guardar contiene el método main pero no existe un archivo que lo contenga, llama el modal para convertir el archivo en main (líneas 205 a la 208).

```
196 intento_guardar_contenido(){
197
198     if(this.ds.comunicandose_con_el_servidor == true) {
199         alert("Estableciendo comunicación con el servidor, espere un momento.");
200     } else if(!this.contiene_public_class_nombre(this.editor.getValue())) {
201         alert("Debe de contener la defición de clase y el mismo nombre del archivo.");
202     } else if(this.contiene_main(this.editor.getValue())
203     && !this.archivo_seleccionado.es_main && this.ds.si_existe_main){
204         alert("Este archivo contiene el método main, ya existe un archivo con método main en este proyecto.");
205     } else if(this.contiene_main(this.editor.getValue())
206     && !this.archivo_seleccionado.es_main && !this.ds.si_existe_main){
207         //activar modal para convertir en main
208         $("#modalConvertirMain").modal("show");
```

Figura 3.159 Método "intento_guardar_contenido()" en "editor.component.ts" Parte 1

En la figura 3.160 se obervan los 6 casos restantes que maneja el método.

Caso 5: Si se trata de guardar un archivo que tiene la propiedad "es_main = true" sin el contenido del método main, se le notifica al usaurio.

Caso 6-9: Existen ciertas clases de java que no se permiten declarar para permitir que el servidor funcione de la manera apropiada. De las líneas 211 a 218 se definen los casos para notificar al usuario lo sucedido. No se permite el uso de la clase "Scanner", "BufferedReader", "Timer" y "Thread".

Caso 10: Si no se cumple alguna condición de los casos anteriores, entonces la aplicación procede a llamar el método "guardar_contenido()" para cargar el contenido del editor de texto hacia la nube.

```
209     } else if(!this.contiene_main(this.editor.getValue()) && this.archivo_seleccionado.es_main){
210         alert("Este archivo debe de contener el método main, no se puede guardar.");
211     } else if(this.contiene_scanner(this.editor.getValue())) {
212         alert("Este archivo contiene un Scanner, favor de removerlo.");
213     } else if(this.contiene_buffered_reader(this.editor.getValue())) {
214         alert("Este archivo contiene un BufferedReader, favor de removerlo.");
215     } else if(this.contiene_timer(this.editor.getValue())) {
216         alert("Este archivo contiene un Timer, favor de removerlo.");
217     } else if(this.contiene_thread(this.editor.getValue())) {
218         alert("Este archivo contiene un Thread, favor de removerlo.");
219     } else {
220         this.guardar_contenido();
221     }
```

Figura 3.160 Método "intento_guardar_contenido()" en "editor.component.ts" Parte 2

Método "guardar_contenido()": Este método define y establece una comunicación con el repositorio de *firebase* para iniciar la carga del archivo a guardar. Una vez terminada la carga, ejecuta un bloque de código para establecer algunas variables auxiliares a su valor original.

En la figura 3.161 se muestra la primera parte de la definición del método "guardar_contenido". Se crea un archivo a través del contenido del editor de texto (línea 234). Después se define en una variable (variable de tarea) la ruta de almacenamiento en el repositorio y el archivo a cargar (línea 236). Una vez declarada la variable de tarea, se inicia la carga con el método "on()" que ofrece la *API* de *firebase* (línea 239).

```
233         //obtener el contenido del editor y guardarlo en un archivo blob
234         var archivo = new Blob([this.editor.getValue()], {type: 'text/plain'});
235         //agregar el archivo con el método put() a la referencia
236         var uploadTask = this.almacenamiento.refFromURL(this.archivo_seleccionado.url_archivo).put(archivo);
237
238         //comenzar la carga
239         uploadTask.on(firebase.storage.TaskEvent.STATE_CHANGED,
240         function(snapshot) {//mientras carga
241             switch (snapshot.state) {
242                 case firebase.storage.TaskState.PAUSED: // pausado
243                 console.log('carga pausada');
244                 break;
245                 case firebase.storage.TaskState.RUNNING: // subiendo
246                 console.log('cargando archivo generado');
247                 break;
248             }
249         },
```

Figura 3.161 Método "guardar_contenido()" en "editor.component.ts" Parte 1

En la figura 3.162 se muestra el resto de la definición del método. Una vez concluida la carga, se ejecuta el bloque de código de las líneas 266 a 274. Se actualiza la variable

"contenido_nube" para mostrar la etiqueta con el contenido "Contenido guardado" (línea 271). Se le asigna su valor original a la variable "es_necesario_guardar" y "activar_advertencia".

```
265         function() {//al terminar la carga
266             console.log("carga completa");
267             console.log("url:" + uploadTask.snapshot.downloadURL);
268             /* se actualiza el contenido de la nuba al terminar la carga
269             ** para mostrar la etiqueta de estado de una manera apropiada
270             */
271             ref.contenido_nube = ref.editor.getValue();
272             ref.es_necesario_guardar = false;
273             ref.ds.activar_advertencia = false;
274         });
```

Figura 3.162 Método "guardar_contenido()" en "editor.component.ts" Parte 2

Método "enviar_mensaje()": Como se observa en la figura 3.163, este método genera un registro en la base de datos de *firebase* en la ruta "/mensajes/mensajes_proyecto_{id-proyecot}" (línea 279). Obtiene la cadena introducida en el campo de texto y el id del usuario para generar el registro (líneas 281 a 283). Una vez concluido el registro, se vacía el campo de texto (línea 284).

```
278     enviar_mensaje() {
279         let consulta_mensaje = this.af.database.list('/mensajes/mensajes_proyecto_' + this.ds.proyecto.id);
280         consulta_mensaje.push({
281             contenido_mensaje: this.contenido_mensaje,
282             propietario_mensaje: this.ds.usuario.id
283         });
284         this.contenido_mensaje = null;
285     }
```

Figura 3.163 Método "enviar_mensaje()" en "editor.component.ts"

Método "contiene_main()": Este método comprueba que el editor de texto contenga el patrón del método main. En la figura 3.164 se observa la definición del método.

Todo el contenido del editor de texto se guarda en el arreglo "palabras" (línea 288). A través de un ciclo for, se busca el patrón "public static void main" para corroborar la existencia del método (líneas 289 a 299). Si cumple con el patrón, el método retorna un valor booleano verdadero (línea 294).

```
287     contiene_main(contenido) {
288         var palabas = contenido.split(/\s+/);
289         for(let i = 0; i < palabas.length; i++) {
290             if(palabas[i] == 'public'){
291                 if(palabas[i + 1] == 'static') {
292                     if(palabas[i + 2] == 'void') {
293                         if(palabas[i + 3].includes("main")) {
294                             return true;
295                         }
296                     }
297                 }
298             }
299         }
300     }
```

Figura 3.164 Método "contiene_main()" en "editor.component.ts"

Método "seleccionar_plantilla()": Este método se activa cuando se hace clic al botón siguiente de las plantillas. En la figura 3.165 se muestra la definición del método. Se aumenta el contador de plantilla para apuntar la siguiente plantilla del arreglo "plantillas_proyecto" (línea 367).

Se actualiza la plantilla seleccionada en el servicio datos y también el área de texto con su correspondiente contenido de la plantilla para ser insertada en el editor de texto.

El método "seleccionar_plantilla_anterior" tiene este mismo enfoque, pero en vez de aumentar el contador, lo disminuye para apuntar la plantilla anterior.

```
366     seleccionar_plantilla_siguiente() {
367         this.contador_plantillas ++;
368         this.ds.plantilla_seleccionada = this.plantillas_proyecto[this.contador_plantillas];
369         this.area_texto_contenido_plantilla = this.ds.plantilla_seleccionada.contenido_plantilla;
370     }
```

Figura 3.165 Método "seleccionar_plantilla_siguiente()" en "editor.component.ts"

Con esto se concluye el desarrollo lógico del componente editor.

3.5.2.2 Desarrollo Gráfico del Componente Editor

A continuación, se detalla el desarrollo de la interfaz gráfica del usuario del componente editor. Este componente se encarga en dibujar el editor de texto y el área multifuncional de la aplicación.

En la figura 3.166 se observa la definición del editor de texto. Como se observa, este editor es una instancia del elemento "<textarea>" de html (línea 13).

Además del editor de texo, se definieron 3 etiquetas para notificar al usuario el estado en el que se encuentra el editor o el archivo seleccionado.

Existe la etiqueta "Contenido guardado" para que el usuario sea capaz de ejecutar o compilar el proyecto, esta etiqueta aparece cuando la variable "es_necesario_guardar" contiene un valor booleano falso (línea 8).

De manera contraria, si contiene un valor booleano verdadero la variable "es_necesario_guardar", aparece la etiqueta con el contenido "Contenido no guardado" (línea 9). Cuando esta etiqueta está presente, el usuario necesita guardar el contenido del editor de texto para ser capaz de compilar o ejecutar el proyecto.

Si el archivo seleccionado contiene el método main, entonces aparece la etiqueta "Archivo Main". Verifica el contenido de la variable "si_archivo_main_seleccionado" para mostrar la etiqueta (línea 10).

```
4   <!-- editor de texto -->
5   <div class="column col-md-9" id="main">
6       <h4>
7           {{nombre_archivo}}
8           <span class="label label-info" *ngIf="!es_necesario_guardar"> Contenido guardado</span>
9           <span class="label label-warning" *ngIf="es_necesario_guardar"> Contenido no guardado</span>
10          <span class="label label-success"*ngIf="si_archivo_main_seleccionado"> Archivo Main</span>
11      </h4>
12
13      <textarea id="editorTexto">
14      </textarea>
```

Figura 3.166 Definición del Editor de Texto

El editor de texto tiene la apariencia como se observa en la figura 3.167. En este caso, el archivo seleccionado contiene el método main (es el archivo main) y el contenido que se muestra en el editor de texto coincide con el de la nube.

Main.java Contenido guardado Archivo Main

```
1  /*creado por Ken Ruiz Inoue*/
2  public class Main {
3
4      public static void main(String[] args) {
5          System.out.print("Hola Mundo");
6      }
7
8  }
```

Figura 3.167 Apariencia del Editor de Texto con el Contenido Guardado

En la figura 3.168 se muestra la apariencia del editor de texto cuando el contenido del editor de texto no está guardado y el archivo seleccionado no contiene el método main (no es el archivo main).

Humano.java Contenido no guardado

```
1  /*creado por Ken Ruiz Inoue */
2  public class Humano {
3
4  }
```

Figura 3.168 Apariencia del Editor de Texto sin el Contenido Guardado

El área multifuncional contiene 2 pestañas. La pestaña para los mensajes de proyecto y la pestaña para las plantillas del proyecto.

La pestaña para los mensajes se divie en 2 partes. En la ventana de los mensajes que muestra todos los mensajes del proyecto y el área de texto para escribir el contenido de los mensajes.

La definición de la ventana de mensajes es como aparece en la figura 3.169. Cada mensaje del proyecto se obtiene del arreglo “mensajes_proyecto” (línea 31). El objeto mensaje tiene 3 pripiedades en total. La propiedad de url que muestra la foto de perfil del usuario (línea 35), la propiedad de contenido que muestra el texto de mensaje (línea 40) y la propiedad del propietario para mostrar el nombre del usuario que escribió el mensaje (línea 41).

```
28    <!-- contenedor de mensajes-->
29    <div id="contenedor-mensajes" class="pre-scrollable">
30        <!-- carga los mensajes -->
31        <div class="row" *ngFor="let mensaje of mensajes_proyecto" >
32
33            <div class="column col-md-2" id="main">
34                <div class="image-cropper">
35                    <img src={{mensaje.url_img_mensaje}} class="rounded" />
36                </div>
37            </div>
38
39            <div class="column col-md-10" id="main">
40                <h5>{{mensaje.contenido_mensaje}}</h5>
41                <h6>{{mensaje.propietario_mensaje}}</h6>
42                <hr class="style1">
43            </div>
44
45        </div>
46    </div>
```

Figura 3.169 Definición de la Ventana de Mensajes

La ventana de mensajes tiene la apariencia como se muestra en la figura 3.170.

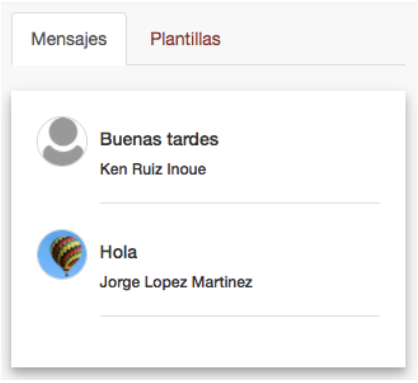

Figura 3.170 Apariencia de la Ventana de Mensajes

El área de texto para escribir los mensajes se definió como se observa en la figura 3.171. El área de texto se vincula con la variable "contenido_mensaje" para guardar el contenido del texto de manera dinámica en la variable (línea 53). Al hacer clic al botón definido en la línea 58, se llama el método "enviar_mensaje()" para generar un registro del mensaje en la base de datos de *firebase*.

```
50    <div class="column col-md-9">
51        <div class="form-group">
52            <textarea class="form-control" rows="3" id="area-texto-mensaje"
53            [(ngModel)]="contenido_mensaje"></textarea>
54        </div>
55    </div>
56
57    <div class="column col-md-3">
58        <button type="button" class="btn btn-info" (click)="enviar_mensaje()">
59            <span class="glyphicon glyphicon-send"></span>
60        </button>
```

Figura 3.171 Definición del Área de Texto para Escribir Mensajes

En la figura 3.172 se muestra la apariencia del área de texto de la pestaña de mensajes.

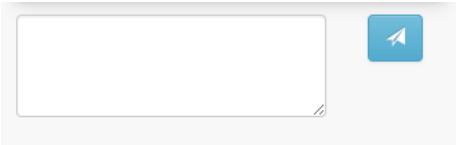

Figura 3.172 Apariencia del Área de Texto para Escribir Mensajes

La otra pestaña, que contiene el área de plantillas, se definió como se muestra en la figura 3.173 (primera parte). En la línea 67 se definió una etiqueta de encabezado que muestra la cantidad de plantillas a través de la variable "cantidad_plantilla". También se definió otro encabezado para mostrar el nombre de la plantilla seleccionada a través de la variable "plantilla_seleccionada" (línea 68).

Esta pestaña muestra al usuario el contenido de la plantilla en el área de texto definido en la línea 69. El contenido de la plantilla se puede modificar e insertar directamente hacia el editor de texto. Este contenido se conecta con la variable "area_texto_contenido_plantilla" (línea 69).

En el siguiente bloque de código se definieron los botones para navegar entre todas las plantillas que contiene el proyecto (líneas 71 a la 76). El botón para seleccionar la plantilla anterior, no aparece si el contador de plantillas contiene el valor "0", esto para evitar que el contador apunte un valor inválido en el arreglo de las plantillas (línea 73). También adoptó la misma lógica el botón para seleccionar la plantilla siguiente, no aparece el botón si el contador alcanza el valor de la variable "limite_contador" (línea 75).

```
66      <div id="plantillas" class="tab-pane fade">
67          <h4>Total de plantillas: <span class="badge">{{cantidad_plantillas}}</span></h4>
68          <h4 class="pager" *ngIf="cantidad_plantillas != 0">{{ds.plantilla_seleccionada.nombre_plantilla}}</h4>
69          <textarea class="form-control" rows="8" [(ngModel)]="area_texto_contenido_plantilla"></textarea>
70
71          <ul class="pager">
72              <li><a href="javascript:void(0)" (click)="seleccionar_plantilla_anterior()"
73                  *ngIf="contador_plantillas != 0"><span aria-hidden="true">&larr;</span></a></li>
74              <li><a href="javascript:void(0)" (click)="seleccionar_plantilla_siguiente()"
75                  *ngIf="contador_plantillas != limite_contador"><span aria-hidden="true">&rarr;</span></a></li>
76          </ul>
```

Figura 3.173 Definición de la Pestaña de Plantillas Parte 1

En la figura 3.174 se muestra el resto de la definición de la pestaña de plantillas. Se definieron 3 botones para crear, insertar o eliminar plantillas.

El botón "Crear" llama el método "abrir_modal_crear_plantilla" para abrir el modal correspondiente (línea 79). El método "insertar_plantilla()", se activa con el botón "Insertar" para agregar el contenido que se muestra en el área de esto en la ubicación del cursor en el editor de texto (línea 80). Finalmente, el botón "Eliminar" abre el modal de confirmación para eliminar con el método "abrir_modal_eliminar_plantilla".

```
78      <div>
79          <button class="bottomleft btn btn-success" (click)="abrir_modal_crear_plantilla()" >Crear</button>
80          <button class="bottomright btn btn-primary" (click)="insertar_plantilla()">Insertar</button>
81          <button class="bottomaligned btn btn-danger" (click)="abrir_modal_eliminar_plantilla()">Eliminar</button>
82      </div>
```

Figura 3.174 Definición de la Pestaña de Plantillas Parte 2

En la figura 3.175 se muestra la apariencia de la pestaña de plantillas.

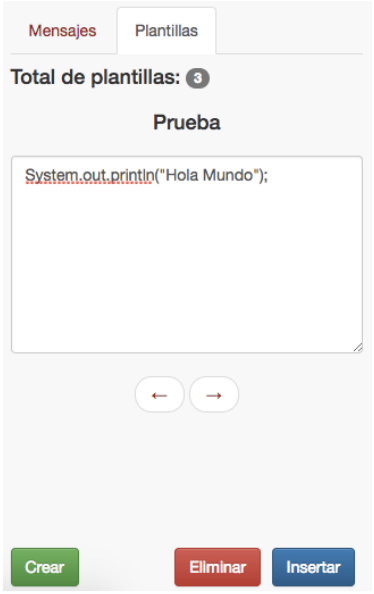

Figura 3.175 Apariencia de la Pestaña de Plantillas

Con esto se concluye la explicación del desarrollo del componente editor.

3.5.3 Desarrollo del Componente Topbar

Este componente define la barra superior en la interfaz gráficia del usuario y se encarga de recibir órdenes a través de la interfaz para realizarlas en el componente área de trabajo o editor. El usuario es capaz de navegar hacia la consola de proyectos, correr el proyecto, cerrar sesión y entre otras acciones desde la barra superior.

3.5.3.1 Desarrollo Lógico del Componente Topbar

Este componente es hijo del componente área de trabajo, por lo cual para establecer comunicación con el componente padre fue necesario utilizar el enfoque emisor-recepor con el decorador “Output”. Las definiciones de los emisores se observan en la figura 3.176. En total se definieron 4 decoradores “Output”, cada uno es una instancia del objeto “EventEmitter” que es capaz de emitir una señal al componente área de trabajo para ejecutar uno de sus métodos desde el componente topbar.

```
19  export class TopbarComponent implements AfterViewInit,OnInit {
20
21    @Output() eventoToggleAreaProyecto:EventEmitter<string> = new EventEmitter();
22    @Output() eventoCrearArchivo:EventEmitter<string> = new EventEmitter();
23    @Output() eventoEliminarArchivo:EventEmitter<string> = new EventEmitter();
24    @Output() eventoCerrarSesion:EventEmitter<string> = new EventEmitter();
```

Figura 3.176 Definición de los Emisores en “tobar.component.ts”

El componente topbar contiene en total 13 métodos para comunicarse con los componentes correspondientes. En la tabla 3.9 se obeservan los métodos.

Tabla 3.9 Métodos del Componente Topbar

Método	Funcionalidad
ngAfterViewInit()	Define el mensaje de las etiquetas flotantes y obtiene la referencia del componente editor en una variable.
toggle_area_proyecto()	Se comunica con el componente área de trabajo para esconder o mostrar el área de proyecto según el estado en el que se encuentra. Utiliza el decorador "Output."
crear_archivo()	Se comunica con el componente área de trabajo para abrir el modal para crear archivos. Utiliza el decorador "Output."
intento_guardar_contenido()	Se comunica con el componente editor para llamar su método "intento_guardar_contenido()".
eliminar_archivo()	Se comunica con el componente área de trabajo para llamar su método "inicio_eliminar_archivo()". Utiliza el decorador "Output."
cambiar_estado_area_ejecucion()	Cambia el contenido de la variable "si_esconder_area_ejecucion" del servicio datos para mostrar o esconder el área de ejecución.
cambiar_estado_area_multifuncional()	Se comunica con el componente editor para mostrar o esconder el área de multifunciones.
deshacer()	Ejecuta el método "deshacer()" del componente editor.
rehacer()	Ejecuta el método "rehacer()" del componente editor.
ejecutar()	Verifica ciertas condiciones para llamar el método "ejecutar()" que se encuentra en el servicio auxiliar.
compilar()	Verifica ciertas condiciones para llamar el método "compilar()" que se encuentra en el servicio auxiliar.
cerrar_sesion()	Ejecuta el método "cerrar_sesion()" del componente área de trabajo. Utiliza el decorador "Output."

Método "ngAfterViewInit()": Este método es el primero en llamarse debido a que no se definió un método "ngOnInit()" en este componente. En la figura 3.177 se observa la definición del método. Define un atributo para mostrar de manera apropiada las etiquetas flotantes que aparecen al colocar el cursor encima (línea 40). Obtiene la referencia del componente editor desde el servicio auxiliar y la guarda en la variable "refEditor" (línea 42)

para comunicarse con el componente editor a través de la referencia y no de un emisor (decorador "Output").

```
38    ngAfterViewInit() {
39        $(function () {
40            $('[data-toggle="tooltip"]').tooltip()
41        });
42        this.auxServ.referencia_componente_editor = this.refEditor;
43    }
```

Figura 3.177 Método "ngAfterViewInit()" en "tobar.component.ts"

Método "toggle_area_proyecto()": Hace el uso del método "emit()" (línea 46) sobre el tipo de objeto "EventEmitter" como se observa en la figura 3.178. Lo mismo sucede con los métodos "crear_archivo()", "eliminar_archivo()" y "cerrar_sesion()".

```
45    toggle_area_proyecto() {
46        this.eventoToggleAreaProyecto.emit('complete');
47    }
```

Figura 3.178 Método "toggle_area_proyecto()" en "tobar.component.ts"

Método "intento_guardar_contenido()": Como se observa en la figura 3.179, este método ejecuta un método del componente editor. El método se ejecuta a través de la referencia guardada en la variable "refEditor" (línea 60). Con los métodos "guardar_contenido()", "cambiar_estado_area_ejecucion()", "deshacer()", "rehacer()" y "seleccionar_contenido()" se utiliza el mismo enfoque.

```
59    intento_guardar_contenito() {
60        this.refEditor.intento_guardar_contenido();
61    }
```

Figura 3.179 Método "intento_guardar_contenido()" en "tobar.component.ts"

Método "cambiar_estado_area_ejecucion()": Este método actualiza el estado booleano de una variable que se encuentra en el servicio auxiliar para cambiar el estado del área de ejecución. En la figura 3.180 se observa la definición del método.

```
78    cambiar_estado_area_ejecucion() {
79        if(this.ds.si_esconder_area_ejecucion == true) {
80            this.ds.si_esconder_area_ejecucion = false;
81        } else {
82            this.ds.si_esconder_area_ejecucion = true;
83        }
84    }
```

Figura 3.180 Método "cambiar_estado_area_ejecucion()" en "tobar.component.ts"

Método "ejecutar()": Este método verifica 5 casos diferentes para mostrar el mensaje de alrte apropiado o ejecutar el proyecto. En la figura 3.181 se observa la definición del método.

A continuación, se enlistan los casos de falla al ejecutar el proyecto que muestra las alertas.

Caso de Falla 1: Si se encuentra en plena comunicación la aplicación con el servidor (línea 103).

Caso de Falla 2: Si el proyecto se encuentra vacío y no contiene algún archivo (línea 106).

Caso de Falla 3: Si el archivo seleccionado se ha actualizado y no se encuentra guardado ese contenido (línea 108).

Caso de Falla 4: Si no existe un archivo main en el proyecto (línea 110).

Si no coincide con los casos enlistados anteriores, el método ejecuta el proyecto a través del método "ejecutar()" del servicio de auxilio (línea 113) y establece un valor verdadero en la variable "comunicandose_con_el_servidor" (línea 114) para no permitir múltiples peticiones hacia el servidor.

```
102   ejecutar() {
103       if(this.ds.comunicandose_con_el_servidor == true) {
104           alert("Estableciendo comunicación con el servidor, espere un momento.");
105       } else if(this.ds.si_proyecto_vacio) {
106           alert("No se puede ejecutar, el proyecto está vacío");
107       } else if(this.ds.activar_advertencia) {
108           alert("Primero guarde los cambios para ejecutar el proyecto");
109       } else if (!this.ds.si_existe_main){
110           alert("Para ejecutar el proyecto es necesario contener un main.");
111       } else {
112           //crear archivo zip y subirlo en el almacén de firebase
113           this.auxServ.ejecutar(this.refEditor);
114           this.ds.comunicandose_con_el_servidor = true;
115       }
116   }
```

Figura 3.181 Método "ejecutar()" en "tobar.component.ts"

Método "compilar()": Este método es similar al método "ejecutar()", pero no requiere verificar si existe un archivo main. En la figura 3.182 se observa la definición del método.

```
118   compilar() {
119       if(this.ds.comunicandose_con_el_servidor == true) {
120           alert("Estableciendo comunicación con el servidor, espere un momento.");
121       } else if(this.ds.si_proyecto_vacio) {
122           alert("No se puede compilar, el proyecto está vacío");
123       } else if (this.ds.activar_advertencia) {
124           alert("Primero guarde los cambios para poder compilar el proyecto");
125       } else {
126           //crear archivo zip y subirlo en el almacén de firebase
127           this.auxServ.compilar(this.refEditor);
128           this.ds.comunicandose_con_el_servidor = true;
129       }
130   }
```

Figura 3.182 Método "compilar()" en "tobar.component.ts"

Con esto se concluye el desarrollo lógico del componente topbar.

