UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Integración de sistema de registros de salud con sistema administrador de agendas

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Julio Alberto Bastida Bernal

ASESOR DE INFORME

Ing. Luis Sergio Valencia Castro

Ciudad Universitaria, CDMX, 2019

Agradecimientos

A mis padres por todas sus enseñanzas y por su apoyo incondicional.

A Karen por su paciencia.

A la Universidad por todas las oportunidades de crecimiento.

A la Facultad de Ingeniería por forjar mi carácter.

Al PROTECO (y a los amigos que encontré ahí) por todo lo aprendido.

Al Ing. Heriberto Olguín Romo por compartir su filosofía.

Al Dr. Daniel Trejo por ser un excelente guía.

A Daniel Garibay por sus comentarios.

A Andrés y David por su gran ejemplo.

A Nimblr por darme una gran oportunidad.

A Eduardo Espinosa por sus breviarios culturales

Índice

Índice de ilustraciones

Índice de tablas

Objetivo

El objetivo del proyecto asignado fue integrar el sistema *Holly* con un nuevo proveedor de sistema electrónico de salud a través de una interfaz *HTTP*. Este proceso tuvo una duración de 6 meses, en el cual realicé un plan de trabajo basado en lo que aprendí en mis materias de: “Ingeniería de software” y “Administración de proyectos de software”. Durante la implementación de la integración utilicé también patrones de diseño, algoritmos y estructuras acorde con las clases de: “Algoritmos y estructuras de datos”, “Lenguajes de programación”, “Programación Avanzada” e “Inteligencia Artificail”.

Al implantar las estrategias y marcos de trabajo adquiridos en la facultad de ingeniería el proyecto se llevó a cabo exitosamente. Hoy en día se encuentra completamente operativo en un ambiente de producción atendiendo a más de 1500 usuarios por día.

Introducción

Este reporte contiene el recuento de 6 meses de trabajo en la empresa *Nimblr Inc*. en los cuales fui encargado de múltiples tareas propias de un ingeniero de software como: planear procesos de desarrollo de software, diseño de algoritmos, creación de código, ejecución de procesos de aseguramiento de calidad y análisis de ejecución de sistemas.

El primer capítulo contiene la historia de Nimblr Inc. y la problemática que busca resolver, así como una descripción detallada de su estructura, las dinámicas de trabajo y la identidad de la empresa. A pesar de ser una empresa muy joven (2 años) dentro del sector salud se encuentra en un momento de crecimiento muy interesante, lleno de retos técnicos y administrativos. Cabe destacar la descripción de los procesos existentes dentro del departamento de ingeniería, pues estos muestran el uso de buenas prácticas de la ingeniería para el desarrollo de código, también por parte del área de operaciones con el protocolo de incidencias del equipo.

El siguiente capítulo detalla el sistema computacional creado por Nimblr Inc., el cual busca resolver las problemáticas descritas en el capítulo 1. Se listan los distintos módulos que componen al sistema, junto con las características de estos a nivel técnico. Por último, se especifica el comportamiento de la plataforma de lado del usuario, y se muestran los beneficios con los que cuenta para él.

En el capítulo 3 se puede encontrar una lista de los conceptos necesarios para la realización de mis tareas, los cuales adquirí en la facultad de ingeniería de la UNAM. Estos conceptos son: servicios web, arquitectura de microservicios, patrones de diseño, paradigma de programación funcional y el marco de trabajo *Scrum*. Como se puede apreciar en este capítulo los conceptos permitieron la realización del proyecto que me fue encargado. Al inicio de este tercer capítulo se encuentran las menciones especiales a las materias que a mi parecer fueron fundamentales para mi formación como ingeniero, así como los temas más importantes para mi desarrollo laboral.

El cuarto capítulo describe el proyecto asignado y dirigido por mí dentro de la empresa, consistió en crear una integración de la plataforma de Nimblr Inc. con un Electronic Health Record (sistema electrónico de registros de salud, *EHR*). Se encuentra la planeación realizada para completar el proyecto de manera exitosa, incluyendo la identificación de las subtareas y el orden en que debían realizarse. También se incluyó una tabla con las estimaciones de tiempo obtenidas por el equipo mediante la técnica del póker de planeación. Las tareas identificadas fueron: Análisis técnico y comercial, diseño de software, implementación, ejecución, monitoreo y mantenimiento.

El resto del capítulo detalla cómo llevé a cabo cada subtarea, y con los problemas encontrados a los que les di solución. En esta parte del trabajo es donde se puede apreciar la relación con los conceptos hallados en el capítulo 3, y cuál fue la razón para utilizarlos, asimismo la manera en la que fueron aplicados en las diferentes etapas del proyecto.

La parte culminante del proyecto involucró realizar un proceso de certificación de la ley de seguridad de la información digital de salud. Esto resultó al final ser lo más complicado de todo el proyecto por el esfuerzo realizado por el equipo. Se invirtieron más de 100 horas hombre para poder completarla. La importancia de esta certificación radica en las oportunidades que abre para el aspecto legal con la expansión comercial de la empresa, en un marco completamente legal, también es una muestra de la madurez de la empresa como parte del sector salud.

Se cierra el capítulo con una descripción del proceso de retrospectiva y las conclusiones obtenidas.

Finalmente se presentan las conclusiones de la ejecución del proyecto y cómo este contribuyó a la continuación de mi formación profesional.

Capítulo 1
Nimblr Inc.

¿Qué es Nimblr Inc.?

Nimblr Inc. es una empresa fundada en Estados Unidos con el objetivo de