3.5.3.2 Desarrollo Gráfico del Componente Topbar

Como se había mencionado este componente dibuja la barra superior en la aplicación. Esta barra superior se divide en 4 partes como se detallan a continuación.

Parte Gráfica 1 Botón Logo: Es un botón que se ubica en la parte izquierda de la barra para permitir al usuario regresar a la consola de proyectos con este botón. En la figura 3.183 se observa la definición del botón.

Si el tamaño de la pantalla es reducido o es visualizado desde un dispositivo móvil, se encapsulan todos los elementos en un botón a excepción del botón logo gracias al atributo "data-toggle" (línea 8) que ofrece *bootstrap*.

El botón de logo utiliza el atributo de "routerLink" para navegar hacia la pantalla de consola de proyectos (línea 15).

```
6   <div class="navbar-header">
7       <!-- Botón reajustable por tamaño -->
8       <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
9       data-target="#bs-example-navbar-collapse-1" aria-expanded="false">
10      <span class="sr-only">Toggle navigation</span>
11      <span class="icon-bar"></span>
12      <span class="icon-bar"></span>
13      <span class="icon-bar"></span>
14      </button>
15      <a class="navbar-brand" href="javascript:void(0)" routerLink="/consola">CodeCloud</a>
16  </div>
```

Figura 3.183 Definición del Botón Logo y Encapsulamiento del Resto

Apariencia del botón logo en la figura 3.184.

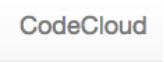

Figura 3.184 Apariencia del Botón Logo

Apariencia del botón logo con los elementos encapsulados en la figura 3.185.

Figura 3.185 Apariencia del Botón Logo con el Resto de los Elementos Encapsulados

Parte Gráfica 2 Barra de Menús: Esta parte de la barra, situada a la derecha del botón logo, proporcionan al usuario 4 menús que contienen en total 10 opciones. En la tabla 3.10 se observan las opciones que el usuario puede escoger en la barra superior.

Tabla 3.10 Opciones Disponibles en la Barra Superior

Contenedor	Opción	Acción
Archivo	Nuevo Archivo	Abre el modal para crear un archivo.
Archivo	Guardar Archivo	Guarda el contenido del archivo seleccionado.
Archivo	Eliminar Archivo	Abre el modal para confirmar la eliminación del archivo seleccionado.
Edición	Deshacer	Regresa al contenido de texto anterior desde el historial de editor de texto.
Edición	Rehacer	Regresa al contenido de texto posteior desde el historial de editor de texto.
Visualización	Área de Proyecto	Muestra o esconde el área de proyecto.
Visualización	Multifunciones	Muestra o esconde el área de multifunciones.
Visualización	Área de Ejecución	Muestra o esconde el área de ejecución.

Correr	Compilar	Compila el proyecto e imprime el resultado en el área de ejecución.
Correr	Ejecutar	Ejecuta el proyecto e imprime el resultado en el área de ejecución.

En la figura 3.186 se observa la definición de las opciones que contiene el menú "Archivo". Se utilizó la clase "dropdown-menu" para la creación de un menú despegable (línea 27). Contiene 3 opciones diferentes que se mencionaron anteriormente, cada una de las opciones llama un método del componente topbar (líneas 28 a la 30). Los menús "Edición", "Visualización" y "Correr" se definieron de la misma manera.

```
23    <!-- Archivo -->
24    <li class="dropdown">
25        <a href="#" class="dropdown-toggle" data-toggle="dropdown"
26        role="button" aria-haspopup="true" aria-expanded="false">Archivo<span class="caret"></span></a>
27        <ul class="dropdown-menu">
28            <li><a href="javascript:void(0)" (click)="crear_archivo()">Nuevo archivo</a></li>
29            <li><a href="javascript:void(0)" (click)="intento_guardar_contenito()">Guardar Archivo</a></li>
30            <li><a href="javascript:void(0)" (click)="eliminar_archivo()">Eliminar Archivo</a></li>
31        </ul>
32    </li>
```

Figura 3.186 Definición del Menú "Archivo" y sus Opciones

En la figura 3.187 se observa la apariencia del menú "Archivo" con sus opciones correspondientes.

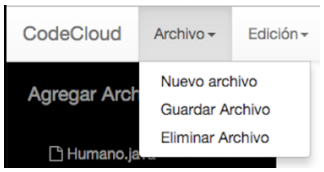

Figura 3.187 Apariencia del Menú "Archivo" y sus Opciones

Parte Gráfica 3 Acciones Rápidas: Son botones que se representan a través de un símbolo la acción a realizar. En total hay 3 opcione que son "Guardar Archivo", "Ejecutar" y "Copiar Contenido". En la figura 3.188 se muestra la definición de la acción "Guardar Archivo".

El botón tiene el método "intento_guardar_contenido()" definido para llamarlo al ser presionado (línea 77). También contiene un título a través del atributo "title" (línea 79), para mostrarlo al colocar el cursor encima del botón. El resto de las acciones rápidas se definieron de la misma manera.

```
75    <ul class="nav navbar-nav navbar-right">
76        <li class="dropdown"><!-- Guardar Archivo-->
77            <a href="javascript:void(0)" (click)="intento_guardar_contenito()">
78                <span class="glyphicon glyphicon-floppy-save" aria-hidden="true"
79                data-toggle="tooltip" data-placement="bottom" title="Guardar archivo"></span></a>
80        </li>
```

Figura 3.188 Definición de la Acción Rápida "Guardar Contenido"

En la figura 3.189 se observa la definición de las acciones rápidas.

Figura 3.189 Apariencia de las Acciones Rápidas

Parte Gráfica 4 Foto de Perfil: Esta parte de la barra muestra la foto de perfil del usuario y proporciona la opción de cerrar la sesión.

En la figura 3.190 se observa la definición de la foto de perfil. La dirección de la foto de perfil está guardada en la variable "usuario.imgurl" en el servicio auxiliar y de manera dinámica obtiene el la dirección (línea 69). Muestra el nombre de usuario de la misma manera que se hace con la foto de perfil (línea 70). Se define un botón de menú para cerrar la sesión a través del método "cerrar_sesion()" (línea 73).

```
67    <li class="dropdown">
68        <a id="user-profile" href="#" class="dropdown-toggle" data-toggle="dropdown">
69            <img src={{ds.usuario.imgurl}} class="img-responsive img-thumbnail img-circle">
70            {{ds.usuario.nombre}}
71        </a>
72        <ul class="dropdown-menu dropdown-block" role="menu">
73            <li><a href="javascript:void(0)" (click)="cerrar_sesion()">Cerrar Sesión</a></li>
74        </ul>
75    </li>
```

Figura 3.190 Definición de la Foto de Perfil

En la figura 3.191 se observa la apariencia de la foto de perfil con su opción para cerrar la sesión.

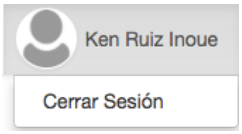

Figura 3.191 Apariencia de la Foto de Perfil y la Opción para Cerrar Sesión

Con esto se concluye el desarrollo gráfico del componente topbar.

3.6 Alojamiento del Cliente al Servidor

La aplicación desarrollada se alojó en una maquina virtual del servicio de *Amazon Web Services*. A continuación, se detalla todo el proceso de alojamiento de la aplicación a producción para alcanzarla a través de un navegador con la dirección ip pública.

Primero fue necesario compilar la aplicación desarrollada en *angular* a un directorio. Con el comando "ng build" se compila y crea el directorio "dist" que contiene la aplicación web lista para ser alojada. Para facilitar la carga de la aplicación en la máquina virtual, se cargó el directorio dist generado en un repositorio de *github* para descargar el contenido en la máquina vritual a través de comandos *git*.

Se creó una instancia de la máquina virtual con la imagen "Ubuntu Server 16.04 LTS" de tipo *t2.micro* de propósito general (figura 3.192) que ofrece *Amazon Web Services*.

Figura 3.192 Selección de la Imagen Ubuntu Server 16.04 LTS

Se configuró un grupo de seguridad de todo tipo de tráfico (*multipunto*) para permitir la conección a la máquina virtual a través de cualquier navegador web (figura 3.193).

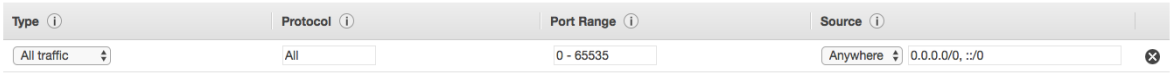

Figura 3.193 Configuración de Grupo de Seguridad

Una vez creada la máquina virtual, se descarga el par de llaves de seguridad y se revisó la dirección ip pública para conectarse a la instancia a través de *SSH*. En la figura 3.194 se muestra el comando que se introdujo en la terminal para conectarse a la máquina virtual donde se alojará la aplicación.

```
MacBook-Air-de-Ken:~ kenruizinoue$ ssh -i ~/.ssh/tesis-key.pem ubuntu@34.205.17.240
```

Figura 3.194 Comando para Conectarse a la Máquina Virtual

Al conectarse con la máquina virtual, se instaló el servidor *nginx* para alojar la aplicación. Para instalar el servidor se utilizó el comando que aparece en la figura 3.195 en la máquina virtual.

```
ubuntu@ip-172-31-68-171:~$ sudo apt-get install nginx
```

Figura 3.195 Comando para Installar el Servidor Nginx

Después de haber instalado el servidor, se alojó la aplicación en el directorio "var/www/html". Por defecto el servidor *nginx* configura esta dirección para servir la aplicación. Para descargar la aplicación web, se creó una carpeta de prueba para almacenar de manera temporal la

aplicación desde el repositorio de *github*. Una vez guardada la aplicación en el directorio temporal, se movió el contenido hacia la dirección "var/www/html" con el comando que aparece en la figura 3.196.

```
ubuntu@ip-172-31-68-171:~$ sudo cp -a /home/ubuntu/prueba/. /var/www/html
```

Figura 3.196 Comando para Mover el Contenido de la Carpeta

Con esto queda servida la aplicación y puede ser alcanzada a través de internet con la dirección pública.

El servidor *nginx* es una buena opción para servir aplicaciones singulares, a pesar de que la aplicación está desarrollada en *Angular*, se generaron 5 rutas diferentes dentro de la aplicación para permitir la navegación al usuario. Cuando una aplicación tiene diferentes rutas dentro de la dirección de ip pública, el servidor de *nginx* no logra identificar rutas y manda el *error 404*. Para evitar este error y causar confusión al usuario final, se tuvo que configurar el archivo "default" ubicada en la dirección "/etc/nginx/sites-enabled/".

En la figura 3.197 se observa el contenido del archivo configurado para evitar el error mencionado. La línea "try_files $uri $uri/ =404" significa que al no encontrar el recurso solicitado, manda el *error 404* al usuario, es por eso que se comentó y se escribió la línea "try_files $uri $uri/ /index.html" para reenviar al usuario a la ruta por defecto.

```
location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        #try_files $uri $uri/ =404;
        try_files $uri $uri/ /index.html;
}
```

Figura 3.197 Configuración para Evitar el Error 404

Con esto termina la explicación del alojamiento de la aplicación en el servidor.

3.7 Desarrollo del Servicio Web

Como se definió en capítulos anteriores el servicio web *REST* se distribuye en cuatro acciones principales: Recepción de datos, Almacenamiento, Compilación y Ejecución del código, por lo que la aplicación web usa dicho servicio para completar con las acciones principales como son las de compilar y ejecutar el código que el usuario desee.

Para ello es necesario previamente establecer un ambiente que permita el alojamiento del servicio web para ser consumidos por la aplicación web a través de internet.

3.7.1 Establecimiento del Ambiente

Para realizar el desarrollo del servicio web es necesario un servidor, por lo que para este caso se utiliza *Digital Ocean*, el cual almacena un sistema operativo *Ubuntu* con el cual montaremos el servidor en la nube, el servicio web se basa en *Node Js.*

La instalación de *Node Js* y las librerías y lenguajes implementados se muestran en la tabla 3.11 con sus comandos correspondientes (la funcionalidad de cada librería se puede encontrar en el Glosario de Términos). Para configurar el sistema operativo almacenado en *Digital Ocean* se siguieron los pasos indicados por la misma plataforma [13].

Tabla 3.11 Instalación de Herramientas necesarias

Herramienta	Comando
Node Js	sudo apt-get install npm
Express	npm install -g express
Body Parser	npm install body-parser
Rimraf	npm install rimraf
ExecSync	npm install execSync
Java	sudo apt-get install default-jre
Javac	sudo apt-get install default-jdk
PM2	npm install pm2 -g

Todas las librerías y los lenguajes de programación implementados se comunican entre sí, como se puede ver en la figura 3.198 la petición para consumir el servicio web proviene de internet, esta información es procesada por *Node js* y *Express* (cabe destacar que *Node Js* está presente en todo el ciclo, ya que es el lenguaje de programación en el que está implementado el servicio web) una vez que la información ha sido recibida la librería *Body Parser* nos permite mapear de forma adecuada el *json* recibido en el paso anterior, *ExecSync* permite ejecutar los comandos (*javac*, *java*) que son los encargados de compilar y ejecutar el código enviado por el usuario de forma síncrona, una vez que se concluyen los procedimientos anteriores Rimraf permite eliminar todos los archivos (.java y .class) que se generaron en la descarga, compilación y ejecución del código, una vez concluido la respuesta

de cada uno de los procesos es mapeada y actualizada con el resultado de la compilación o ejecución del código y después ser enviada por el servidor a Internet.

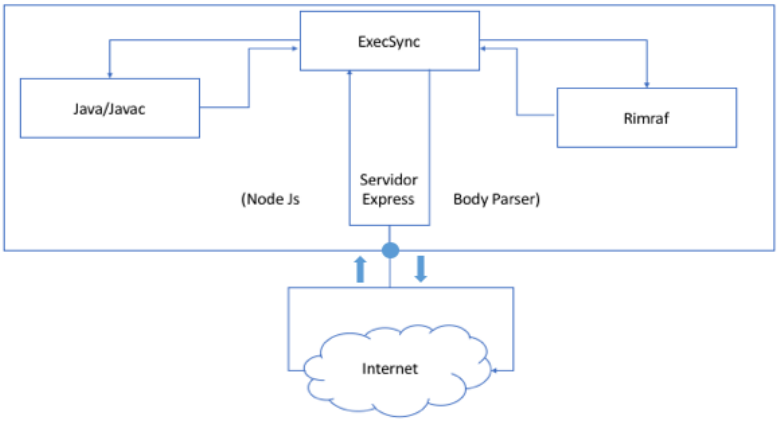

Figura 3.198 Interacción Librerías y Lenguajes

3.7.2 Procesamiento de Información

Para que el servicio web pueda funcionar de forma adecuada la información que es enviada al mismo debe ser en formato de texto *JSON*; El servicio web desempeña principalmente dos funciones: compilar y ejecutar código, cada acción define su propia estructura *JSON,* las cuales son muy similares, ya que básicamente utilizan la misma información para compilar y ejecutar el código, en la figura 3.199 se muestra la estructura que espera el servicio en caso de que la acción a ejecutar sea la de compilar.

Dicho JSON está estructurado por un Usuario y una contraseña con el cual se puede validar la autenticación de quien consume dicho servicio web, aunado a eso se debe definir el número de archivos que contiene la solución, y la url de donde se podrá descargar el proyecto para poder ser compilado, El nombre del proyecto y el nombre de los archivos que contiene dicha solución.

```
// Compilar
{
   "Usuario" : "CodeCloud",
   "Acceso" : "H467!a89lopz#0l",
   "NumeroArchivos" : "2",
   "Url" : "https://firebasestorage.googleapis.com/v0/b/tesis-4b2ae.appspot.com/o/test-proyect.zip?alt=media&token=07339f1b-5d7a-4ed9-af93-d838e8
   "NombreProyecto" : "test-proyect",
   "Archivos" : [
        {
            "NombreArchivo" : "Test"
        },
        {
            "NombreArchivo" : "Human"
        }
   ]
}
```

Figura 3.199 Estructura JSON Compilar

La notación JSON correspondiente a la acción ejecutar tiene únicamente un atributo más en comparación al de compilar, este atributo permite identificar al archivo principal, el que es configurado por el usuario como el inicio de su aplicación, en pocas palabras es el que contiene el método main de la aplicación Java, este atributo como se muestra en la figura

3.200 no es necesario que esté presente en todos los archivos, sino únicamente en el archivo principal y debe contener el valor en "True".

```
// Ejecutar
{
   "Usuario" : "CodeCloud",
   "Acceso" : "H467!a89lopz#0l",
   "NumeroArchivos" : "2",
   "Url" : "https://firebasestorage.googleapis.com/v0/b/tesis-4b2ae.appspot.com/o/test-proyect.zip?alt=media&token=07339f1b-5d7a-4ed9-af93-d83
   "NombreProyecto" : "test-proyect",
   "Archivos" : [
        {
            "NombreArchivo" : "Test",
            "Main" : "True"
        },
        {
            "NombreArchivo" : "Human"
        }
   ]
}
```

Figura 3.200 Estructura JSON Ejecutar

Cabe destacar que esta información es invisible para el usuario, el JSON es formado por la aplicación web y es enviada al servicio web cuando este quiere realizar las acciones de compilar o ejecutar, la aplicación web se encarga de forman este archivo dependiendo de la configuración del proyecto del usuario.

3.7.3 Recepción de Datos

La parte del servidor que implementa la recepción de los datos es básicamente el servicio REST, implementa algunas de las librerías que se mencionaron en la sección "3.4.1 Estableciendo el ambiente" y son importadas al proyecto como se muestra en la figura 3.201.

```
var express = require('express'),
    bodyParser= require('body-parser');

var app= express();

var compilarCodigo = require('./Compilar/compilarCodigo.js');

var ejecutarCodigo = require('./Ejecutar/ejecutarCodigo.js');
```

Figura 3.201 Librerías Importadas

También se importan los archivos que se encargan de la compilación y ejecución del código y los cuales se comunican directamente con la implementación del servicio REST. Una vez importados los archivos necesarios, se deben definir las cabeceras del servicio web, estas son las encargadas de permitir la comunicación del servicio con Internet, estas se definen como se muestra en la figura 3.202.

```
app.use(function (req, res, next) {

    res.setHeader('Access-Control-Allow-Origin', 'http://localhost:4200');

    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');

    res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');

    res.setHeader('Access-Control-Allow-Credentials', true);

    next();
});
```

Figura 3.202 Cabeceras para el Servicio Web

Para la implementación de compilar y ejecutar se utiliza un direccionamiento distinto para cada una de ellas, lo cual permite que dependiendo la acción el servicio sea consumido por una url dedicada a ejecutar o compilar, "http://104.131.86.147:8000/CodeCloud/Compilar" y "http://104.131.86.147:8000/CodeCloud/Ejecutar". El direccionamiento para compilar es el que se muestra en la figura 3.203

```
direccionamiento.route('/Compilar')
    .post(function(req, res){
        if(validarUsuario(req.body.Usuario, req.body.Acceso)){
            try{
                compilarCodigo.compilar(req, 1, function(respuesta){
                    res.json(respuesta);
                });
            }catch(e){
                resp.body.Resultado = "Información erronea";
                res.send(resp);
            }

        }else{
            res.send(resp);
        }
    });
```

Figura 3.203 Direccionamiento Compilar

El direccionamiento para ejecutar es implementado de la misma forma, la única diferencia es que este utiliza el método ejecutar y no el de compilar como se observa en la imagen 3.203 y la dirección cambia a "/Ejecutar". Para que el servicio web pueda ser consumido con las url que se mencionaron anteriormente es necesaria la siguiente línea de código, la que se muestra en la figura 3.204.

```
app.use('/CodeCloud', direccionamiento);
```

Figura 3.204 Implementación Direccionamiento

3.6.4 Almacenamiento del Código

El almacenamiento del código se divide en dos acciones, la de descargar el código de la url que es proporcionada por la página web, dicha url contiene todos los archivos necesarios para la compilación y/o ejecución del código y la acción de descomprimir, ya que el código es

enviado en un zip para poder realizar una sola descarga y no implementar una descarga por cada archivo de la solución creada por el usuario. La implementación de la descarga del código como se muestra en la figura 3.205 abre un proceso en la consola que es el encargado de descargar el archivo zip usando *curl,* en caso de que la descarga no sea exitosa se fija una respuesta con un resultado del proceso igual a uno, lo cual impedirá que el código continúe ejecutándose y responderá a la llamada del servicio web con el error que se generó al tratar de realizar la descarga.

```
// Descargar Proyecto
var descargarProyecto = function(url, nombreArchivo, callback) {

    if (!fs.existsSync(ubicacionDescarga)){
        fs.mkdirSync(ubicacionDescarga);
    }else{
        eliminarCarpeta();
    }

    var child = exec('cd ' +  ubicacionDescarga + ' && curl -o ' + nombreArchivo + '.zip ' + url,
        function (error, stdout, stderr){

            if(error !== null){
                respuesta[0].Resultado = error;
                respuesta[0].Proceso = 1;
            }else{
                respuesta[0].Resultado = "Descarga exitosa";
                respuesta[0].Proceso = "Ejecutado";
            }
            callback(respuesta);
        });
};
```

Figura 3.205 Código para Descarga Archivo Zip

El proceso de descomprimir el archivo zip es muy parecido al de descargarlo, se abre una conexión con la línea de comandos y se genera un hilo hijo que se encarga de descomprimir el archivo, al igual que el proceso de descarga se implementa un manejo de error y en caso de que no se descomprima exitosamente se envía el error de regreso al sitio web. La implementación para descomprimir el archivo se observa en la figura 3.206.

```
// Descomprimir Zip
var descomprimirZip = function(nombreArchivo, callback){

    var child = exec('cd ' + ubicacionDescarga + ' && unzip ' + nombreArchivo,
        function (error, stdout, stderr){
            if(error !== null){
                respuesta[0].Resultado = error;
                respuesta[0].Proceso = 1;
            }else{
                respuesta[0].Resultado = "Compilado exitosamente";
                respuesta[0].Proceso = "Ejecutado";
            }
            callback(respuesta);
        });
}
```

Figura 3.206 Código para Descomprimir Archivo Zip

3.7.5 Compilar Código

Este proceso utiliza los procesos anteriores, si uno de estos llegase a fallar nunca se ejecutaría la compilación del código, ya que debido a las características que plantea el servicio web todas las tareas deben ser síncronas, antes de ejecutar este proceso se realiza una validación

de si los procesos ejecutados anteriormente se realizaron de forma exitosa, de lo contrario la acción se aborta.

Esta función es implementada al abrir un hilo que vaya a la línea de comandos y se ubique en la dirección del archivo, una vez que se encuentre en la ruta especificada debe compilar el comando "*javac* NombreArchivo,java" que es el que ejecutará la compilación, cabe destacar que la implementación de este hilo es particularmente diferente a los anteriores, ya que cuando en *Node* se implementa "*exec*" se genera un hilo independiente al del proceso que se está ejecutando actualmente, por lo que se tienen que implementar un *callback* para que la función no continúe hasta que el resultado de este hilo hijo haya concluido, pero al utilizar "*execSync*" no genera ningún hilo hijo, sino que hace el llamado a la línea de comandos a través del mismo hilo que lleva la ejecución del servicio web.

Esto se implementó debido a la cantidad de archivos que se necesitan compilar, ya que los archivos pueden depender de otros, el método de compilar es el que se observa en la figura 3.207.

```
// Comiplar Codigo
var compilarCodigo = function(nombreArchivo, nombreProyecto, callback){

    try{
        var child = execSync('cd Codigo/' + nombreProyecto + '/ && javac ' +
        nombreArchivo + '.java');
        console.log("compilar " + nombreArchivo);
        respuesta[0].Resultado = "Compilado exitosamente";
        respuesta[0].Proceso = "Ejecutado";
    }catch(err){
        respuesta[0].Resultado = err.message;
        respuesta[0].Proceso = 1;
    }finally{
        callback(respuesta);
    }
}
```

Figura 3.207 Código para Compilar Código

Una vez que se concluye el proceso de compilado, el servicio web realiza una validación, en caso de que el propósito de este sea compilar procede a eliminar los archivos generados ".class" y ".java" por el proceso de compilado, si el propósito del servicio es ejecutar código, no realizará la eliminación de los archivos ya que estos son necesarios para la ejecución del código, esta función utiliza la librería *Rimraf* y se implementa como se muestra en la figura 3.208.

```
// Eliminar Archivos
var eliminarCarpeta = function(){
    rimraf(ubicacionDescarga, function () {
        if (!fs.existsSync(ubicacionDescarga)){
            fs.mkdirSync(ubicacionDescarga);
        }
    });
}
```

Figura 3.208 Código para Eliminar Archivos

3.7.6 Ejecutar Código

Cuando todos los procesos se ejecutan de forma exitosa se dispara la función "hiloEjecutar". Como se observa en la figura 3.2, a diferencia del proceso de compilar no es necesario realizar la llamada de un hilo mediante "*execSync*". Únicamente implementando el "*callback"* es más que suficiente para ejecutar el archivo principal, esta función implementa un "*ciclo for*" que recorre todos los archivos que contiene el texto "*JSON*" y selecciona el que tiene el atributo "Main" igualado a "True" y este es el que se ejecuta en la línea de comandos.

```
var hiloEjecutar = function(jsonInfo, respuesta, callback){
    var numeroArchivos = parseInt(jsonInfo.body.NumeroArchivos, 10);

    for (var i = 0; i < numeroArchivos; i++) {
        if(jsonInfo.body.Archivos[i]['Main'] ==  "True"){
            var comando = 'cd Codigo/' + jsonInfo.body.NombreProyecto + '/ && java ' +
            jsonInfo.body.Archivos[i]['NombreArchivo'];

            var child = exec(comando, function (error, stdout, stderr){

                if(error !== null){
                    respuesta[0].Resultado = error;
                    respuesta[0].Proceso = "Error";
                }else{
                    console.log("Ejecutando " + jsonInfo.body.NombreProyecto);
                    respuesta[0].Resultado = stdout;
                    respuesta[0].Proceso = "Ejecutado";
                }

                callback(respuesta);
            });
        }
    }
```

Figura 3.209 Código para Ejecutar

Cuando concluye el proceso de ejecución, el servidor procede con la eliminación de los archivos generados, de la misma forma que lo hace el proceso de compilado, como se mostró en la figura 3.208.

Con esto se concluye el desarrollo del servicio web.

Capítulo 4

Pruebas del Sistema

Las pruebas del software son de suma importancia para así garantizar la calidad del producto a entregar, además de que estas representan una revisión final de las especificaciones, diseño y codificación del mismo. Estas pruebas se encargan de verificar y/o validar los componentes del sistema tanto de forma independiente como en conjunto, a fin de que funcionen en la forma que inicialmente se planeó.

Existen diferentes tipos de pruebas, como por ejemplo las estáticas las cuales prueban por componente o nivel de especificación sin ejecución del código del software y y dinámicas que son las pruebas que para llevarse a cabo requieren la ejecución del código de la aplicación respectivamente. Las pruebas elegidas son las dinámicas ya que se desea una mayor profundidad en la ejecución de estas.

Dentro de las pruebas dinámicas existen las manuales y las automáticas. Las pruebas manuales se enfocan en la página web, el *frontend* del sistema, dichas pruebas se enfocan en la interacción del usuario con el sistema, las acciones que este realiza, todo a través de la interfaz gráfica.

Dichas pruebas pueden tener diferentes enfoques, como lo son las de caja blanca, caja negra y pruebas aleatorias; las pruebas de caja blanca se enfocan en los detalles procedimentales de la aplicación, por lo que están ligadas al código fuente, las pruebas aleatorias son las que seleccionan casos de pruebas con el fin de asemejas un perdil operativo, esta técnica se puede usar para probar atributos no funcionales como la fiabilidad y el rendimiento.

Y por último pruebas de caja negra, estas se llevan a cabo fijando entradas al sistema, las cuales son procesadas y arrojan una respuesta específica dependiendo de la acción a probar. Las pruebas de caja negra es el enfoque en el cual las pruebas realizadas para este sistema fueron basadas.

Las pruebas manuales contemplan escenarios en los que la información es proporcionada de forma adecuada, en la que se mantiene una integridad de datos la cual mantiene un flujo adecuado del programa, pero a su vez, contempla pruebas en las que el software es probado sin que los datos de entrada al sistema estén íntegramente proporcionados, por lo cual el programa será incapaz de concluir con la acción deseada y debe ser capaz de continuar sin comprometer la estabilidad del mismo y notificar al usuario que la información proporcionada es errónea y necesita intentarlo de nuevo.

Mientras que las pruebas automáticas tienen un enfoque para el *backend* del sistema, básicamente probar la funcionalidad y disponibilidad del servicio web, calcular el número de conexiones que puede recibir el servicio web sin verse afectado y es un gran ejemplo de estas pruebas, y se catalogan como automáticas ya que para poder probar este escenario es necesario implementar un módulo automatizado que permite simular estas condiciones en el sistema.

4.1 Pruebas Automáticas

Las pruebas automatizadas son implementadas haciendo uso *Node*, con el fin de realizar las llamadas al servicio web de forma rápida y eficaz. En la tabla 4.1 se describen las principales razones por las cuales esta herramienta fue seleccionada para la elaboración de las pruebas automatizadas para el servicio web.

Tabla 4.1 Ventajas de las Herramientas Para el desarrollo de las Pruebas

Herramienta	Ventajas
Node	Lenguaje simplificado y rápido
	Flexible
	Programación ordenada y simple
	Fácil

4.1.1 Casos de Pruebas

Los casos de prueba para el servicio web se dividen en los puntos que se muestran en la tabla 4.2

Tabla 4.2 Principales Características a Probar

Característica	Detalle
Accesibilidad del servicio web	Verificar si la disponibilidad del servicio web
Integridad de la Información	Verificar la respuesta del servicio web sea la adecuada en caso de que la información que se le esté proporcionando sea o no la correcta
Validación de Credenciales	Verificar el acceso o denegación al servicio dependiendo un usuario y contraseña
Compilar	Verificar que el proceso de compilado se ejecute de manera exitosa
Ejecutar	Ejecutar aplicaciones basadas en lenguaje Java
Capacidad	Verificar el número de conexiones que el servicio web es capaz de almacenar

Para verificar la accesibilidad del servicio web se generan los casos de prueba que se muestran de la tabla 4.3 a la tabla 4.6.

Tabla 4.3 Caso de Prueba *Ping* Servicio Web

Caso de Prueba 01	Ping Servicio Web
Propósito	Verificar que el servicio contratado que almacena el servicio web esté disponible

Prerrequisitos	1. *Script* en *Node* que permita hacer un ping
Pasos	1. Ejecutar el *script*
Resultado esperado	Obtener un mensaje de conexión exitosa

Tabla 4.4 Caso de Prueba Consumir Servicio Web

Caso de Prueba 02	Consumir Servicio Web
Propósito	Verificar que el servicio web recibe información
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON como el que se muestra en la figura 3.86
Pasos	1. Definir archivo *JSON* acorde a las necesidades de la prueba 2. Ejecutar *script*
Resultado esperado	El servicio web debe responder

Tabla 4.5 Caso de Prueba Consumir Servicio Web con Puerto Diferente del 8000

Caso de Prueba 03	Consumir Servicio Web con Puerto Diferente del 8000
Propósito	Verificar que el servicio web recibe información
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON como el que se muestra en la figura 3.86
Pasos	1. En la dirección del servicio web a consumir cambiar el puerto 8000 por cualquier otro, ejemplo 8001 2. Definir archivo *JSON* acorde a las necesidades de la prueba 3. Ejecutar *script*
Resultado esperado	Se debe de obtener una respuesta que el servicio web no pudo ser alcanzado

Tabla 4.6 Caso de Prueba Consumir Servicio Web con URL en Minúsculas y Mayúsculas

Caso de Prueba 04	Consumir Servicio Web con URL en Minúsculas y Mayúsculas
Propósito	Verificar que el servicio web recibe información

Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON como el que se muestra en la figura 3.86
Pasos	1. *En la dirección del servicio web a consumir cambiar la dirección "http://104.131.86.147:8000/CodeCloud/Compilar" por minúsculas y mayúsculas, ejemplo "http://104.131.86.147:8000/coDeClOud/Compilar"* 2. Definir archivo *JSON* acorde a las necesidades de la prueba 3. Ejecutar *script*
Resultado esperado	El servicio web debe ser encontrado y brindar una respuesta dependiendo al contenido del *JSON* enviado

Los casos de prueba para verificar la integridad de la información en el servicio web son los que se muestran de la tabla 4.7 a la tabla 4.12.

Tabla 4.7 Caso de Prueba JSON Formado Erróneo Servicio Compilar

Caso de Prueba 05	JSON Erróneo Compilar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON si el nodo "URL"
Pasos	1. Definir archivo *JSON* acorde a las necesidades de la prueba 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje de error y que no pudo completar la acción compilar

Tabla 4.8 Caso de Prueba JSON Formado Erróneo para Servicio Ejecutar

Caso de Prueba 06	JSON Erróneo Ejecutar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON si el nodo "URL"

Pasos	1. Definir archivo *JSON* acorde a las necesidades de la prueba 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje de error y que no pudo completar la acción ejecutar

Tabla 4.9 Caso de Prueba JSON No Formado Compilar

Caso de Prueba 07	JSON no Formado Compilar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web
Pasos	1. Definir archivo *JSON* acorde a las necesidades de la prueba 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje de error y que no pudo completar la acción compilar

Tabla 4.10 Caso de Prueba JSON No Formado Ejecutar

Caso de Prueba 08	JSON no Formado Ejecutar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web
Pasos	1. Definir archivo *JSON* acorde a las necesidades de la prueba 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje de error y que no pudo completar la acción ejecutar

Tabla 4.11 Caso de Prueba JSON sin Usuario y Contraseña Compilar

Caso de Prueba 09	JSON sin Usuario y Contraseña Compilar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web

	2. Formar un archivo JSON si los nodos "Usuario" y "Contraseña"
Pasos	1. Definir archivo *JSON* acorde a las necesidades de la prueba 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje mencionando que el acceso fue denegado

Tabla 4.12 Caso de Prueba JSON sin Usuario y Contraseña Ejecutar

Caso de Prueba 10	JSON sin Usuario y Contraseña Ejecutar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON si los nodos "Usuario" y "Contraseña"
Pasos	1. Definir archivo *JSON* acorde a las necesidades de la prueba 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje mencionando que el acceso fue denegado

Es importante validar que el servicio es capaz de autenticar a un único usuario y que solo este puede realizar las acciones correspondientes del mismo y estos se muestran de la tabla 413 a la tabla 4.18.

Tabla 4.13 Caso de Prueba Usuario Incorrecto Compilar

Caso de Prueba 11	Usuario Incorrecto Compilar
Propósito	Verificar que el servicio web valida la autenticidad de quien lo consume
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON con un usuario incorrecto
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje mencionando que el acceso fue denegado

Tabla 4.14 Caso de Prueba Usuario Incorrecto Ejecutar

Caso de Prueba 12	Usuario Incorrecto Ejecutar
Propósito	Verificar que el servicio web valida la autenticidad de quien lo consume
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON con un usuario incorrecto
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje mencionando que el acceso fue denegado

Tabla 4.15 Caso de Prueba Contraseña Incorrecta Compilar

Caso de Prueba 13	Contraseña Incorrecta Compilar
Propósito	Verificar que el servicio web valida la autenticidad de quien lo consume
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON con una contraseña incorrecta
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje mencionando que el acceso fue denegado

Tabla 4.16 Caso de Prueba Contraseña Incorrecta Ejecutar

Caso de Prueba 14	Contraseña Incorrecta Ejecutar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 3. Formar un archivo JSON si los nodos “Usuario” y “Contraseña”
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe mostrar un mensaje mencionando que el acceso fue denegado

Tabla 4.17 Caso de Prueba Usuario y Contraseña Correctos Compilar

Caso de Prueba 15	Usuario y Contraseña Correctos Compilar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON con los campos usuario y contraseña adecuados
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe permitir la operación de compilar

Tabla 4.18 Caso de Prueba JSON sin Usuario y Contraseña Ejecutar

Caso de Prueba 16	Usuario y Contraseña Correctos Ejecutar
Propósito	Verificar que el servicio web procesa la información de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Formar un archivo JSON si los nodos “Usuario” y “Contraseña”
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe permitir la operación de ejecutar

Para probar que la compilación de un archivo se ejecute de forma adecuada es necesario probar múltiples escenarios los cuales se describen de la tabla 4.19 a la 4.22.

Tabla 4.19 Caso de Prueba Compilado Exitoso

Caso de Prueba 17	Compilado Exitoso
Propósito	Verificar que el servicio web procesa realiza la compilación de un proyecto de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Proyecto en lenguaje *Java* almacenado en *Zip* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe compilar exitosamente

Tabla 4.20 Caso de Prueba Compilado no Exitoso

Caso de Prueba 18	Compilado no Exitoso
Propósito	Verificar que el servicio web retorna el resultado de la compilación fallida
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Proyecto en lenguaje *Java* con errores almacenado en *Zip* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe notificar que la compilación no se ejecutó exitosamente

Tabla 4.21 Caso de Prueba Compilar más de 10 Archivos

Caso de Prueba 19	Proyecto con más de 10 Archivos
Propósito	Verificar el tiempo que le toma al servicio web compilar más de 10 archivos
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Proyecto en lenguaje *Java* con más de 10 archivos almacenado en *Zip* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe compilar en no más de 1 minuto

Tabla 4.22 Caso de Prueba Proyecto sin Contenido Valido Compilar

Caso de Prueba 20	Proyecto sin Contenido Valido Compilar
Propósito	Verificar el resultado del servicio web cuando no archivos son encontrados en archivo *Zip* descargado
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Archivo *Zip sin contenido almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script*

	2. Ejecutar *script*
Resultado esperado	El servicio web debe notificar que no se pudo compilar el proyecto

Para probar que la ejecución de un archivo se ejecute de forma adecuada al igual que el proceso de ejecución es necesario probar múltiples escenarios los cuales se describen de la tabla 4.23 a la 4.26.

Tabla 4.23 Caso de Prueba Ejecución Exitosa

Caso de Prueba 21	Ejecución Exitosa
Propósito	Verificar que el servicio web procesa realiza la compilación de un proyecto de forma adecuada
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Proyecto en lenguaje *Java* almacenado en *Zip* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe ejecutar el proyecto exitosamente

Tabla 4.24 Caso de Prueba Ejecución no Exitosa

Caso de Prueba 22	Ejecución no Exitosa
Propósito	Verificar que el servicio web retorna el resultado de la compilación fallida
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Proyecto en lenguaje *Java* con errores lógicos almacenado en *Zip* en la *nube* 3. Formar archivo JSON para compilar
Pasos	3. Cargar contenido del archivo JSON al *script* 4. Ejecutar *script*
Resultado esperado	El servicio web debe notificar que la ejecución generó un error

Tabla 4.25 Caso de Prueba Ejecutar Proyecto con más de 10 Archivos

Caso de Prueba 23	Ejecutar Proyecto con más de 10 Archivos
Propósito	Verificar el tiempo que le toma al servicio web compilar más de 10 archivos

Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Proyecto en lenguaje *Java* con más de 10 archivos almacenado en *Zip* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe ejecutar en no más de 3 minuto

Tabla 4.26 Caso de Prueba Proyecto sin Contenido Valido Ejecutar

Caso de Prueba 24	Proyecto Contenido no Valido Ejecutar
Propósito	Verificar el resultado del servicio web cuando no archivos java son encontrados en archivo *Zip* descargado
Prerrequisitos	1. *Script* en *Node* que permita consumir un método *POST* de un servicio web 2. Archivo *Zip sin contenido almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Ejecutar *script*
Resultado esperado	El servicio web debe notificar que no se pudo compilar el proyecto

Las tablas tabla 4.27 a la 4.30 se enfocan en probar cuál es el límite de peticiones simultaneas que el servicio web puede soportar, esto con el fin de verificar si las especificaciones físicas del servidor que almacena al servicio web son las suficientes para este proyecto o es necesario adquirir una maquina más poderosa en términos de hardware.

Tabla 4.27 Caso de Prueba 5 Conexiones Automatizadas Compilar

Caso de Prueba 25	5 Conexiones Automatizadas Compilar
Propósito	Verificar el rendimiento con 5 llamadas al servicio web
Prerrequisitos	1. *Script* en *Node* que consuma 5 veces un método *POST* de un servicio web 2. Archivo *Zip almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Medir tiempo de ejecución 3. Ejecutar *script*
Resultado esperado	El servicio web debe responder en un tiempo razonable

Tabla 4.28 Caso de Prueba 5 Conexiones Automatizadas Ejecutar

Caso de Prueba 26	5 Conexiones Automatizadas Ejecutar
Propósito	Verificar el rendimiento con 5 llamadas al servicio web
Prerrequisitos	1. *Script* en *Node* que consuma 5 veces un método *POST* de un servicio web 2. Archivo *Zip almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Medir tiempo de ejecución 3. Ejecutar *script*
Resultado esperado	El servicio web debe responder en un tiempo razonable

Tabla 4.29 Caso de Prueba 10 Conexiones Automatizadas Compilar

Caso de Prueba 27	10 Conexiones Automatizadas Compilar
Propósito	Verificar el rendimiento con 10 llamadas al servicio web
Prerrequisitos	1. *Script* en *Node* que consuma 10 veces un método *POST* de un servicio web 2. Archivo *Zip almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Medir tiempo de ejecución 3. Ejecutar *script*
Resultado esperado	El servicio web debe responder en un tiempo razonable

Tabla 4.30 Caso de Prueba 10 Conexiones Automatizadas Ejecutar

Caso de Prueba 28	10 Conexiones Automatizadas Ejecutar
Propósito	Verificar el rendimiento con 10 llamadas al servicio web
Prerrequisitos	1. *Script* en *Node* que consuma 10 veces un método *POST* de un servicio web 2. Archivo *Zip almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Medir tiempo de ejecución 3. Ejecutar *script*
Resultado esperado	El servicio web debe responder en un tiempo razonable

Tabla 4.31 Caso de Prueba 20 Conexiones Automatizadas Compilar

Caso de Prueba 29	20 Conexiones Automatizadas Compilar
Propósito	Verificar el rendimiento con 20 llamadas al servicio web
Prerrequisitos	1. *Script* en *Node* que consuma 20 veces un método *POST* de un servicio web 2. Archivo *Zip almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Medir tiempo de ejecución 3. Ejecutar *script*
Resultado esperado	El servicio web debe responder en un tiempo razonable

Tabla 4.32 Caso de Prueba 20 Conexiones Automatizadas Ejecutar

Caso de Prueba 30	20 Conexiones Automatizadas
Propósito	Verificar el rendimiento con 20 llamadas al servicio web
Prerrequisitos	1. *Script* en *Node* que consuma 20 veces un método *POST* de un servicio web 2. Archivo *Zip almacenado* en la *nube* 3. Formar archivo JSON para compilar
Pasos	1. Cargar contenido del archivo JSON al *script* 2. Medir tiempo de ejecución 3. Ejecutar *script*
Resultado esperado	El servicio web debe responder en un tiempo razonable

4.1.2 Ejecución de Pruebas

La ejecución de los *scripts* que ejecutan cada caso de prueba se ejecuta desde la línea de comandos, para la ejecución del caso de prueba 1 descrito en la tabla 4.3 se implementa el código mostrado en la figura 4.1.

```
var ping = require('ping');

var hosts = ['104.131.86.147'];

hosts.forEach(function(host){
    ping.sys.probe(host, function(isAlive){
        var msg = isAlive ? 'Ping a ' + host + ' realizada exitosamente' : 'Ping a ' + host + ' fallo';
        console.log(msg);
    });
});
```

Figura 4.1 Código Caso de Prueba Ping Servidor

La ejecución de dicha prueba es la que se muestra en la figura 4.2, como se mencionó anteriormente estas pruebas se ejecutan directamente desde la terminal.

```
CodeCloud : Accesibilidad$node pruebaPing.js
Ping a 104.131.86.147 realizada exitosamente
CodeCloud : Accesibilidad$
```

Figura 4.2 Ejecución Caso de Prueba Ping Servidor

Por el resultado de la prueba se puede observar que el servidor está disponible para recibir llamadas desde el cliente. Para el caso de prueba mostrado en la tabla 4.4 es implementado el código mostrado en la figura 4.3

```
var fs = require('fs');
var http = require('http');
var request = require('request');

var json = JSON.parse(fs.readFileSync('/Users/antonio/OneDrive/Tesis/Capitulo 4/Scripts Test/JSON Files/JSONExitoso.json'));

var headers =  {
    'Content-Type': 'application/json; charset=utf-8',
  }

var options = {
  url: 'http://104.131.86.147:8000/CodeCloud/Compilar',
  header: headers,
  method: 'POST',
  form: json
};

request(options, function (error, response, body) {
  console.log('\n\n');
    if (!error) {
        // Print out the response body
        console.log(body)
    }else{
      console.log(error)
    }
    console.log('\n\n');
})
```

Figura 4.3 Código Caso de Prueba Consumir Servicio Web

Y la ejecución de la pertinente prueba es la que se muestra en la figura 4.4; como se puede observar el resultado del proceso fue exitoso, cabe mencionar que el resultado de compilación exitosa no es relevante para esta prueba, sino únicamente que el servidor procesa el archivo *JSON* enviado de forma adecuada.

```
CodeCloud : Accesibilidad$node pruebaJsonExitoso.js


[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
```

Figura 4.4 Ejecución Caso de Prueba Consumir Servicio Web

Para el caso de las pruebas siguientes se omitirá el contenido del código ya que la implementación de dichas pruebas es similar al mostrado en la figura 4.3; para la prueba mencionada en la tabla 4.5 el puerto utilizado para realizar la llamada del servicio web es el 8001, por lo que la conexión no debe poder ser establecida como se muestra en la figura 4.5.

```
[Antonio : Accesibilidad$node pruebaPuertoErroneo.js


{ Error: connect ECONNREFUSED 104.131.86.147:8001
    at Object.exports._errnoException (util.js:1014:11)
    at exports._exceptionWithHostPort (util.js:1037:20)
    at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1138:14)
  code: 'ECONNREFUSED',
  errno: 'ECONNREFUSED',
  syscall: 'connect',
  address: '104.131.86.147',
  port: 8001 }
```

Figura 4.5 Ejecución Caso de Prueba Puerto Erróneo

En el caso de la prueba descrita en la tabla 4.6 la *url* destino del servicio web es la que cambia, sin importar que la dirección no coincida con la descrita por la implementación, respecto a mayúsculas y minúsculas el servicio web debería procesar la información de forma correcta, la cadena implementada para esta prueba es la siguiente "http://104.131.86.147:8000/CoDeCloUd/ComPiLar" mientras que la ejecución de la misma es la mostrada en la figura 4.6.

```
CodeCloud : Accesibilidad$node pruebaMayusYMinus.js


[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
```

Figura 4.6 Ejecución Caso de Prueba URL en Minúsculas y Mayúsculas

El quinto caso de prueba presente en la tabla 4.7 consiste en que el archivo JSON no cuente con todas las especificaciones necesarias para la ejecución del servicio web de forma adecuada, por lo cual es enviado sin el nodo "URL", el resultado debe ser que la acción compilar falló, como se muestra en la figura 4.7.

```
[CodeCloud : Integridad$node pruebaNoUrl.js


[{"Accion":"Compilar","Proceso":1,"Resultado":{"killed":false,"code":2,"signal":null,"cmd":"/bin/sh -c cd /root/CodeCloud/Codigo/ && curl --ipv4 -XGET -o test-
proyect.zip undefined"}}]
```

Figura 4.7 Ejecución Caso de Prueba JSON Erróneo Compilar

Para la prueba número 6 el procedimiento es el mismo, solo que en este caso es para el servicio ejecutar, el resultado de dicha prueba se muestra en la figura 4.8.

```
CodeCloud : Integridad$node pruebaNoUrlEjecutar.js

[{"Accion":"Compilar","Proceso":1,"Resultado":{"killed":false,"code":6,"signal":null,"cmd":"/bin/sh -c cd /root/CodeCloud/Codigo/ && curl --ipv4 -XGET -o test-proyect.zip undefined"}}]
```

Figura 4.8 Ejecución Caso de Prueba JSON Erróneo Ejecutar

En la tabla 4.9 la prueba consiste en enviar un archivo *JSON* sin contenido alguno y el servicio web Compilar debe responder adecuadamente y no detener su ejecución debido a la corrupción en el archivo enviado, la ejecución de esta prueba es la que se muestra en la figura 4.9

```
CodeCloud : Integridad$node pruebaNoJSONCompilar.js


[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.9 Ejecución Caso de Prueba JSON No Formado Compilar

El resultado de la prueba terminó con un acceso denegado debido a que el archivo JSON no contenía la información referente al usuario y contraseña que da acceso a dichos servicios, es por eso que la prueba que se menciona en la tabla 4.10 al realizar la ejecución de la misma se obtiene el mismo resultado, como se muestra en la figura 4.10.

```
CodeCloud : Integridad$node pruebaNoJSONEjecutar.js


[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.10 Ejecución Caso de Prueba JSON No Formado Ejecutar

El caso de prueba mostrado en la tabla 4.11 consiste en consumir el servicio web sin el nodo de usuario y contraseña, el resultado debe ser el mismo que las pruebas anteriores, la ejecución de dicha prueba corresponde al servicio web Compilar y es el que se muestra en la figura 4.11.

```
CodeCloud : Integridad$node pruebaNoCredencialesCompilar.js


[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.11 Ejecución Caso de Prueba JSON sin Usuario y Contraseña Compilar

La figura 4.12 muestra la ejecución de la prueba descrita en la tabla 4.12 que consiste en lo mismo que la prueba anterior, sin embargo, esta es la correspondiente del servicio Ejecutar.

```
CodeCloud : Integridad$node pruebaNoCredencialesEjecutar.js

[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.12 Ejecución Caso de Prueba JSON sin Usuario y Contraseña Ejecutar

La prueba descrita en la tabla 4.13 valida que el servicio web Compilar autentique de forma adecuada a quien consume dicho servicio, para este caso en particular el Usuario es incorrecto, la ejecución de esta prueba es la que se muestra en la figura 4.13.

```
CodeCloud : Credenciales$node pruebaUsuarioIncorrectoCompilar.js

[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.13 Ejecución Caso de Prueba Usuario Incorrecto Compilar

De igual manera la prueba en la tabla 4.14 realiza la misma validación, pero para el servicio web Ejecutar, la ejecución de esta prueba se puede ver en la figura 4.14.

```
CodeCloud : Credenciales$node pruebaUsuarioIncorrectoEjecutar.js

[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.14 Ejecución Caso de Prueba Usuario Incorrecto Ejecutar

En la figura 4.15 se observa la ejecución de la prueba que consiste en que la contraseña es incorrecta para el servicio Compilar, esta prueba la podemos encontrar descrita en la tabla 4.15.

```
CodeCloud : Credenciales$node pruebaAccesoIncorrectoCompilar.js

[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.15 Ejecución Caso de Prueba Contraseña Incorrecta Compilar

En la figura 4.16 se muestra la ejecución de la prueba descrita en la tabla 4.16 que es la correspondiente para el servicio Ejecutar.

```
CodeCloud : Credenciales$node pruebaAccesoIncorrectoEjecutar.js


[{"Accion":"Compilar","Proceso":"Error","Resultado":"Acceso Denegado"}]
```

Figura 4.16 Ejecución Caso de Prueba Contraseña Incorrecta Ejecutar

Una vez probados los escenarios en donde las credenciales no son las correctas es necesario validar que cuando dichas credenciales son las adecuadas el servicio web permite el acceso y realiza la ejecución pertinente, en la figura 4.17 se pude observar el resultado de la prueba descrita en la tabla 4.17 para el servicio Compilar.

```
CodeCloud : Credenciales$node pruebaDatosCorrectosCompilar.js


[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
```

Figura 4.17 Ejecución Caso de Prueba Usuario y Contraseña Correctos Compilar

De nuevo el resultado de la compilación no es de importancia para el fin de esta prueba, en la figura 4.18 al igual que la prueba anterior el resultado de la validación debe ser exitoso solo que este se enfoca al servicio Ejecutar.

```
CodeCloud : Credenciales$node pruebaDatosCorrectosEjecutar.js


[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Human greeting!\n"}]
```

Figura 4.18 Ejecución Caso de Prueba Usuario y Contraseña Correctos Ejecutar

Una vez concluidas las pruebas de accesibilidad, integridad y credenciales, es necesario probar la compilación de los proyectos en el servicio Compilar, por lo que en la figura 4.19 se muestra la ejecución del caso de prueba descrito en la tabla 4.19.

```
CodeCloud : Compilar$node pruebaCompilarExitoso.js


[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
```

Figura 4.19 Ejecución Caso de Prueba Compilado Exitoso

El servicio web debe manejar adecuadamente la información y anunciar al cliente cuando la compilación no pudo realizarse, esto ya sea por un error de sintaxis o semántico ocurrió en la compilación del proyecto, ese es el fin de la prueba especificada en la tabla 4.10 y su ejecución se puede observar en la figura 4.20. El error en el código enviado al servicio web se encuentra en la ausencia de un punto y coma.

```
CodeCloud : Compilar$node pruebaCompilarNoExitoso.js

[{"Accion":"Compilar","Proceso":1,"Resultado":"Command failed: cd Codigo/CompilarErrorc6s3o/ && javac *.java\nPruebaError.java:8: error: ';' expected\n
System.out.println(\"Constructor\")\n                                        ^\n1 error\n"}]
```

Figura 4.20 Ejecución Caso de Prueba Compilado no Exitoso

A su vez es necesario probar que el proceso de compilación se ejecuta adecuadamente cuando el número de archivos a compilar es mayor a 2 o 3 archivos que es con los que se ha venido probando dicho servicio, es por eso que en la tabla 4.21 se establece una compilación con 10 archivos y que el resultado de este es exitoso como se muestra en la figura 4.21.

```
CodeCloud : Compilar$node pruebaCompilarMultiplesArchivos.js

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
```

Figura 4.21 Ejecución Caso de Prueba Compilar más de 10 Archivos

Para finalizar es necesario verificar el comportamiento del servicio Compilar cuando el archivo a compilar no contiene código, ver tabla 4.22, en este caso en particular que contiene un archivo java con nada más que un comentario en dicho archivo, cabe destacar que hay validaciones a nivel web que impiden realizar este tipo de acciones pero de igual manera el servicio Compilar debe reaccionar adecuadamente en caso de que este escenario llegase a presentarse, la ejecución de esta prueba es la que se muestra a continuación.

```
CodeCloud : Compilar$node pruebaCompilarProyectoVacio.js

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
```

Figura 4.22 Ejecución Caso de Prueba Proyecto Solo con Comentarios Compilar

Esto concluye con la fase de pruebas del servicio Compilar, ahora es necesario probar el servicio Ejecutar, como se describe en la prueba (tabla 4.23) el resultado de la ejecución de un proyecto debe ser retornado al cliente, la ejecución de esta prueba se muestra en la figura 4.23.

```
CodeCloud : Ejecutar$node pruebaEjecutarExitoso.js

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Human greeting!\n"}]
```

Figura 4.23 Ejecución Caso de Prueba Ejecución Exitosa

En la tabla 4.24 se describe la prueba que consiste en la ejecución fallida de un proyecto, esto quiere decir que tiene que compilar, pero debe fallar en la ejecución, esto se logra con un arreglo fuera de índice y su ejecución es la que se muestra a continuación.

```
CodeCloud : Ejecutar$node pruebaEjecutarError.js

[{"Accion":"Compilar","Proceso":"Error","Resultado":{"killed":false,"code":1,"signal":null,"cmd":"/bin/sh -c cd Codigo/EjecutarErroruo0tp/ && java Main"}}]
```

Figura 4.24 Ejecución Fallida Caso de Prueba Ejecución no Exitosa

En esta ocasión la prueba no se ejecutó de forma adecuada, ya que el servicio web estaba regresando un resultado diferente al esperado, por lo que indagando en el código se detectó una falla en el servicio web por lo que el código al servicio Ejecutar en específico la función "hiloEjecutar" tuvo que ser actualizado a como se muestra en la figura 4.25.

```
var hiloEjecutar = function(jsonInfo, respuesta, callback){
    var numeroArchivos = parseInt(jsonInfo.body.NumeroArchivos, 10);
    console.log('Inicia Proceso Ejecutar');

    for (var i = 0; i < numeroArchivos; i++) {
        if(jsonInfo.body.Archivos[i]['Main'] ==  "True"){
            var comando = 'cd Codigo/' + jsonInfo.body.NombreProyecto + compilarCodigo.randomString + '/ && java ' +
            jsonInfo.body.Archivos[i]['NombreArchivo'];

            try{
                var child = execSync(comando);
                respuesta[0].Resultado = child;
                respuesta[0].Proceso = "Ejecutado";
                callback(respuesta);
            }catch(err){
                respuesta[0].Resultado = err.message;
                respuesta[0].Proceso = 1;
            }finally{
                callback(respuesta);
            }
        }
    }
}
```

Figura 4.25 Actualización Función hiloEjecutar

Básicamente en la actualización se cambió la forma en la que se ejecuta el hilo la acción de ejecutar el archivo java, esto debido a que el hilo que se encargaba de esto no esperaba a que la ejecución terminase y enviaba la respuesta al cliente, es por esto que el mensaje de error no era el adecuado, al cambiarlo a "execSync" básicamente espera a que la ejecución del hilo termine y después envía la respuesta ya que realiza el proceso de forma síncrona. También se tuvo que cambiar la librería que se estaba empleando, como se muestra en la figura 4.26.

```
var execSync = require('child_process').execSync;
```

Figura 4.26 Actualización Librería ExecSync

Al ejecutar nuevamente la prueba el resultado es el adecuado como el que se muestra en la figura 4.27.

```
CodeCloud : Ejecutar$node pruebaEjecutarError.js

[{"Accion":"Compilar","Proceso":1,"Resultado":"Command failed: cd Codigo/EjecutarError6cie9/ && java Main\nException in thread \"main\" java.lang.ArrayIndexOut
OfBoundsException: 20\n\tat Main.main(Main.java:13)\n"}]
```

Figura 4.27 Ejecución Caso de Prueba Ejecución no Exitosa

De igual manera que el servicio Compilar se debe probar al servicio Ejecutar con un volumen de archivos mayor a los establecidos por las pruebas anteriores, como se describe en la tabla 4.25, la ejecución de dicha prueba es la que se muestra a continuación.

```
CodeCloud : Ejecutar$node pruebaEjecutarDiezArchivos.js

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":{"type":"Buffer","data":[10,68,101,108,32,98,105,103,32,98,97,110,103,32,115,117,114,103,105,111,32,101
,108,32,85,110,105,118,101,114,115,111,10,10,69,108,32,117,110,105,118,101,114,115,111,32,99,111,110,116,105,101,110,101,32,71,97,108,97,120,105,97,115,10,10,6
9,110,32,117,110,97,32,103,97,108,97,120,105,97,32,101,120,105,115,116,101,110,32,109,117,99,104,111,115,32,115,105,115,116,101,109,97,115,32,115,111,108,97,11
4,101,115,10,10,76,111,115,32,112,108,97,110,101,116,97,115,32,99,111,110,99,105,100,111,115,32,100,101,108,32,115,105,115,116,101,109,97,32,115,111,108,97,114
,32,115,111,110,10,77,101,114,99,117,114,105,111,10,86,101,110,117,115,10,84,105,101,114,114,97,10,77,97,114,116,101,10,74,117,112,105,116,101,114,10,83,97,116
,117,114,110,111,10,85,114,97,110,111,10,80,108,117,116,111,110,10,10,10,69,110,32,101,108,32,112,108,97,110,101,116,97,32,116,105,101,114,114,97,32,104,97,98,
105,116,97,110,58,10,83,101,114,101,115,32,86,105,118,111,115,10,10,10,76,111,115,32,104,117,109,97,110,111,115,32,104,97,98,105,116,97,110,32,99,111,110,32,11
2,108,97,110,116,97,115,32,121,32,97,110,105,109,97,108,101,115,10,65,108,103,117,110,97,115,32,112,108,97,110,116,97,115,32,115,111,110,58,10,10,65,114,98,111
,108,10,65,114,98,117,115,116,111,10,72,105,101,114,98,97,115,10,77,97,116,97,115,10,10,76,97,115,32,112,108,97,110,116,97,115,32,110,111,32,100,117,101,114,10
9,101,110,10,65,108,103,117,110,111,115,32,97,110,105,109,97,108,101,115,32,115,111,110,58,10,10,80,101,114,114,111,10,76,111,98,111,10,76,101,111,110,10,69,10
8,101,102,97,110,116,101,10,71,97,116,111,10,84,105,103,114,101,10,90,111,114,114,111,10,82,97,116,111,110,10,10,76,111,115,32,97,110,105,109,97,108,101,115,32
,100,117,101,114,109,101,110,32,101,110,116,114,101,32,51,32,97,32,50,50,32,104,114,115,46,10,10,76,111,115,32,104,117,109,97,110,111,115,32,100,117,101,114,10
9,101,110,32,56,32,104,114,115,46,10]}}]
```

Figura 4.28 Ejecución Fallida Caso de Prueba Ejecutar más de 10 Archivos

Como se puede observar en la figura 4.28 la respuesta no es la esperada, ya que la respuesta se está enviando en bytes, para solucionar este conflicto se tuvo que modificar una vez más la función “hiloEjecutar”, haciendo uso de la función “toString” fue más que suficiente para solucionar este inconveniente, el código fue actualizado como se muestra en la figura 4.29.

```
var hiloEjecutar = function(jsonInfo, respuesta, callback){
    var numeroArchivos = parseInt(jsonInfo.body.NumeroArchivos, 10);
    console.log('Inicia Proceso Ejecutar');

    for (var i = 0; i < numeroArchivos; i++) {
        if(jsonInfo.body.Archivos[i]['Main'] ==  "True"){
            var comando = 'cd Codigo/' + jsonInfo.body.NombreProyecto + compilarCodigo.randomString + '/ && java ' +
            jsonInfo.body.Archivos[i]['NombreArchivo'];
            try{
                var child = execSync(comando);
                respuesta[0].Resultado = child.toString('utf8');
                respuesta[0].Proceso = "Ejecutado";
                callback(respuesta);
            }catch(err){
                respuesta[0].Resultado = err.message;
                respuesta[0].Proceso = 1;
                callback(respuesta);
            }
        }
    }
}
```

Figura 4.29 Corrección Error Respuesta Inadecuada Ejecutar

Después de la actualización de la función “hiloEjecutar” la ejecución de la prueba presente en la tabla 4.25 da como resultado el que se muestra en la figura 4.30.

```
CodeCloud : Ejecutar$node pruebaEjecutarDiezArchivos.js

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"\nDel big bang surgio el Universo\n\nEl universo contiene Galaxias\n\nEn una galaxia existen muchos si
stemas solares\n\nLos planetas concidos del sistema solar son\nMercurio\nVenus\nTierra\nMarte\nJupiter\nSaturno\nUrano\nPluton\n\n\nEn el planeta tierra habita
n:\nSeres Vivos\n\n\nLos humanos habitan con plantas y animales\nAlgunas plantas son:\n\nArbol\nArbusto\nHierbas\nMatas\n\nLas plantas no duermen\nAlgunos anim
ales son:\n\nPerro\nLobo\nLeon\nElefante\nGato\nTigre\nZorro\nRaton\n\nLos animales duermen entre 3 a 22 hrs.\n\nLos humanos duermen 8 hrs.\n"}]
```

Figura 4.30 Ejecución Caso de Prueba Ejecutar más de 10 Archivos

Las siguientes pruebas consisten en probar la capacidad del servicio web para compilar archivos y ejecutarlos de forma concurrente, es por eso que se definen las pruebas de Capacidad, en la tabla 4.26 la prueba menciona la ejecución de 5 llamadas al servicio Compilar y su ejecución es la que se muestra en la figura 4.31.

```
CodeCloud : Capacidad$node pruebaCompilar5Veces.js
Ejecutando llamada: 5

[{"Accion":"Compilar","Proceso":1,"Resultado":{"killed":false,"code":9,"signal":null,"cmd":"/bin/sh -c cd /root/CodeCloud/Codigo/ && unzip Blanco"}}]
Ejecutando llamada: 2

[{"Accion":"Compilar","Proceso":1,"Resultado":{"killed":false,"code":9,"signal":null,"cmd":"/bin/sh -c cd /root/CodeCloud/Codigo/ && unzip MultiplesArchivos"}}
]
Ejecutando llamada: 4

[{"Accion":"Compilar","Proceso":1,"Resultado":{"killed":false,"code":9,"signal":null,"cmd":"/bin/sh -c cd /root/CodeCloud/Codigo/ && unzip CompilarError"}}]
Ejecutando llamada: 1

[{"Accion":"Compilar","Proceso":1,"Resultado":{"killed":false,"code":9,"signal":null,"cmd":"/bin/sh -c cd /root/CodeCloud/Codigo/ && unzip test-proyect"}}]
Ejecutando llamada: 3

[{"Accion":"Compilar","Proceso":1,"Resultado":{"killed":false,"code":9,"signal":null,"cmd":"/bin/sh -c cd /root/CodeCloud/Codigo/ && unzip EjecutarError"}}]
Tiempo de ejecucion: 1.729 s
```

Figura 4.31 Ejecución Fallida Caso de Prueba 5 Conexiones Automatizadas Compilar

El resultado de dicha prueba no fue el esperado ya que ninguna de las llamadas se ejecutó como se esperaba, analizando a fondo el código se llegó a la conclusión de que el servicio trataba de compilar los archivos antes de que estos fuesen descargados, es por eso que fue necesario actualizar la función “descargarProyecto” en el archivo “compilarCodigo.js” para que esta acción fuera síncrona y una vez terminada el servicio sea capaz de continuar de forma adecuada, dicha actualización queda plasmada como se muestra en la figura 4.32.

```
// Descargar Proyecto
var descargarProyecto = function(url, nombreArchivo, callback) {

    if (!fs.existsSync(ubicacionDescarga)){
        fs.mkdirSync(ubicacionDescarga);
    }else{
        eliminarCarpeta();
    }

    try{
        var child = execSync('cd ' +  ubicacionDescarga + ' && curl --ipv4 -XGET -o ' + nombreArchivo + '.zip ' + url);
        respuesta[0].Resultado = "Descarga exitosa";
        respuesta[0].Proceso = "Ejecutado";
        callback(respuesta);
    }catch(err){
        respuesta[0].Resultado = error;
        respuesta[0].Proceso = 1;
        callback(respuesta);
    }
};
```

Figura 4.32 Corrección Error Descarga Asíncrona

Una vez ejecutados los cambios en dicha función al ejecutar una vez más la prueba descrita en la tabla 4.26 su resultado es el que se muestra a continuación.

```
CodeCloud : Capacidad$node pruebaCompilar5Veces.js
Ejecutando llamada: 5

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
Ejecutando llamada: 4

[{"Accion":"Compilar","Proceso":1,"Resultado":"Command failed: cd Codigo/CompilarErrorjk9uy/ && javac *.java\nPruebaError.java:8: error: ';' expected\n
System.out.println(\"Constructor\")\n                                                ^\n1 error\n"}]
Ejecutando llamada: 3

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
Ejecutando llamada: 1

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
Ejecutando llamada: 2

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Compilado exitosamente"}]
Tiempo de ejecucion: 8.214 s
```

Figura 4.33 Ejecución Caso de Prueba 5 Conexiones Automatizadas Compilar

De igual forma se debe probar el servicio Ejecutar (tabla 4.27) y medir los tiempos de ejecución de dicho servicio, el resultado de dicha ejecución es el que se muestra en la figura 4.34.


```
CodeCloud : Capacidad$node pruebaEjecutar5Veces.js
Ejecutando llamada: 2

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"\nDel big bang surgio el Universo\n\nEl universo contiene Galaxias\n\nEn una galaxia existen muchos si
stemas solares\n\nLos planetas concidos del sistema solar son\nMercurio\nVenus\nTierra\nMarte\nJupiter\nSaturno\nUrano\nPluton\n\n\nEn el planeta tierra habita
n:\nSeres Vivos\n\n\nLos humanos habitan con plantas y animales\nAlgunas plantas son:\n\nArbol\nArbusto\nHierbas\nMatas\n\nLas plantas no duermen\nAlgunos anim
ales son:\n\nPerro\nLobo\nLeon\nElefante\nGato\nTigre\nZorro\nRaton\n\nLos animales duermen entre 3 a 22 hrs.\n\nLos humanos duermen 8 hrs.\n"}]
Ejecutando llamada: 3

[{"Accion":"Compilar","Proceso":1,"Resultado":"Command failed: cd Codigo/EjecutarErrorbb2km/EjecutarError/ && java Main\nException in thread \"main\" java.lang
.ArrayIndexOutOfBoundsException: 20\n\tat Main.main(Main.java:13)\n"}]
Ejecutando llamada: 4

[{"Accion":"Compilar","Proceso":1,"Resultado":"Command failed: cd Codigo/CompilarErrorrna6e/CompilarError/ && javac *.java\nPruebaError.java:8: error: ';' expe
cted\n        System.out.println(\"Constructor\")\n                                                ^\n1 error\n"}]
Ejecutando llamada: 1

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"Human greeting!\n"}]
Ejecutando llamada: 5

[{"Accion":"Compilar","Proceso":"Ejecutado","Resultado":"\nDel big bang surgio el Universo\n\nEl universo contiene Galaxias\n\nEn una galaxia existen muchos si
stemas solares\n\nLos planetas concidos del sistema solar son\nMercurio\nVenus\nTierra\nMarte\nJupiter\nSaturno\nUrano\nPluton\n\n\nEn el planeta tierra habita
n:\nSeres Vivos\n\n\nLos humanos habitan con plantas y animales\nAlgunas plantas son:\n\nArbol\nArbusto\nHierbas\nMatas\n\nLas plantas no duermen\nAlgunos anim
ales son:\n\nPerro\nLobo\nLeon\nElefante\nGato\nTigre\nZorro\nRaton\n\nLos animales duermen entre 3 a 22 hrs.\n\nLos humanos duermen 8 hrs.\n"}]
Tiempo de ejecucion: 11.238 s
```


Figura 4.34 Ejecución Caso de Prueba 5 Conexiones Automatizadas Ejecutar

En la tabla 4.28 se describe el caso de prueba que consiste en llamar al servicio Compilar 10 veces y calcular su tiempo de ejecución, el resultado de esta prueba es el que se muestra a continuación. Para fines de las pruebas mostradas en las figuras 4.35 a 4.38 el resultado de la compilación y ejecución de cada proyecto no se muestra en la ejecución de las mismas para así enfocarse únicamente en el tiempo que le toma al servidor realizar estas acciones.


```
CodeCloud : Capacidad$node pruebaCompilar10Veces.js
Ejecutando llamada: 7

Ejecutando llamada: 6

Ejecutando llamada: 10

Ejecutando llamada: 3

Ejecutando llamada: 4

Ejecutando llamada: 5

Ejecutando llamada: 8

Ejecutando llamada: 1

Ejecutando llamada: 2

Ejecutando llamada: 9

Tiempo de ejecucion: 18.028 s
```


Figura 4.35 Ejecución Caso de Prueba 10 Conexiones Automatizadas Compilar

En la figura 4.36 se muestra la ejecución el caso de prueba correspondiente en la tabla 4.29 que consiste en llamar al servicio Ejecutar 10 veces.


```
CodeCloud : Capacidad$node pruebaEjecutar10Veces.js
```

```
Ejecutando llamada: 5

Ejecutando llamada: 8

Ejecutando llamada: 9

Ejecutando llamada: 4

Ejecutando llamada: 6

Ejecutando llamada: 10

Ejecutando llamada: 3

Ejecutando llamada: 1

Ejecutando llamada: 2

Ejecutando llamada: 7

Tiempo de ejecucion: 16.503 s
```

Figura 4.36 Ejecución Caso de Prueba 10 Conexiones Automatizadas Ejecutar

La tabla 4.30 contiene la última prueba de capacidad correspondiente al servicio Compilar que consiste en llamar 20 veces al servicio web y verificar el tiempo que le toma compilar dicha solución, el resultado es el que se muestra en la figura 4.37.

```
[CodeCloud : Capacidad$node pruebaCompilar20Veces.js
Ejecutando llamada: 7

Ejecutando llamada: 5

Ejecutando llamada: 6

Ejecutando llamada: 1

Ejecutando llamada: 4

Ejecutando llamada: 2

Ejecutando llamada: 14

Ejecutando llamada: 10

Ejecutando llamada: 12

Ejecutando llamada: 8

Ejecutando llamada: 11

Ejecutando llamada: 18

Ejecutando llamada: 13

Ejecutando llamada: 17

Ejecutando llamada: 9

Ejecutando llamada: 19

Ejecutando llamada: 16

Ejecutando llamada: 15

Ejecutando llamada: 20

Ejecutando llamada: 3

Tiempo de ejecucion: 37.418 s
```

Figura 4.37 Ejecución Caso de Prueba 20 Conexiones Automatizadas Compilar

Y para concluir con las pruebas automatizadas tenemos el caso de prueba en la tabla 4.31 correspondiente al servicio Ejecutar con 20 llamadas concurrentes, el resultado de dicha prueba es el mostrado en la figura 4.38.

```
CodeCloud : Capacidad$node pruebaEjecutar20Veces.js
Ejecutando llamada: 5

Ejecutando llamada: 8

Ejecutando llamada: 9

Ejecutando llamada: 10

Ejecutando llamada: 3

Ejecutando llamada: 1

Ejecutando llamada: 2

Ejecutando llamada: 20

Ejecutando llamada: 19

Ejecutando llamada: 17

Ejecutando llamada: 15

Ejecutando llamada: 16

Ejecutando llamada: 13

Ejecutando llamada: 6

Ejecutando llamada: 7

Ejecutando llamada: 12

Ejecutando llamada: 11

Ejecutando llamada: 14

Ejecutando llamada: 4

Ejecutando llamada: 18

Tiempo de ejecucion: 38.655 s
```

Figura 4.38 Ejecución Caso de Prueba 20 Conexiones Automatizadas Ejecutar

Con esto se concluye la ejecución de las pruebas automatizadas para el servicio web, en el siguiente subcapítulo se implementan las pruebas manuales para la página web y en el subcapítulo Análisis de Resultados de Pruebas se explicará el motivo de las pruebas y que se concluye del resultado de las mismas.

4.2 Pruebas Manuales

Las pruebas manuales se enfocan en la realizar acciones directamente en la aplicación web, a través de la implementación de pruebas funcionales, que se enfocan primordialmente en verificar los componentes primordiales de la página anteriormente mencionada, dichas pruebas se dividen en categorías como se muestra en la tabla 4.33.

Tabla 4.33 Principales Características a Probar

Característica	Detalle
Inicio de Sesión	Verificar la funcionalidad del control de accesos a la plataforma web
Proyectos	Verificar las diferentes funcionalidades y características que poseen los proyectos en la plataforma
Barra de Herramientas	Verificar el correcto comportamiento de las diferentes opciones mostradas en la barra de herramientas de la plataforma (figura 4.39)
Compilar	Validar las condiciones necesarias para realizar la compilación de un proyecto y el correcto comportamiento de la compilación del mismo
Ejecutar	Validar las condiciones necesarias para realizar la ejecución de un proyecto y el correcto comportamiento de la ejecución del mismo.

4.2.1 Casos de Pruebas

Es necesario validar acciones referentes al inicio de sesión tales como registro y validación de usuarios en la plataforma web, es sumamente importante un buen control de cuentas de usuario dentro de la plataforma, ya que estas son las que se ligan de forma directa con los proyectos creados por los usuarios.

Tabla 4.34 Caso de Prueba Registrar Usuario

Caso de Prueba 30	Registrar Usuario
Propósito	Verificar el correcto registro de un usuario
Prerrequisitos	1. Navegador de Internet
Pasos	1. Ir a la url http://34.205.17.240/login 2. Presionar botón “Registrarse” 3. Llenar campos de correo, nombre y contraseñas 4. Presionar botón “Registrarse”
Resultado esperado	El usuario debe ser registrado exitosamente

Tabla 4.35 Caso de Prueba Registrar Usuario Invalido

Caso de Prueba 31	Registrar Usuario Invalido
Propósito	Verificar la validación en la integridad de los datos de registro
Prerrequisitos	1. Navegador de Internet
Pasos	1. Ir a la url http://34.205.17.240/login 2. Presionar botón “Registrarse” 3. Llenar campos de correo, nombre y contraseña (usar un correo de un usuario existente) 4. Presionar botón “Registrarse”
Resultado esperado	El usuario no debe ser registrado en la plataforma

Tabla 4.36 Caso de Prueba Registrar Usuario Campos Faltantes

Caso de Prueba 32	Registrar Usuario Campos Faltantes
Propósito	Verificar la validación en la integridad de los datos de registro
Prerrequisitos	1. Navegador de Internet
Pasos	1. Ir a la url http://34.205.17.240/login 2. Presionar botón “Registrarse” 3. Llenar campos de correo y nombre 4. El campo contraseña no debe ser llenado 5. Presionar botón “Registrarse”
Resultado esperado	El usuario no debe ser registrado en la plataforma

Tabla 4.37 Caso de Prueba Registrar Usuario Confirmación Contraseña Restante

Caso de Prueba 33	Registrar Usuario Confirmación Contraseña Restante
Propósito	Verificar la validación en la integridad de los datos de registro
Prerrequisitos	Navegador de Internet
Pasos	1. Ir a la url http://34.205.17.240/login 2. Presionar botón “Registrarse” 3. Llenar campos de correo, nombre y contraseña 4. El campo confirmar no debe ser llenado 5. Presionar botón “Registrarse”
Resultado esperado	El usuario no debe ser registrado en la plataforma

Tabla 4.38 Caso de Prueba Registrar Usuario Correo no Valido

Caso de Prueba 34	Registrar Usuario Correo no Valido
Propósito	Verificar la validación en la integridad de los datos de registro
Prerrequisitos	1. Navegador de Internet

Pasos	1. Ir a la url http://34.205.17.240/login 2. Presionar botón “Registrarse” 3. Llenar campo de correo con texto “hola” 4. Llenar campos nombre y contraseña con caracteres especiales tales como 5. El campo confirmar no debe ser llenado 6. Presionar botón “Registrarse”
Resultado esperado	Mensaje de correo no valido

Tabla 4.39 Caso de Prueba Iniciar Sesión

Caso de Prueba 35	Inicio de Sesión
Propósito	Verificar el correcto inicio de sesión de un usuario
Prerrequisitos	1. Navegador de Internet 2. Contar con una cuenta registrada
Pasos	1. Ir a la url http://34.205.17.240/login 2. Llenar campo de usuario y contraseña 3. Presionar botón “Iniciar Sesión”
Resultado esperado	El usuario debe ser iniciar sesión exitosamente

Tabla 4.40 Caso de Prueba Inicio de Sesión Inválido

Caso de Prueba 36	Inicio de Sesión Inválido
Propósito	Verificar el correcto registro de un usuario
Prerrequisitos	Navegador de Internet
Pasos	1. Ir a la url http://34.205.17.240/login 2. Llenar campo de usuario y contraseña con datos erróneos 3. Presionar botón “Iniciar Sesión”
Resultado esperado	El inicio de sesión debe ser rechazado por la plataforma

Tabla 4.41 Caso de Prueba Inicio Sesión Caracteres Especiales

Caso de Prueba 37	Inicio Sesión Caracteres Especiales
Propósito	Verificar el correcto registro de un usuario
Prerrequisitos	1. Navegador de Internet
Pasos	1. Ir a la url http://34.205.17.240/login 2. Llenar campo usuario con caracteres especiales 3. Llenar campos de contraseña 4. Presionar botón “Iniciar Sesión”

Resultado esperado	El inicio de sesión debe ser rechazado por la plataforma

Tabla 4.42 Caso de Prueba Recuperar Contraseña

Caso de Prueba 38	Recuperar Contraseña
Propósito	Verificar el correcto registro de un usuario
Prerrequisitos	1. Navegador de Internet 2. Cuenta registrada en plataforma
Pasos	1. Ir a la url http://34.205.17.240/login 2. Presionar botón “Recuperar Contraseña” 3. Llenar campo de correo 4. Presionar botón “Restablecer Contraseña”
Resultado esperado	Utilizar el correo enviado por la plataforma web y cambiar la contraseña del usuario de forma adecuada

Tabla 4.43 Caso de Prueba Cerrar Sesión

Caso de Prueba 39	Cerrar Sesión
Propósito	Verificar el correcto registro de un usuario
Prerrequisitos	1. Navegador de Internet 2. Haber iniciado sesión en la plataforma
Pasos	1. Ir a la url http://34.205.17.240/consola 2. Presionar botón “Perfil” 3. Presionar botón “Cerrar Sesión”
Resultado esperado	Cerrar Sesión del usuario y regresar a la página de Inicio de Sesión

Para verificar el correcto funcionamiento de la funcionalidad que involucran los proyectos, tales como creación, edición, adición de archivos entre otras. Dichas pruebas se describen a continuación.

Tabla 4.44 Caso de Prueba Creación Proyecto

Caso de Prueba 40	Creación Proyecto
Propósito	Verificar la correcta creación de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Crear un proyecto con el nombre “Prueba” 4. Agregar una descripción del proyecto

Resultado esperado	El proyecto debe ser creado exitosamente

Tabla 4.45 Caso de Prueba Abrir Proyecto

Caso de Prueba 41	Abrir Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto”
Resultado esperado	El proyecto debe abrirse de forma exitosa

Tabla 4.46 Caso de Prueba Agregar Archivos al Proyecto

Caso de Prueba 42	Agregar Archivos al Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Seleccionar botón “Agregar Archivo” 6. Guardar cambios
Resultado esperado	Un archivo debe ser agregado al proyecto

Tabla 4.47 Caso de Prueba Guardar Modificaciones al Proyecto

Caso de Prueba 43	Guardar Modificaciones al Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto”

	5. Modificar un archivo del proyecto 6. Guardar cambios
Resultado esperado	Los cambios deben ser guardados exitosamente

Tabla 4.48 Caso de Prueba Eliminar Archivo del Proyecto

Caso de Prueba 44	Eliminar Archivo del Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Dar click secundario sobre un archivo 6. Seleccionar “Eliminar Archivo” 7. Guardar cambios
Resultado esperado	El archivo debe ser eliminado exitosamente

Tabla 4.49 Caso de Prueba Descargar Proyecto

Caso de Prueba 45	Descargar Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón de configuración del proyecto 5. Seleccionar “Descargar Proyecto”
Resultado esperado	El proyecto debe ser descargado exitosamente

Tabla 4.50 Caso de Prueba Eliminar Proyecto

Caso de Prueba 46	Eliminar Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa

	3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón de configuración del proyecto 5. Seleccionar “Eliminar Proyecto” 6. Confirmar la eliminación del proyecto
Resultado esperado	El proyecto debe ser eliminado exitosamente

Tabla 4.51 Caso de Prueba Compartir Proyecto

Caso de Prueba 47	Compartir Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón de configuración del proyecto 5. Seleccionar “Compartir Proyecto” 6. Ingresar el correo del usuario con el cual se va a compartir el proyecto
Resultado esperado	El proyecto debe ser compartido exitosamente

Tabla 4.52 Caso de Prueba Eliminar Integrante del Proyecto

Caso de Prueba 48	Eliminar Integrante del Proyecto
Propósito	Verificar la correcta edición de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón de configuración del proyecto 5. Seleccionar “Eliminar Integrante” 6. Seleccionar integrante
Resultado esperado	El integrante debe ser eliminado del proyecto exitosamente

La barra de herramientas es de utilidad para generar múltiples acciones tales como agregar archivos, así como guardar los cambios relazados sobre uno de ellos, envió de mensajes, creación de plantillas, deshacer, y rehacer acciones, entre otras las cuales son descritas en la tabla 4.52 en adelante.

Tabla 4.53 Caso de Prueba Botón "Codeloud"

Caso de Prueba 49	Botón "Codeloud"
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre "Prueba" 4. Presionar botón "Abrir Proyecto" 5. Presionar botón "CodeCloud"
Resultado esperado	La pantalla de inicio debe ser mostrada

Tabla 4.54 Caso de Prueba Guardar Archivo Barra de Tareas

Caso de Prueba 50	Guardar Archivo Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre "Prueba" 4. Presionar botón "Abrir Proyecto" 5. Modificar cualquier archivo 6. Presionar botón "Archivo" en la barra de tareas 7. Seleccionar "Guardar Archivo"
Resultado esperado	El archivo debe ser guardado exitosamente

Tabla 4.55 Caso de Prueba Deshacer Acción Barra de Tareas

Caso de Prueba 51	Deshacer Acción Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login

	2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Modificar cualquier archivo 6. Presionar botón “Edición” en la barra de tareas 7. Seleccionar “Deshacer”
Resultado esperado	El archivo debe modificar el archivo a un estado anterior

Tabla 4.56 Caso de Prueba Deshacer Acción Barra de Tareas

Caso de Prueba 52	Rehacer Acción Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Modificar cualquier archivo 6. Presionar botón “Edición” en la barra de tareas 7. Seleccionar “Rehacer”
Resultado esperado	El archivo debe modificar el archivo a un estado posterior

Tabla 4.57 Caso de Prueba Seleccionar Todo Barra de Tareas

Caso de Prueba 53	Seleccionar Todo Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Edición” en la barra de tareas 6. Seleccionar botón “Seleccionar Todo”
Resultado esperado	Todo el archivo debe ser seleccionado

Tabla 4.58 Caso de Prueba Enviar Mensaje Barra de Tareas

Caso de Prueba 54	Enviar Mensaje Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto compartido
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Visualización” en la barra de tareas 6. Seleccionar “Multifunciones” 7. Redactar Mensaje 8. Seleccionar botón “Enviar”
Resultado esperado	El mensaje debe ser enviado exitosamente

Tabla 4.59 Caso de Prueba Agregar Plantilla Barra de Tareas

Caso de Prueba 55	Agregar Plantilla Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto compartido
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Visualización” en la barra de tareas 6. Seleccionar “Multifunciones” 7. Seleccionar botón “Crear” 8. Insertar nombre de plantilla 9. Insertar Contenido
Resultado esperado	La plantilla debe ser creada exitosamente

Tabla 4.60 Caso de Prueba Insertar Plantilla Barra de Tareas

Caso de Prueba 56	Insertar Plantilla Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto compartido

	4. Tener una plantilla creada
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Crear un nuevo Archivo 6. Presionar botón “Visualización” en la barra de tareas 7. Seleccionar “Multifunciones” 8. Seleccionar Plantilla 9. Seleccionar botón “Insertar”
Resultado esperado	La plantilla debe ser insertada exitosamente

Tabla 4.61 Caso de Prueba Eliminar Plantilla Barra de Tareas

Caso de Prueba 57	Eliminar Plantilla Barra de Tareas
Propósito	Verificar el correcto funcionamiento de la barra de tareas
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un proyecto compartido 4. Tener una plantilla creada
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Crear un nuevo Archivo 6. Presionar botón “Visualización” en la barra de tareas 7. Seleccionar “Multifunciones” 8. Seleccionar Plantilla 9. Seleccionar botón “Eliminar”
Resultado esperado	La plantilla debe ser eliminada exitosamente

La compilación de los proyectos es pieza fundamental de la funcionalidad de este proyecto, a pesar de que dichas acciones como compilar y ejecutar ya fueron probadas en el capítulo 4.1 es necesario probarlo en la interfaz web, para asegurar que la comunicación entre el servicio y página web funcionan adecuadamente, dichas pruebas se describen a continuación.

Tabla 4.62 Caso de Prueba Compilar Proyecto Exitoso

Caso de Prueba 58	Compilar Proyecto Exitoso
Propósito	Verificar compilación de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa

	3. Tener un Proyecto valido creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Correr” en la barra de tareas 6. Seleccionar “Compilar”
Resultado esperado	Se debe mostrar el mensaje “Compilado exitosamente”

Tabla 4.63 Caso de Prueba Compilar Proyecto no Exitoso

Caso de Prueba 59	Compilar Proyecto no Exitoso
Propósito	Verificar compilación de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un Proyecto con errores de compilación creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Correr” en la barra de tareas 6. Seleccionar “Compilar”
Resultado esperado	Se debe mostrar el mensaje con los errores de compilación

Tabla 4.64 Caso de Prueba Compilar Proyecto Vacío

Caso de Prueba 60	Compilar Proyecto Vacío
Propósito	Verificar compilación de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un Proyecto vacío creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Borrar el contenido de todo el proyecto 6. Presionar botón “Correr” en la barra de tareas 7. Seleccionar “Compilar”
Resultado esperado	Se debe mostrar mensaje “Debe de contener la definición de clase y el mismo nombre del archivo”

Respecto a la ejecución de un proyecto de igual forma que la compilación, esta debe ser probado por lo que se plantean las pruebas en las tablas 4.66, 4.67 y 4.68.

Tabla 4.65 Caso de Prueba Ejecutar Proyecto

Caso de Prueba 61	Ejecutar Proyecto
Propósito	Verificar ejecución de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un Proyecto creado
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Correr” en la barra de tareas 6. Seleccionar “Ejecutar”
Resultado esperado	Se debe mostrar el resultado de la ejecución del proyecto

Tabla 4.66 Caso de Prueba Ejecutar Proyecto Errores de Ejecución

Caso de Prueba 62	Ejecutar Proyecto Errores de Ejecución
Propósito	Verificar ejecución de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un Proyecto creado con errores de ejecución
Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Correr” en la barra de tareas 6. Seleccionar “Ejecutar”
Resultado esperado	Se debe mostrar el resultado del error de la ejecución del proyecto

Tabla 4.67 Caso de Prueba Ejecutar Proyecto Múltiples Archivos

Caso de Prueba 63	Ejecutar Proyecto Múltiples Archivos
Propósito	Verificar ejecución de un proyecto
Prerrequisitos	1. Navegador de Internet 2. Tener una cuenta activa 3. Tener un Proyecto creado con más de 10 archivos

Pasos	1. Ir a la url http://34.205.17.240/login 2. Iniciar sesión en la plataforma 3. Seleccionar proyecto de nombre “Prueba” 4. Presionar botón “Abrir Proyecto” 5. Presionar botón “Correr” en la barra de tareas 6. Seleccionar “Ejecutar”
Resultado esperado	Se debe mostrar el resultado de la ejecución del proyecto

4.2.2 Ejecución de Pruebas

Los resultados de la ejecución de las pruebas manuales para la página web se muestran a continuación, en la figura 4.39 se muestra los datos ingresados para el registro del usuario y en la figura 4.40 se muestra la pantalla de inicio, una vez que el usuario ha sido creado, dichos resultados correspondientes a la prueba 30 representada en la tabla 4.34.

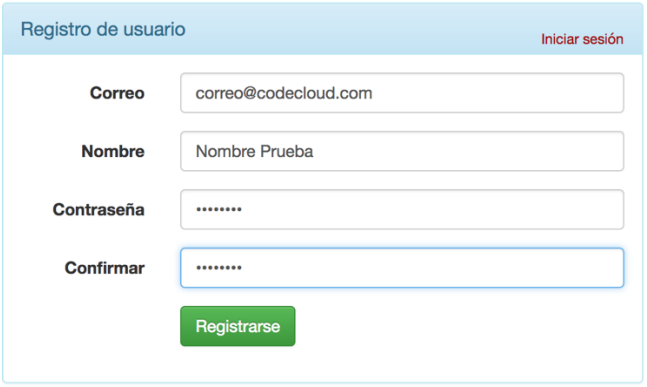

Figura 4.39 Registro Datos Usuario

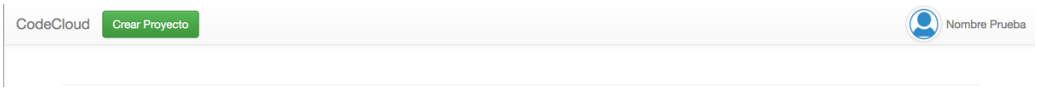

Figura 4.40 Registro de Usuario Exitoso

En la figura 4.41 se muestra el resultado de la prueba descrita en la tabla 4.35, impidiendo la creación de un usuario existente.

Figura 4.41 Registro de Usuario Inválido

Para el caso de prueba descrito en la tabla 4.36 los campos de contraseña no son llenado y se intenta registrar a un usuario de esta forma, dicho resultado de esta prueba es el que se muestra a continuación en la figura 4.42.

Figura 4.42 Registro de Usuario Contraseñas Faltantes

Cabe mencionar que el botón “Registrarse” se mantiene deshabilitado y no es posible presionarlo para registrar al usuario, además de los mensajes que solicitan que se cumplan las condiciones faltantes. En la figura 4.43 se muestra la ejecución de la prueba representada en la tabla 4.37 la cual consiste en registrar a un usuario sin introducir la confirmación de la contraseña.

Figura 4.43 Registro de Usuario Confirmar Contraseñas Faltante

Para la prueba descrita en la tabla 4.38 el resultado de su ejecución correspondiente es la que se muestra en la figura 4.43.

Figura 4.44 Registro de Usuario Correo no Valido

En la figura 4.45 (Datos de inicio de sesión) y 4.46 (sesión iniciada) se muestra el resultado de la ejecución de la prueba representada en la tabla 4.39.

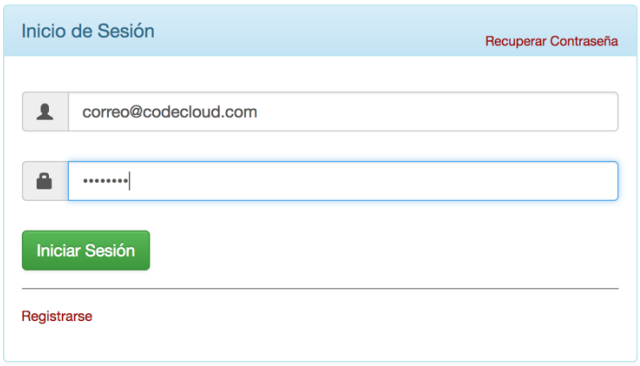

Figura 4.45 Datos Inicio de Sesión

Figura 4.46 Prueba Sesión Iniciada

Los resultados de la ejecución de la prueba descrita en la tabla 4.40 se muestran en la figura 4.47.

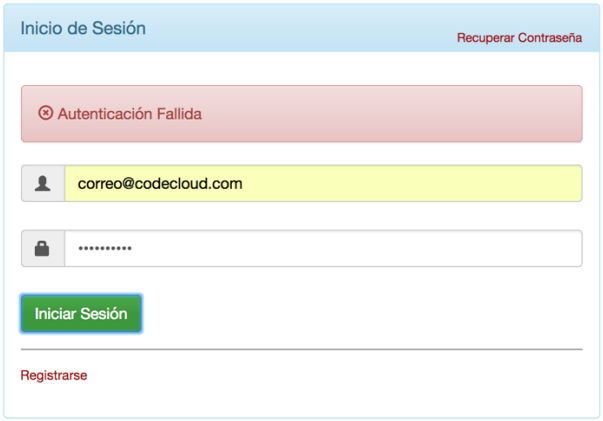

Figura 4.47 Prueba Inicio de Sesión Fallida

Para el caso de prueba descrito en la tabla 4.41; su ejecución correspondiente es la que se muestra en la figura 4.48.

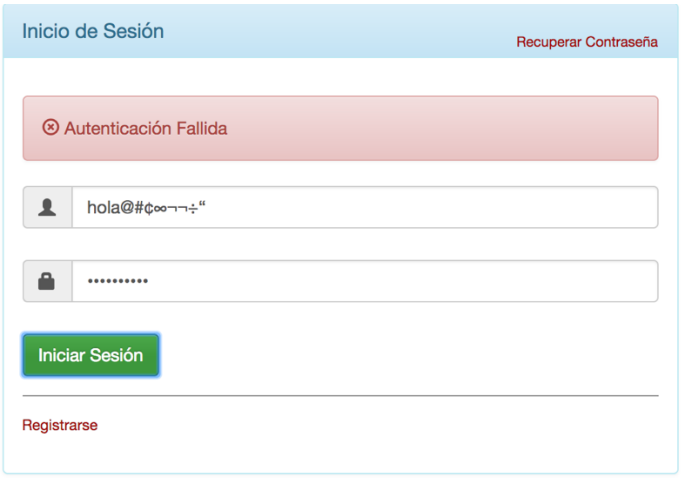

Figura 4.48 Prueba Inicio de Sesión Caracteres Especiales Fallida

Para la prueba (tabla 4.42) se necesita un correo válido para recibir el correo del restablecimiento de la contraseña, es por eso que se usa la cuenta “antonio_lozano@outlook.com” para la ejecución de esta prueba, el resultado de dicha prueba es el que se muestra en las figuras 4.49, 4.50, 4.51, 4.52 y 4.53.

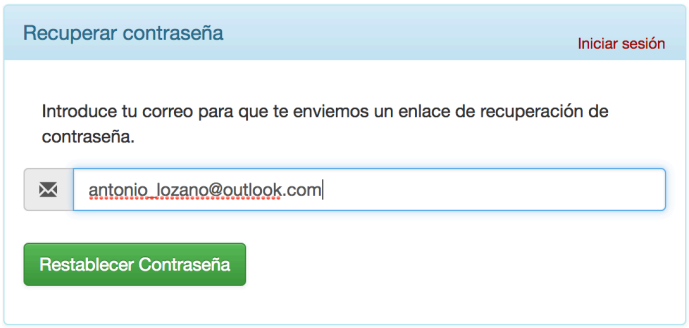

Figura 4.49 Prueba Correo a Restablecer Contraseña

Figura 4.50 Confirmación de Correo Enviado

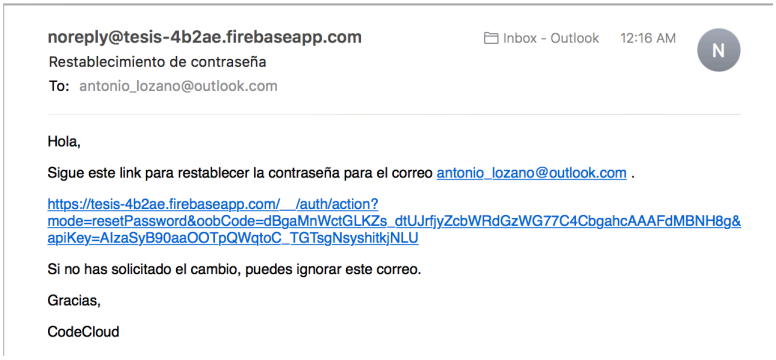

Figura 4.51 Correo Recibido

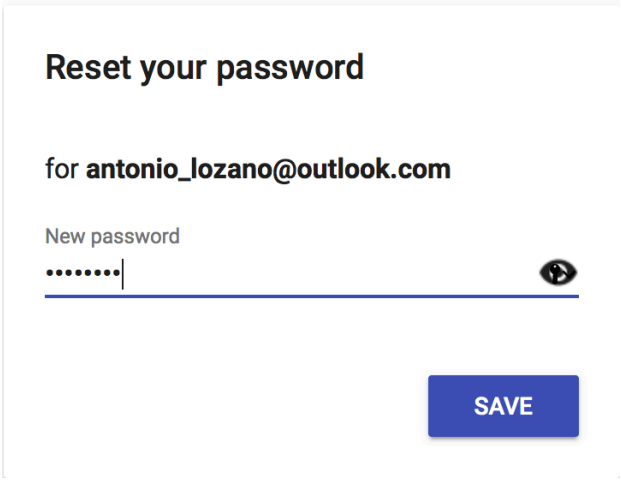

Figura 4.52 Restablecer Contraseña

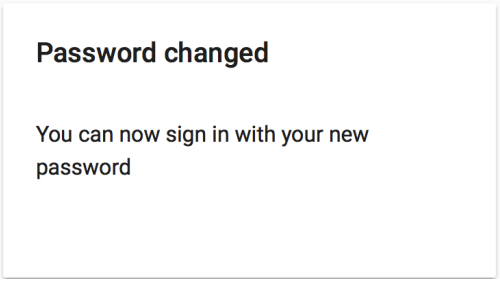

Figura 4.53 Confirmación Contraseña Restablecida

Para la prueba representada en la tabla 4.43 es necesario realizar el cierre de sesión de un usuario y validar que dicha acción se realice de forma adecuada, en la figura 4.53 se muestra el botón de cerrar sesión mientras que en la imagen 4.54 se muestra la pantalla de inicio después de haber cerrado la sesión.

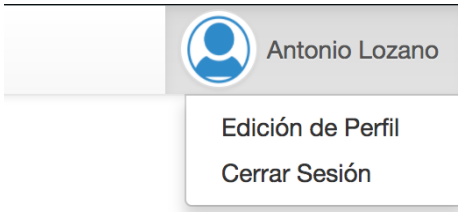

Figura 4.54 Botón Cerrar Sesión

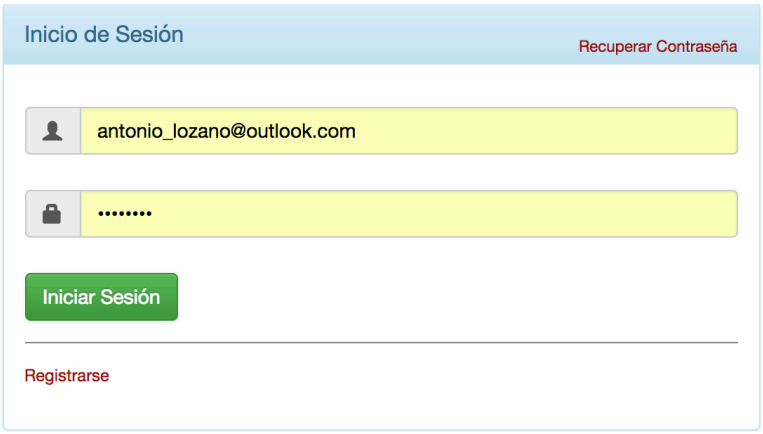

Figura 4.55 Sesión Finalizada

Las pruebas correspondientes encargadas de verificar la funcionalidad de los proyectos son las que se describen a continuación, en la figura 4.56 se puede observar la ventana correspondiente a la creación de un nuevo proyecto mientras que en la 4.57 se puede observar el proyecto creado.

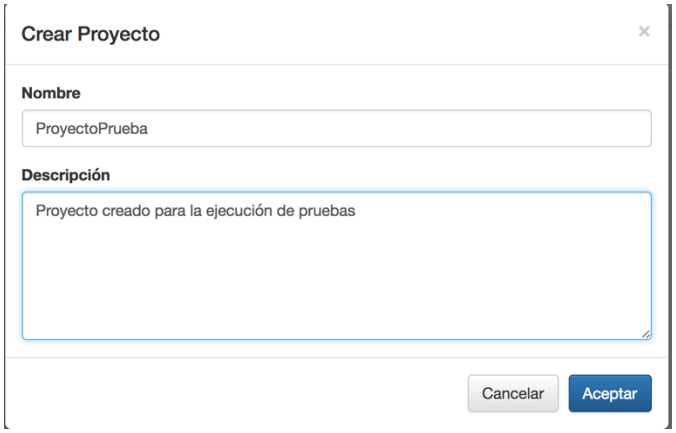

Figura 4.56 Ventana Crear Proyecto

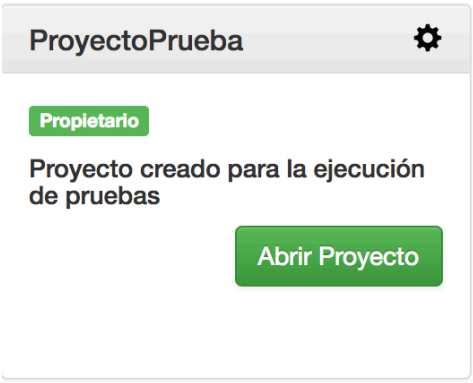

Figura 4.57 Proyecto Creado

Para la prueba representada en la tabla 4.45 correspondiente a la acción de abrir un proyecto, podemos observar en la imagen 4.58 el botón el cual se encarga de abrir el proyecto y en la figura 4.59 podemos observar el proyecto listo para ser editado.

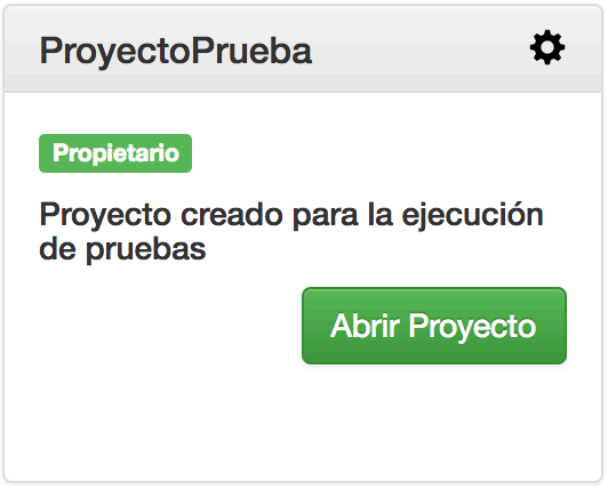

Figura 4.58 Botón Abrir Proyecto

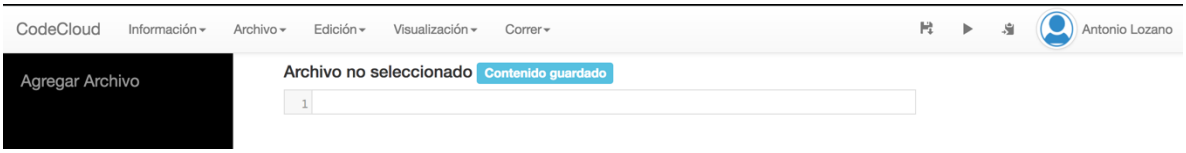

Figura 4.59 Proyecto Abierto

La tabla 4.46 menciona la prueba encargada de verificar la correcta función para la adición de archivos a un proyecto, en la figura 4.60 se muestra la ventana que solicita el nombre de dicho archivo a ser agregado y en la figura 4.61 se muestra el archivo creado.

Figura 4.60 Agregar Archivo a Proyecto

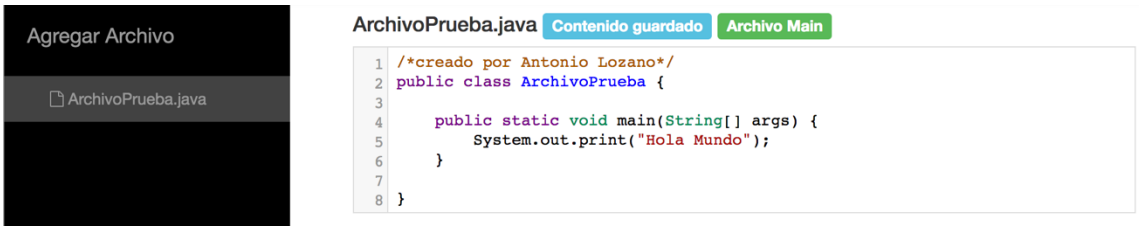

Figura 4.61 Archivo Agregado al Proyecto

Para la ejecución de la prueba correspondiente a la modificación de un archivo (tabla 4.47) se muestran las figuras 4.62 se muestra el archivo modificad y en la figura 4.63 se muestra el archivo una vez los cambios fueron guardados.

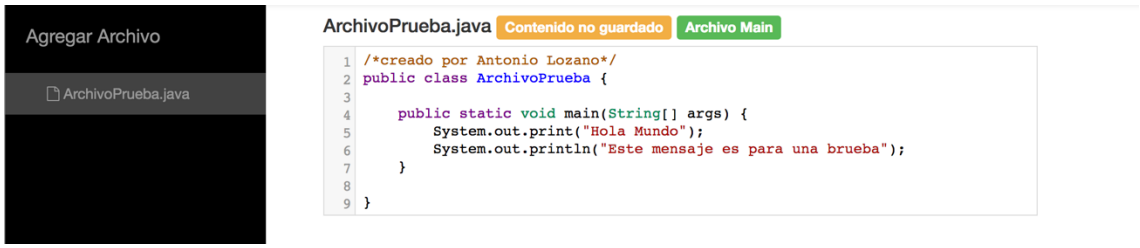

Figura 4.62 Modificar Archivo

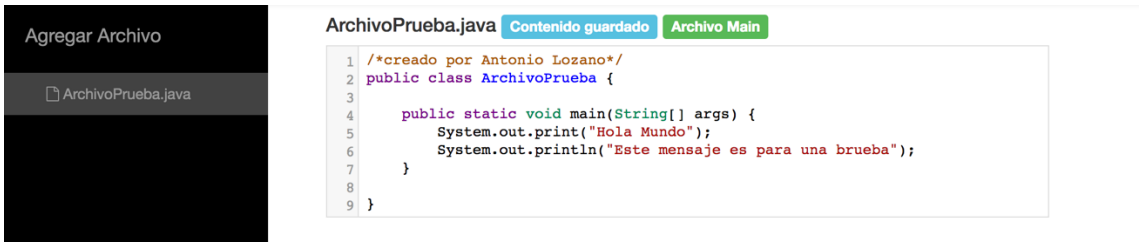

Figura 4.63 Archivo Guardado

Para la prueba correspondiente a la eliminación de un archivo descrita en la tabla 4.48 se muestran la figura 4.64 en la cual se muestra un archivo antes de ser eliminado y la 4.65 una vez que dicho archivo fue removido del proyecto.

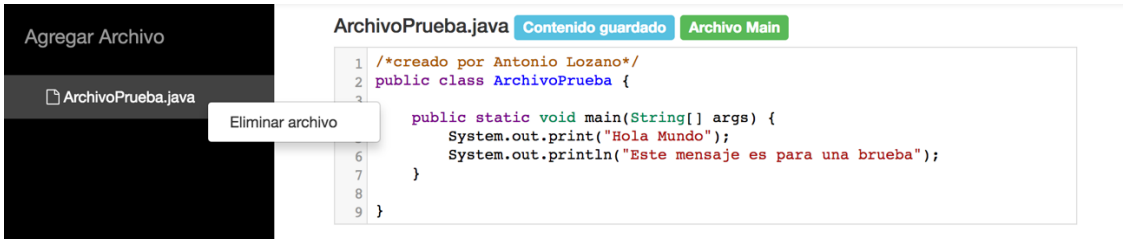

Figura 4.64 Eliminar Archivo

Figura 4.65 Eliminar Archivo

Para figura 4.66 se observa el botón de descarga del proyecto y en la 4.67 se muestra el proyecto descargado, dichas figuras son el resultado de la prueba representada en la tabla 4.49 (Caso de Prueba Descargar Proyecto).

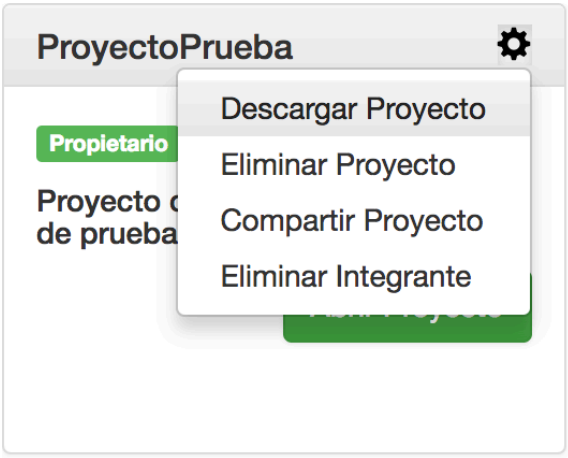

Figura 4.66 Botón Descargar Proyecto

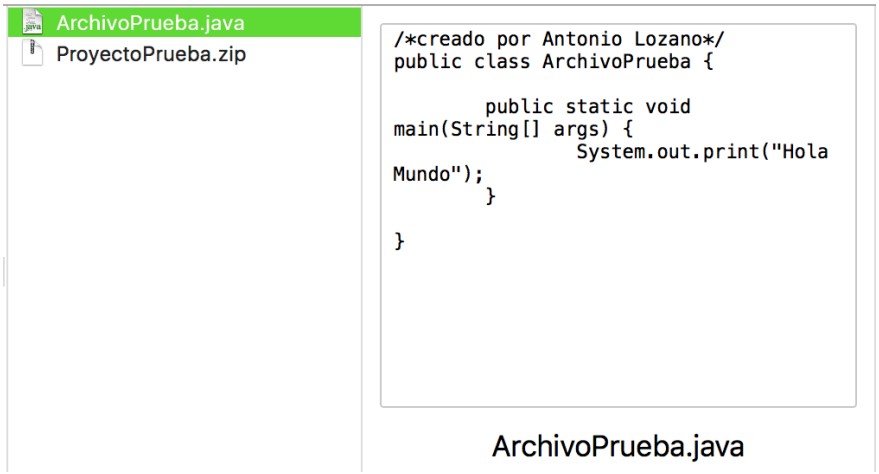

Figura 4.67 Proyecto Descargado

La prueba mencionada en la tabla 4.50 habla de la eliminación del proyecto y validar que este se ejecute de forma adecuada, en la figura 4.68 se muestra el botón para eliminar dicho proyecto y en la 4.69 se muestra la ventana de proyectos en donde se aprecia que dicho proyecto ya no existe.

Figura 4.68 Botón Eliminar Proyecto

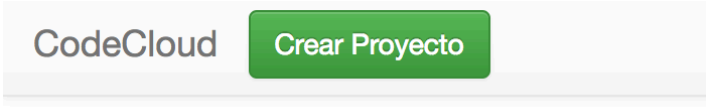

Figura 4.69 Proyecto Eliminado

Compartir un proyecto es un proceso un poco más complicado, se necesita contar con al menos 2 cuentas válidas para poder realizar esta acción, en la figura 4.70 se muestra la ventana correspondiente a la acción de compartir un proyecto y en la 4.71 se muestra el proyecto una vez compartido, dichas figuras corresponden a la prueba descrita en la tabla 4.51.

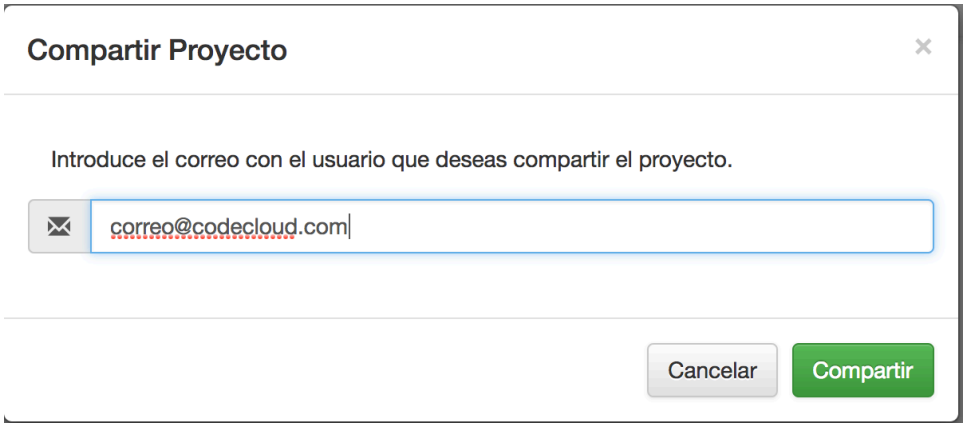

Figura 4.70 Compartir Proyecto

Figura 4.71 Proyecto Compartido

En la prueba (tabla 4.52) correspondiente a la eliminación de un integrante de un proyecto se cuenta con un proyecto con 2 integrantes adicionales al dueño de dicho proyecto, en la figura 4.72 se muestra una lista con los integrantes que pueden ser elegidos para eliminar y en la figura 4.73 se muestra el proyecto una vez que dicho integrante "nombre Prueba" fue removido.

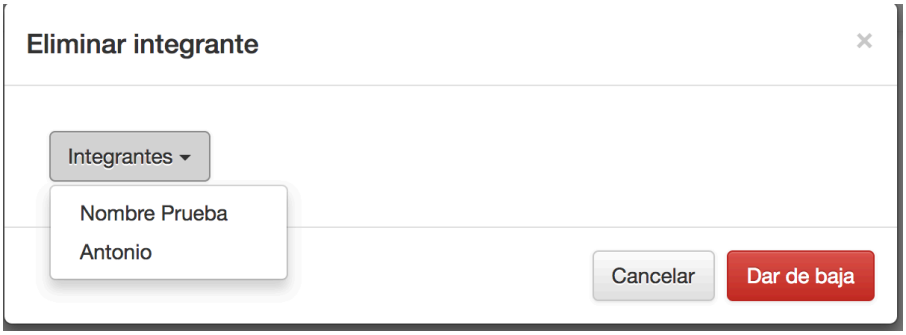

Figura 4.72 Eliminar Integrante

Figura 4.73 Integrante Eliminado

La barra de herramientas juega otro papel fundamental, ya que esta permite realizar múltiples acciones que están disponibles únicamente a través de ella, algunas otras pueden ser realizadas de diferentes formas, tales como "Deshacer" y "Rehacer" pero para enviar mensajes o crear plantillas al proyecto únicamente se pueden acceder a estas acciones a través de la barra de tareas. La barra de tareas consta con un botón llamado "CodeCloud" que se encuentra en el extremo superior izquierdo cuya función principal es la de llevarte a la página de proyectos, existe una prueba descrita en la tabla 4.53 la cual tiene como propósito validar esta acción, en la figura 4.74 se muestra dentro de un proyecto el botón "CodeCloud" y en la figura 4.75 se muestra la pantalla principal una vez presionado este botón.

Figura 4.74 Botón "CodeCloud" Barra de Tareas

Figura 4.75 Pantalla Principal

La barra de tareas permite guardar a través del menú "Archivo" esta prueba está representada en la tabla 4.54 y en la figura 4.76 podemos observar el botón "Guardar" y en la figura 4.77 se muestra el archivo una vez que los cambios fueron guardados.

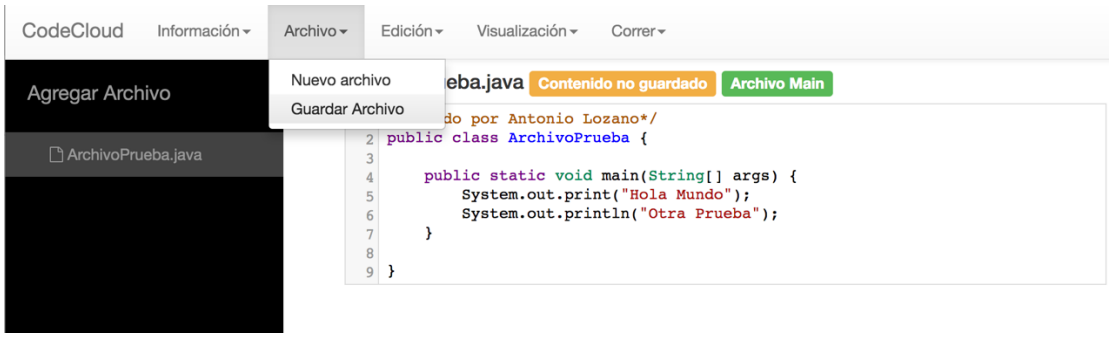

Figura 4.76 Guardar Barra de Tareas

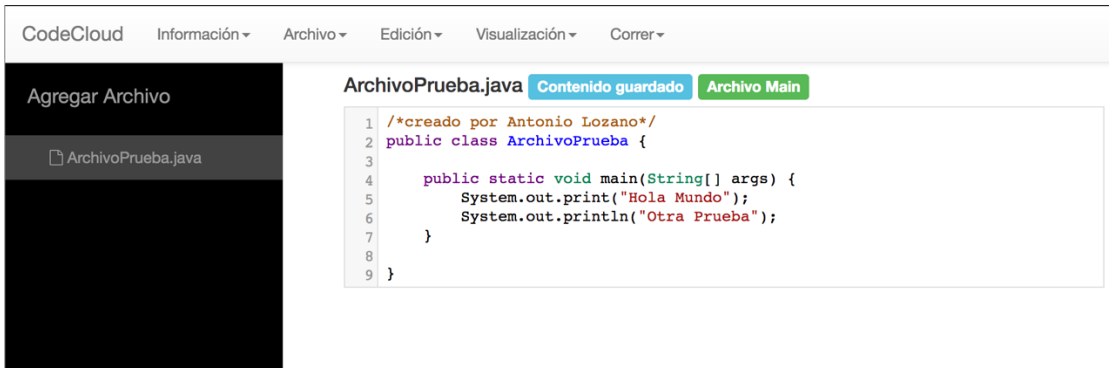

Figura 4.77 Archivo Guardado Barra de Tareas

La barra de tareas en el menú "Edición" cuenta con el botón "Deshacer" que nos permite regresar a un estado anterior del archivo cual sea que se esté editando, probar esta funcionalidad se define en la tabla 4.55 y en la figura 4.78 podemos observar dicho botón mientras que en la figura 4.79 podemos observar que el archivo fue modificado al haber regresado a un estado anterior.

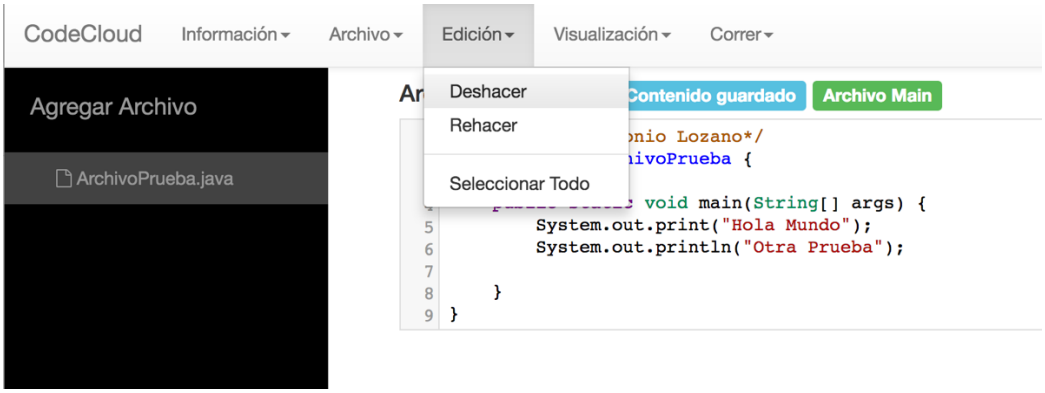

Figura 4.78 Deshacer Barra de Tareas

Figura 4.79 Acción Anterior Barra de Tareas

EL menú "Edición" también cuenta con botón "Rehacer" el caso de prueba de esta funcionalidad se plantea en la tabla 4.56 en la figura 4.80 se muestra el botón mencionado anteriormente, mientras que en la figura 4.81 se muestra el archivo una el archivo se llevo a un estado más reciente.

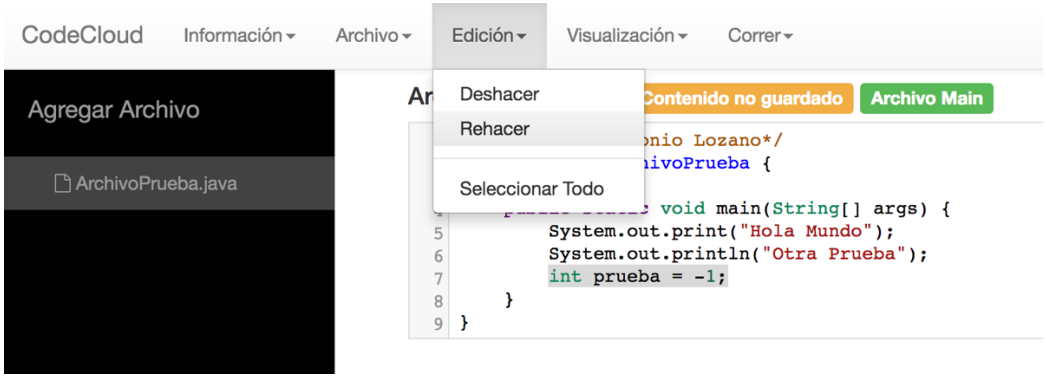

Figura 4.80 Rehacer Barra de Tareas

Figura 4.81 Acción Actual Barra de Tareas

A su vez existe el botón "Seleccionar todo" que nos permite marcar todo el texto que contiene el archivo abierto en cuestión, dicha funcionalidad se plantea la forma de probarla en la tabla 4.57 y en la figura 4.82 se muestra el botón "Seleccionar Todo" y en la figura 4.83 se muestra el contenido del archivo seleccionado.

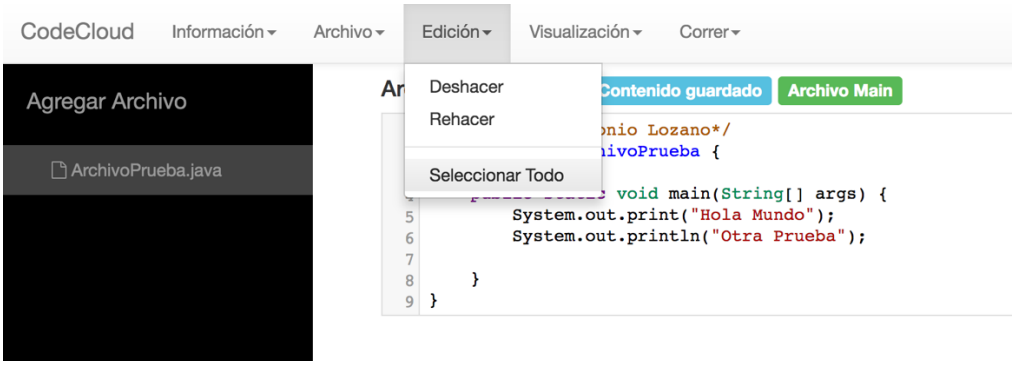

Figura 4.82 Seleccionar Todo Barra de Tareas

Figura 4.83 Contenido Archivo Seleccionado

Las figuras 4.84 y 4.85 muestran el botón “Multifunciones” que permite realizar el envío de mensajes y el envío de un mensaje respectivamente, dichas figuras son el resultado del caso de prueba descrito en la tabla 4.58.

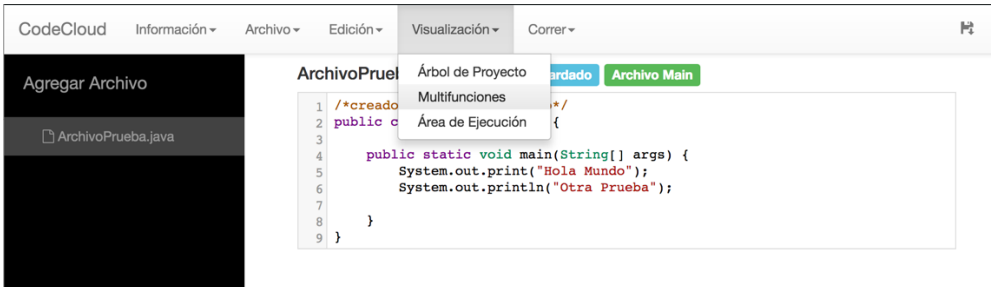

Figura 4.84 Multifunciones Barra de Tareas

Figura 4.85 Mensajes

Para acceder a las plantillas es necesario entrar al menú “Visualización” botón “Multifunciones” figura 4.84; la prueba encargada de validar la funcionalidad de agregar una nueva plantilla es descrita en la tabla 4.59. EN la figura 4.86 podemos observar la ventana en donde se crean las plantillas y en la figura 4.87 podemos ver que en la lista de plantillas no fue creada, por lo que dicha prueba falló.

Figura 4.86 Crear Plantilla

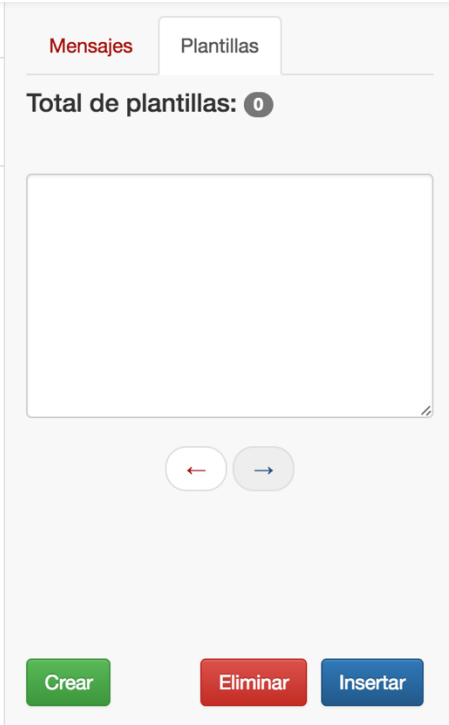

Figura 4.87 Crear Plantilla Prueba no Exitosa

Para corregir este error fue necesario modificar los archivos "editor.component.ts" (figura 4.88) "editor.component.html" (figura 4.89) el problema rescindía en que al tratar de obtener las plantillas existentes no era posible ya que dichas plantillas no se encontraban en la base de datos.

```
consulta_plantillas.subscribe(plantillas => {

  if(plantillas.length != 0) {
    //inicializar arreglo para volver a insertar los mensajes
    this.plantillas_proyecto = [];
    this.contador_plantillas = 0;

    for(let plantilla of plantillas) {
      this.plantillas_proyecto.push(plantilla);
    }

    this.cantidad_plantillas = this.plantillas_proyecto.length;
    this.limite_contador = this.plantillas_proyecto.length - 1;

    this.ds.plantilla_seleccionada = this.plantillas_proyecto[this.contador_plantillas];
    this.area_texto_contenido_plantilla = this.ds.plantilla_seleccionada.contenido_plantilla;
  } else {
    //No hay plantillas o se eliminó la última plantilla
    this.cantidad_plantillas = 0;
    this.area_texto_contenido_plantilla = "";
  }


    });
```

Figura 4.88 Modificación Archivo "editor.component.ts"

```
</div>
<div id="plantillas" class="tab-pane fade">
    <h4>Total de plantillas: <span class="badge">{{cantidad_plantillas}}</span></h4>
    <h4 class="pager" *ngIf="cantidad_plantillas != 0">{{ds.plantilla_seleccionada.nombre_plantilla}}</h4>
    <textarea class="form-control" rows="8" [(ngModel)]="area_texto_contenido_plantilla"></textarea>

    <ul class="pager">
        <li><a href="javascript:void(0)" (click)="seleccionar_plantilla_anterior()" *ngIf="contador_plantillas != 0"><span aria-hidden=
        <li><a href="javascript:void(0)" (click)="seleccionar_plantilla_siguiente()" *ngIf="contador_plantillas != limite_contador"><sp
    </ul>

    <div class="col-lg-3 col-sm-4 col-6 fixedheight">

        <div>
            <button class="bottomleft btn btn-success" (click)="abrir_modal_crear_plantilla()" >Crear</button>

            <button class="bottomright btn btn-primary" (click)="insertar_plantilla()">Insertar</button>

            <button class="bottomaligned btn btn-danger" (click)="abrir_modal_eliminar_plantilla()">Eliminar</button>

        </div>

    </div><!-- /.col-lg-3 -->
</div>
```

Figura 4.89 Modificación Archivo "editor.component.html"

Y al ejecutar nuevamente la prueba como se puede ver en la figura 4.90 y 4.91 la ejecución de dicha prueba se lleva de forma exitosa.

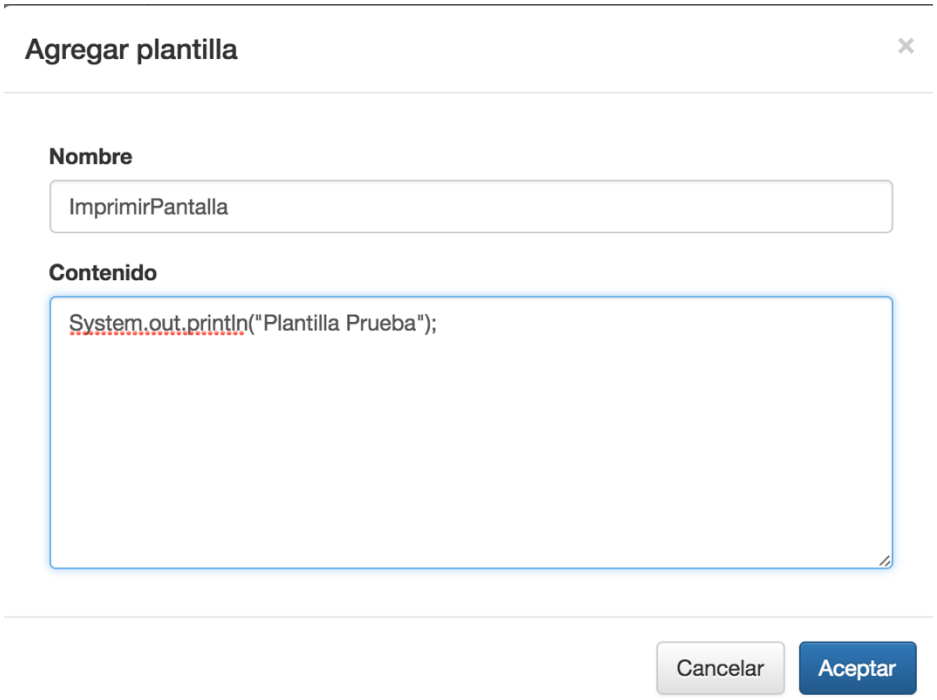

Figura 4.90 Crear Plantilla

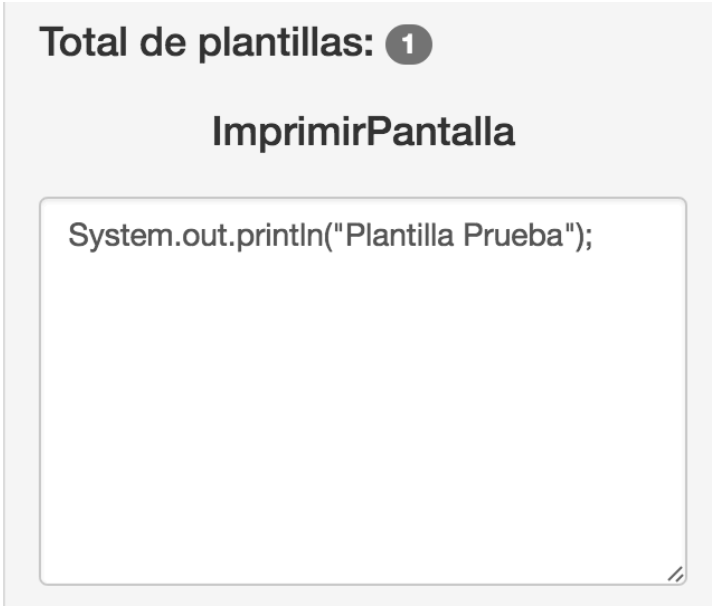

Figura 4.91 Plantilla Creada

El caso de prueba 59 (tabla 4.60) podemos observar la inserción de una plantilla en un archivo del proyecto (figura 4.92).

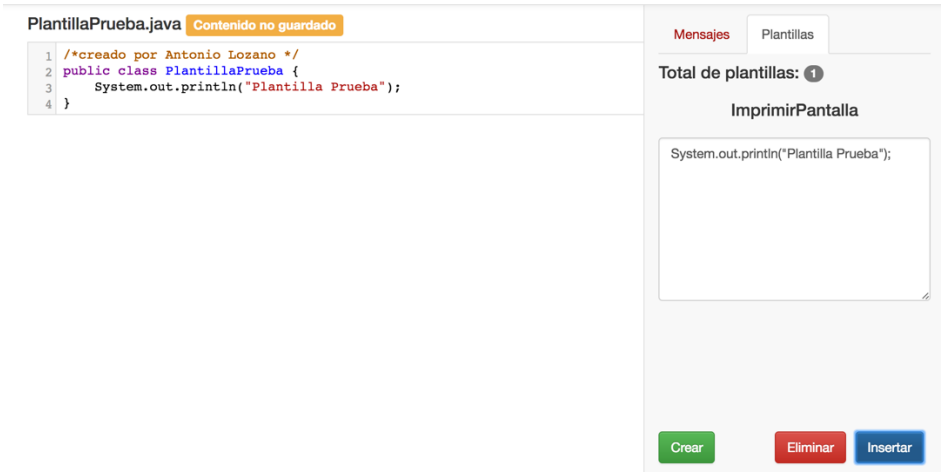

Figura 4.92 Insertar Plantilla

La siguiente prueba consiste en eliminar una plantilla (tabla 4.61) en la figura 4.93 podemos observar el mensaje de confirmación para eliminar una plantilla y de lado derecho el contador de plantillas en 1, mientras que en la figura 4.94 se observa la plantilla eliminada y el contador de plantillas en 0.

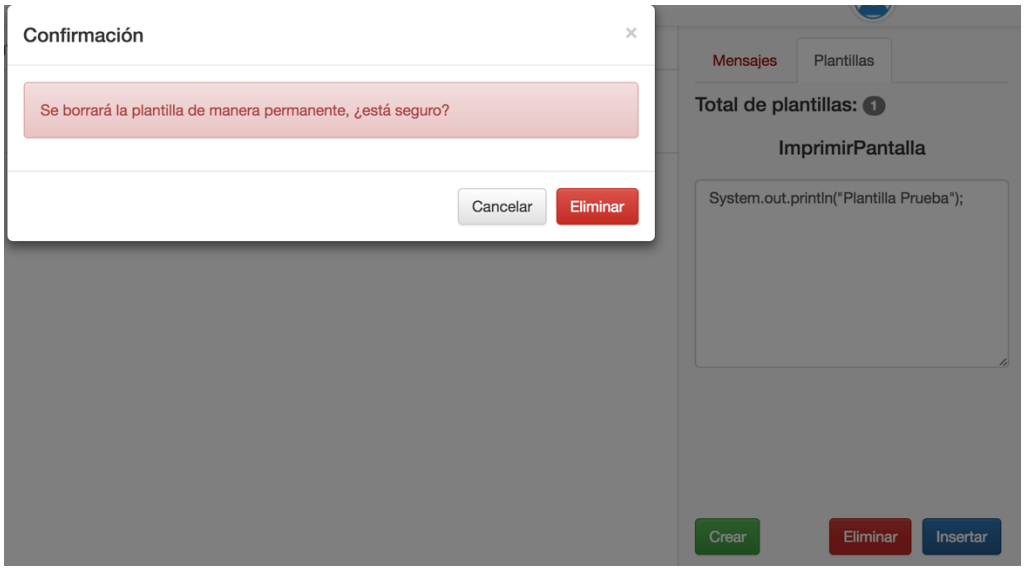

Figura 4.93 Eliminar Plantilla

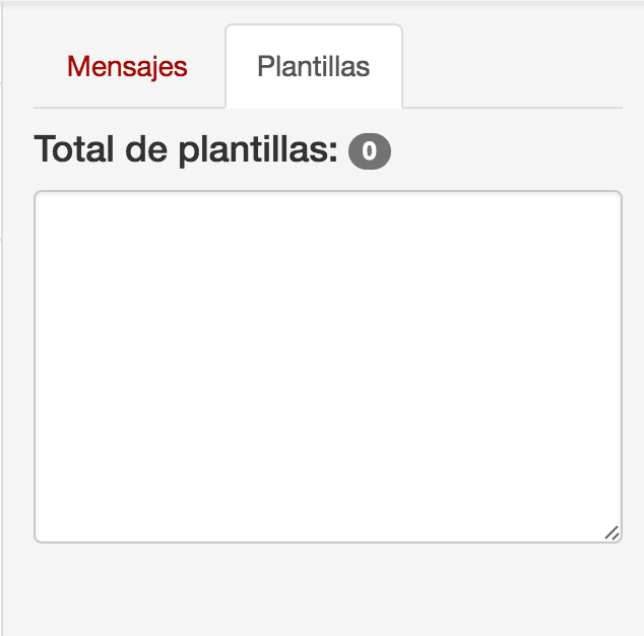

Figura 4.94 Plantilla Eliminada

Una vez concluidas las pruebas referentes a la barra de tareas es necesario realizar pruebas sobre la compilación de los proyectos, en la figura 4.61, en la figura 4.95 se puede observar la compilación del proyecto, en el área de la consola del proyecto.

Figura 4.95 Compilación Exitosa

En la figura 4.96 se observa la compilación fallida del proyecto, en la que únicamente falta un punto y coma para que la compilación no pueda realizarse con éxito, esta prueba es descrita en la tabla 4.63.

Figura 4.96 Compilación no Exitosa

Para finalizar con la prueba de compilación de un proyecto tenemos la prueba de la compilación de un proyecto vacío (tabla 4.64) en la figura 4.97 se puede observar que la página web evita que se pueda realizar la compilación de un proyecto si este no tiene archivos existentes en él.

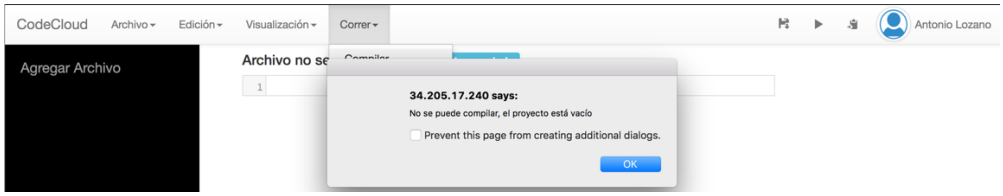

Figura 4.97 Compilación Proyecto Vacío

Las pruebas para la ejecución de un proyecto sirven no solo para verificar el correcto comportamiento de esta acción sino también para reforzar las pruebas de compilación ya que la ejecución de un proyecto lleva a cabo la compilación del mismo, en la figura 4.98 se puede observar la ejecución correcta de la prueba de ejecutar un proyecto de forma exitosa descrita en la tabla 4.65.

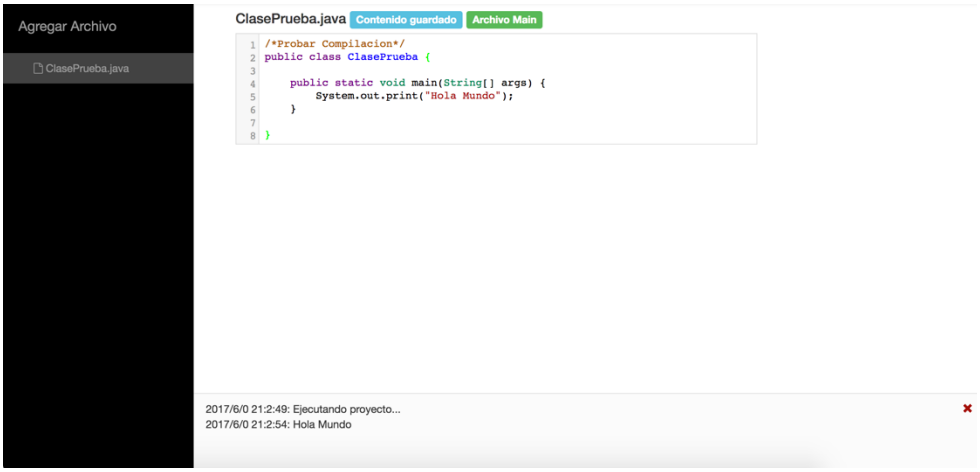

Figura 4.98 Ejecución Proyecto Exitoso

Para realizar la prueba "Ejecutar Proyecto no Exitoso" mostrado en la tabla 4.66 es necesario generar un error en tiempo de ejecución, de lo contrario al tratar de compilar surgirán errores y no permite realizar la ejecución es por es que para esta prueba se genera una excepción al tratar de acceder a un índice inexistente de un arreglo, el resultado de la prueba se puede observar en la figura 4.99.

Figura 4.99 Ejecución Proyecto no Exitoso

La prueba referente a la ejecución de un proyecto que contiene diez archivos (tabla 4.67) se muestra en las siguientes figuras; en la figura 4.100 se muestra el contenido del archvio "ClaseUno.java" que es el mismo que contienen otras clases creadas para esta prueba que únicamente contienen un método que retorna un uno que le permite a la clase principal hacer la suma de todas las clases, en la figura 4.101 se observa la clase principal y el resultado de la ejecución del proyecto.

ClaseUno.java Contenido guardado

```
1 /*creado por Antonio Lozano */
2 public class ClaseUno {
3     public static int contador(){
4       return 1;
5     }
6 }
```

Figura 4.100 Contenido Archivo "ClaseUno.java"

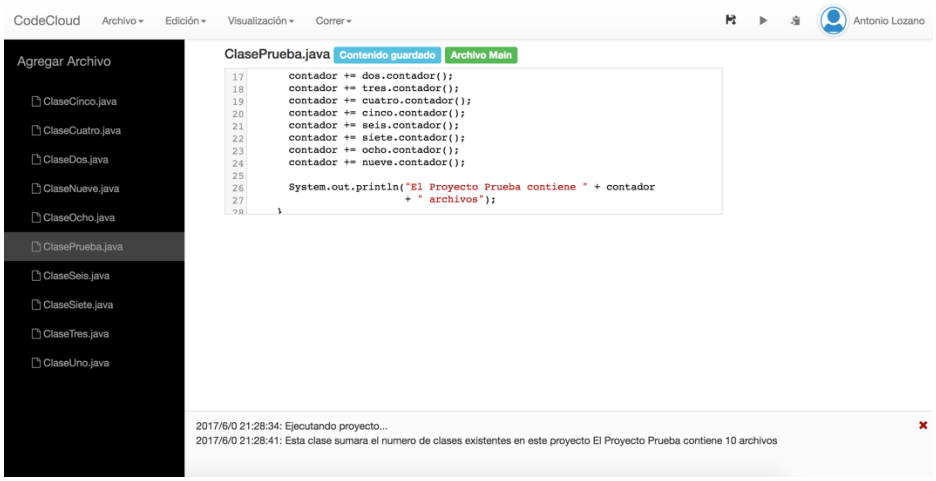

Figura 4.101 Ejecución Proyecto Múltiples Archivos

4.3 Análisis de Resultados de Pruebas

Una vez concluidas las pruebas del servicio y aplicación web es importante realizar un análisis del resultado, las pruebas correspondientes al servicio web tuvieron un enfoque funcional, pero a su vez para analizar la capacidad o tiempo de respuesta que maneja el servicio web, mientras que las pruebas enfocadas para la página web fueron únicamente funcionales, para validar cada una de las características de la aplicación, validar que se hicieran de forma correcta.

A lo largo de las pruebas se puede observar que el código original tanto de la aplicación web como el servicio sufrieron cambios, esto debido a que no cumplieron con el resultado esperado de las pruebas y tuvieron que realizarse modificaciones al código para que dichas pruebas pudiesen ejecutarse de forma adecuada.

Analizando las pruebas del servicio web se puede observar que de un total de 29 pruebas 3 de ellas no pasaron el criterio de aceptación, esto equivale a un 89.6% de efectividad en el código desarrollado como se puede observar en la figura 4.102

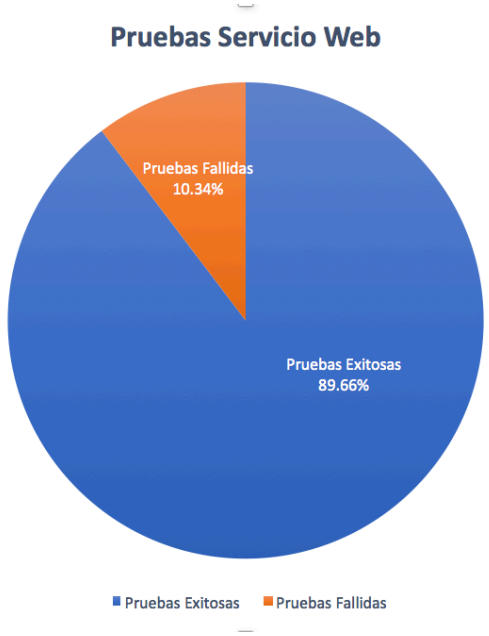

Figura 4.102 Porcentaje Efectividad de Código Servicio Web

También se puede observar que de las 5 secciones en las que se dividieron las pruebas (tabla 4.2) la sección que presentó más fallas al ejecutar las pruebas fue la de “Ejecutar” ya que esta es la que unifica todas las secciones del servicio web y la que permitió observar las pruebas existentes en otras secciones de código que no fueron vistas en la ejecución de las pruebas de otras secciones.

Respecto a los tiempos de ejecución de las pruebas que compilan 5, 10 y 20 proyectos tenemos tiempos de respuesta 8.214, 18.028 y 37.418 segundos respectivamente para cada prueba, se puede observar en la gráfica 4.103 el resultado de estas pruebas.

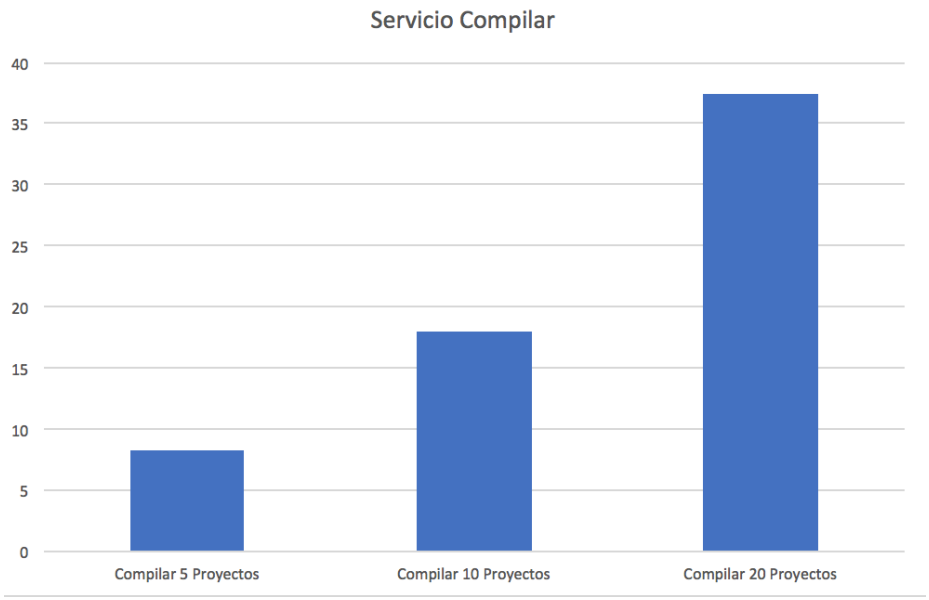

Figura 4.103 Tiempo de Compilación Múltiples Proyectos

Esto concluye que tenemos un incremento en el tiempo de ejecución para los mismos archivos, como se puede ver en la figura 4.104 que tenemos un incremento en el tiempo de compilación de cada archivo tomando en cuenta el tiempo total de compilación de los 5 proyectos se puede decir que cada proyecto le tomó 0.8214 segundos para la compilación de los 5 ya mencionados, 1.8028 segundos por proyecto de los 10 compilados y 1.8709 segundos por proyectos para la compilación de 20.

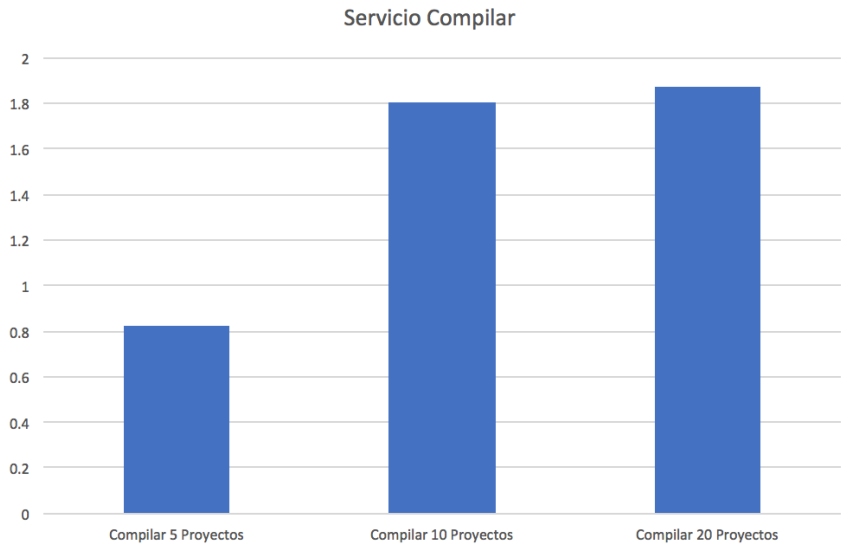

Figura 4.104 Tiempo de Compilación por Proyecto

Para las pruebas “Ejecutar” del servicio web se registraron tiempos de ejecución de 11.238, 16.503 y 38.665 segundos para 5, 10 y 20 proyectos respectivamente como se puede observar en la figura 4.105.

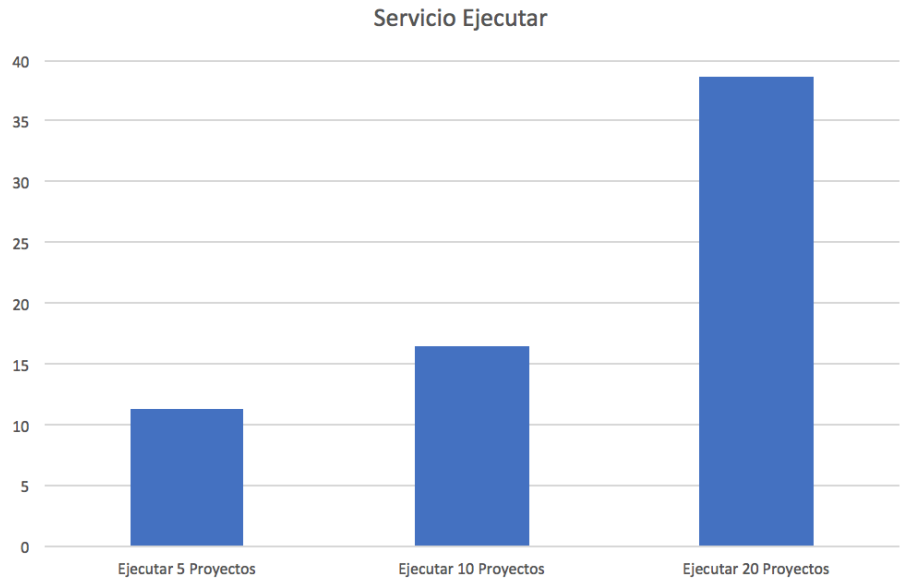

Figura 4.105 Tiempo de Ejecución Múltiples Proyecto

Esto implica que se tienen tiempos de ejecución por proyecto de 1.1238, 1.6503 y 1.93325 segundos por proyecto ejecutado, estos resultados los podemos observar en la figura 4.106.

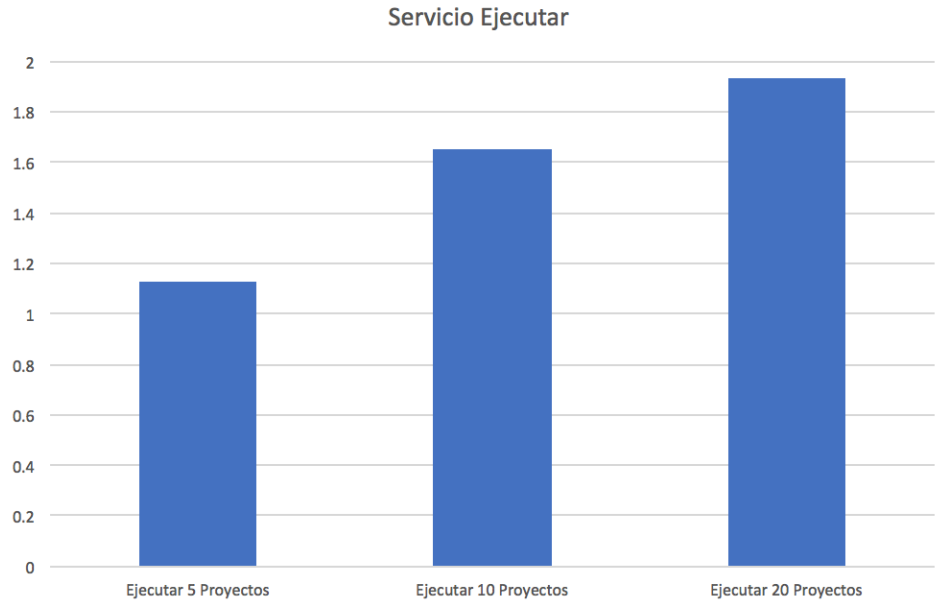

Figura 4.106 Tiempo de Ejecución por Proyecto

De igual forma que el servicio “Compilar” tenemos un incremento en el tiempo de ejecución por proyecto directamente proporcional al número de llamadas que recibe el servicio web, con estos resultados se estima que el servicio web es capaz de recibir múltiples llamadas sin

afectar de forma agresiva su desempeño conforme el número de peticiones al mismo se incremente.

Las pruebas correspondientes a la página web tienen una perspectiva funcional como ya se había mencionado anteriormente en el capítulo 3, el análisis al que se puede llegar es que las pruebas se realizaron de forma correcta en un 96.97% tan solo una prueba no pudo realizarse de forma correcta.

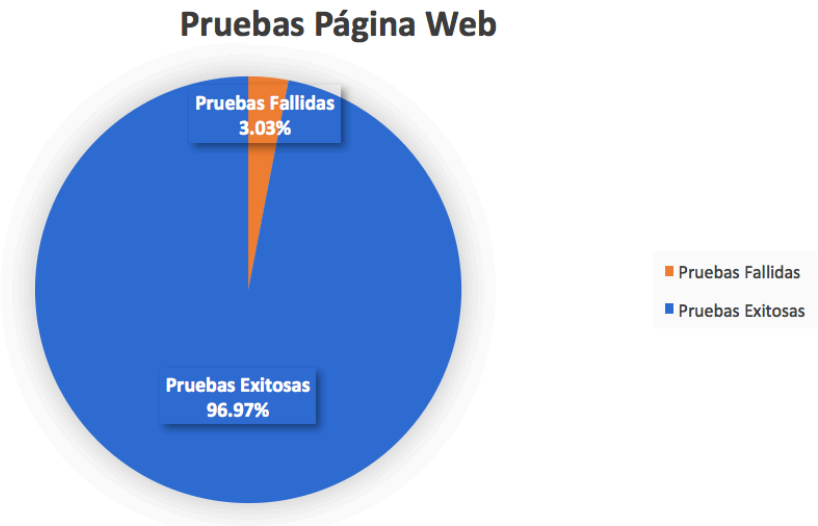

Figura 4.107 Porcentaje Efectividad de Código Página Web

Con esto podemos deducir que el código de error presentado en el código es aceptable y que este fue desarrollado de forma adecuada. Con esto se concluye el análisis de las pruebas y la integración entre las diferentes secciones del proyecto se realizaron de forma adecuada y son funcionales para los usuarios, con esto se concluye que el proyecto está listo para salir a producción.

Conclusiones

Una vez conluido el desarrollo del sistema se puede destacar lo siguiente:

La creación de la aplicación para el desarrollo de software muestra una mejora en cuestion a los tiempos de desarrollo, ya que este permite al usuario hacer modificaciones a su código sin necesidad de equipos robustos, a su vez el sistema permite realizar acciones importantes como la creacción de plantillas de código de fácil incersión, compilación y ejecución de proyectos de forma rápida y eficiente, cuyas acciones presentan una reducción de tiempos en la implementación de dichos proyectos.

La aplicación web que permite el almacenamiento de proyectos en la nube facilita la creación de proyectos, ya que en cuestión de segundos se puede crear un proyecto nuevo o modificar uno existente, por lo que el usuario solo debe preocuparse por el contenido del proyecto y la creación del mismo. Esta propuesta permitiría a las fábricas de software dejar de lado necesidades como compra de equipos con alta capacidad, adquisición de servicios de instalación de ambientes de desarrollo y entre otros.

Las reducciones de tiempo sobre la creación de proyectos son importantes ya que en la instalación de cualquier *IDE* de desarrollo tarda por lo menos 10 minutos en descargar e instalar, mientras que con este sistema es posible descartar todo ese tiempo y dedicarlo a la administración del proyecto.

Al integrar la funcionalidad de mensajería, la comunicación entre los integrantes de un proyecto permite un mejor entendimiento de lo que se requiere para cual sea el proyecto en cuestion que se esté trabajando, ya que todos los desarrolladores están al tanto de lo que se necesita, al contra con un canal grupal no se presentan situaciones en las que la información de lo que se necesita no llegue o se distorciona, además de que dicho canal puede ser accedido desde la misma ventana en la que se está desarrollando el código.

La creación de plantillas es muy útil ya que esta permite la incersión de código de forma rápida y eficiente, ya que en proyectos puede presentarse la situación en la que la estructura del mismo requiera la creación de métodos o clases similares, por lo que con las plantillas se puede crear una en la que se diseñe un bloque de código generico que se pueda insertar y realizar las modificaciones que este necesite dependiendo de su objetivo.

Con esto se puede decir que la creación del sistema propuesto en la presente tesis, cumple con los objetivos previamente establecidos ya que se desarrolló una aplicación web que se puede consumir a través del navegador para crear, compartir, ejecutar y compilar proyectos del lenguaje de programación *java*.

Para que realmente esta solución pueda llegar a tener un espacio en empresas que se dedican a desarrollar software, se consideraron algunas caractarísticas y funcionalidades que se podrían incluir:

1. Implementación de un servidor elástico para soportar el tráfico y hacer rentable este proyecto. Es decir, tener un servidor que cambie de características (Memoría RAM, Capacidad de Disco y Procesador) según la demanda de tráfico y prcesos que tenga el servidor en el momento. Esto para atender a todos los clientes que requieren del servicio y evitar la caída del servidor a un costo reducido.

2. Incorporación de un detector de errores. En el caso que ocurra un error en el uso de la aplicación web, éste debe de brindar al usuario final un formulario para reportar el incidente y registrarlo en un archivo. Esto para ser atendido posteriormente y arregalo por un equipo de soporte.
3. Implementación de un sistema de retroalimentación, que sea capaz de capturar las necesidades del usuario final para auxiliar a generar posibles mejoras en la aplicación.
4. Liberación completo del lenguaje de programación *java*. Para este proyecto se establecieron algunas restricciones para utilizar el lenguaje y desarrollar software. Esto se puede evitar implementando un servidor con características superiores.
5. Adición de lenguajes de programación. Existen varios lenguajes de programación que son demandados en la industria de software, se pueden soportar esos lenguajes en la aplicación para aumentar el segmento de mercado.
6. Mejora de la revisión del código. Al momento de escribir una fracción de código, el sistema puede notificar al usuario el error u opciones de código en tiempo real.
7. Inclusión de servicios como *github*, *trello*, *jira*, *calendario de google* y entre otros para brindar una experiencia más cómoda al usuario final.
8. Uso de paquetes y carpetas dentro del proyecto. Debido a la falta de recursos computacionales en el servidor, se destacó el uso de paquetes en los proyectos. Con un servidor con mejor capacidad, se puede incluir la funcionalidad de usar paquetes y carpetas dentro del proyecto.

Referencias y Fuentes de Información

Introducción

[1] (2014). *Historia de Hardware y Software*. 17-Feb-2017, de Sitio web: https://informaticabasicakatty.files.wordpress.com/2012/10/historia-de-hardware-y-software.pdf.

[2] Rownski, D. (2013). *Google Play Hits One Million Android Apps.* 17-Feb-2017, de readwrite Sitio web: http://readwrite.com/2013/07/24/google-play-hits-one-million-android-apps.

[3] Warren, C. (2013). *Google Play Hits 1 Million Apps*. 17-Feb-2017, de Mashable Sitio web: http://mashable.com/2013/07/24/google-play-1-million/#zqokkt18ppqq. [Último acceso: 17-Feb-2017].

[3] Connie, G. *Android e iOS dominarán el mercado (casi) para siempre.* Forbes México.

[4] F. B. (2003). *Ingeniería del software*. Madrid, España: UOC, pp. 17-18.

[5] I. S. (2005). *Ingeniería del Software*. Madrid, España: Pearson Educación. p. 4.

[6] Canós, J. Letelier, P. Penadés, M. (2007). *Métodologías Ágiles en el Desarrollo de Software*. 13-Abr-2017, Sitio web: http://ima.udg.edu/docencia/07-08/3105200728/todoagil.pdf.

[7] Priolo, S. (2009). *Métodos ágiles*. España: USERSHOP, pp. 32-33.

[8] Jamsa, K. (2012). *Cloud Computing*. Estados Unidos: Jones & Bartlett Publishers.

Capítulo 1 Análisis del Sistema

[9] Meritxell, R. (2007). *Software Libre: empresa y administración en España y Cataluña*. España: UOC.

[10] Ortega, M. Rodríguez, J. Ruiz. *Informática industrial.* Universidad de Castilla-La Mancha, 1997. Editorial Ilustrada.

[11] Kendall, J. (2005). *Análisis y diseño de sistemas*. Estaods Unidos: Pearson educación,

[12] Cass, S. (2015). "The 2015 top ten programming languages." IEEE.

Amo, F. (2005). *Introducción a la Ingeniería del Software*. España: Delta Publicaciones.

Capítulo 2 Diseño del Sistema

R. Llanos. (2010). *Fundamentos de Informática y Programación en C*. España: Universidad de Valladolid, p. 273

Pressman, S. (2001). *Software Engineering a Practitioner's Approach*. Estados Unidos: McGraw-Hil.

Somerville, I. (2011). *Software Engineering. Estados Unidos:* Pearson Education.

Capítulo 3 Desarrollo del Sistema

Paneque, I. (2013). *Linux 4 You!*. Estados Unidos: Safe Creative, p. 343

Eschweiler, S. (2016). *A Practical Introduction to the new Web Development Platform Angular 2*. Estados Unidos: Leanpub.

(2016). *Lanzamiento de una máquina virtual de Linux con Amazon EC2*. 10-Ago-2017, de Amazon Web Services Sitio web: https://aws.amazon.com/es/getting-started/tutorials/launch-a-virtual-machine/
[13] Anicas, M. (2014). *How to Set Up a Node.js Application for Production on Ubuntu 14.04*. 25-Sep-2017, de Digital Ocean Sitio web: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-14-04

Capítulo 4 Pruebas del Sistema

R. Llanos. (2010). *Fundamentos de Informática y Programación en C*. España: Universidad de Valladolid, p. 273

Anexo

Glosario de Términos

En este glosario se enlistan todos los términos que se indican en la tesis escritos en letras Itálica, dichos términos se encuentran ordenados alfabéticamente únicamente por su letra inicial para su fácil acceso.

A

Angular	*Marco de trabajo soportado por Google. Usa TypeScript que es el superconjunto JavaScript y librerias externas para ofrecer facilidad en el desarrollo de aplicaciones web. Se adapta bien para crear aplicaciones de una única pantalla.*
Angular CLI	*Interfaz de línea de comando que facilita la creación de aplicaciones de Angular a través de comandos sencillos.*
Amazon Web Services	*Plataforma que brinda servicios como renta de base de datos, repositorios y máquinas virtuales.*
API	*Interfaz de programación de aplicaciones; conjunto de subrutinas, funciones y procedimientos que ofrece una biblioteca de programación para facilitar la comunicación entre programas.*
AS	*Arquitectura de Software.*

B

Backend	*Hace referencia a la parte del software que se encarga en el procesamiento de datos que entran por una interfaz de usuario.*
Body Parser	*Librería que permite la extracción de datos de forma ordenada de diferentes tipos de texto como JSON*
Bootstrap	*Marco de trabajo para la interfaz gráfica web.*

C

Callback	*La ejecución de una función después de que una tarea ha sido completada*
CDN	*Red global de distribución de contenido.*
Ciclo de vida del objeto	*Indica todas las fases por las que pasa un objeto instanciado de una clase.*
Ciclo For	*Estructura de control para iterar procesos en la programación*
CodeMirror	*Librería que convierte etiquetas de área de texto de html en editores de texto.*
Curl	*Librería de transferencia de archivos a través de url*

D

DCL	*Abreviación de Data Control Language. Lenguaje de base de datos que tiene como finalidad controlar el acceso a los datos con privilegios.*
DDL	*Abreviación de Data Definition Language. Lenguaje de base de datos que tiene como finalidad la definición de la estructura de datos.*
Dependencia Parcial	*Se refeire a que todos los atributos que no son una llave primaria, deben depender únicamente de una llave primaria.*
Digital Ocean	*Plataforma que brinda recursos e infraestructura computacionales como servicio.*
DML	*Abreviación de Data Manipulation Language. Lenguaje de base de datos que tiene como finalidad la manipulación de los datos persistentes en las bases de datos.*

E

Estándar	*Consiste en el establecimiento de normas que se aplican para ajustar procesos o actividades con el fin de ordenarlas y mejorarlas.*
ExecSync	*Librería que permite ejecutar instrucciones en línea de comandos de forma síncrona en Node Js*
Express	*Infraestructura de aplicación web para Node Js*

F

Firebase	*Plataforma de google que brinda servicios para facilitar desarrollo de aplicaciones en la nube a través de APIs.*
Formas Normales de Edgar F. Codd	*Reglas convencionales establecidas para evitar y disminuir problemas y redundancias y protegar la integridad en una base de datos.*

G

Git	*Sistema de versión de control.*
Git Hub	*Plataforma para alojar repositorios remotos haciendo uso de Git.*

I

IDE	*Ambiente de desarrollo integrado; aplicación informática para facilitar el desarrollo de software.*
IntelliJ IDEA	*Es un ambiente de desarrollo integrado desarrollado por JetBrains. Disponible en 2 ediciones, la de comunidad y la comercial. En el presente proyecto se trabajó con la edición de comunidad.*

J

Java	*Lenguaje de Programación orientado a objetos*
Javac	*Compilador del Lenguaje Java*
JQuery	*Es una biblioteca de javascript que permite manipular el DOM, manejar eventos y agregar interacción a través de AJAX, para ampliar las funciones de javascript.*
JSON	*Formato de texto ligero para el intercambio de datos*

K

KitKat	*Undécima versión del sistema operativo Android.*

L

Listener	*Es una servidor o aplicación en espera de una acción o petición en específico para llevar a cabo una respuesta.*

M

Modal	*Ventana de cuadro de diálog para informar al usuario.*
Modelo en Cascada	*Modelo más primitivo del ciclo de vida del software en el cual se identifican prácticamente todas las actividades que intervienen en el desarrollo y explotación de software.*
Modelo en "V"	*Modelo que se basa en una secuencia de fases análoga a la del modelo en cascada; se enfoca directamente en la importancia de la visión jerárquica que se tiene en las distintas partes del desarrollo de un sistema.*
Modelos Evolutivos	*Este modelo puede considerarse como un proceso iterativo, de manera que cada iteración se hace solo una parte del desarrollo lo que permite avanzar un poco en cada fase.*
Modelo de Entidad-Relación	*Es un modelo que representa las relaciones que existen entre todas las entidades que representan la base de datos en un sistema.*

N

NodeJs	*Entorno de ejecución para JavaScript del dalo del servidor.*
NoSQL	*Modelo de datos estructurados no relacionales.*

O

Offline	*Hace referencia a un estado de conectividad como en fuera de línea (sin conexión).*
Online	*Hace referencia a un estado de conectividad como en línea (con conexión).*
OSI	*Es el modelo de interconexión de sistemas abiertos, modelo de referencia para los protocolos de arquitectura en capas de la red.*

P

PM2	*Gestor de procesos para NodeJs.*
Promise	*Una Promesa es un proxy para un valor no necesariamente conocido en el momento que es creada la promesa. Permite asociar manejadores que actuarán asincrónicamente sobre un eventual valor en caso de éxito, o la razón de falla en caso de una falla.*

R

REST	*Arquitectura de software para hipermedia distribuidos.*
Rimraf	*Librería que permite eliminar carpetas desde NodeJs.*

S

Script	*Órdenes o instrucciones para un lenguaje de programación organizadas secuencialmente.*
SDK	*Kit de desarrollo de software para la plataforma Android.*
SOA	*Arquitectura Orientada a Servicios.*
Sublime Text 3	*Es un editor de texto y código fuente con soporte para múltiples lenguajes. No es un software libre o de código abierto.*
SW	*Servicio Web.*

T

Terminal Tonta	*Terminal que consiste en una interfaz para dar entrada y transmitir datos o desplegar a los mismos desde una computadora remota a la cual se está conectando.*
Thread	*Unidad básica de ejecución, también conocido como Hilo en español.*
TypeScript	*Lenguaje mantenido por microsoft. Permite la programación orientada a objetos con javascript.*

U

UML	*Lenguaje de Modelado Unificado. Utilizado para representar de manera gráfica especificaciones de un software.*

V

Visual Studio Code	*Editor de texto que está optimizado para el reconocimiento de TypeScript.*

Z

Zip	*Formato de compresión de datos.*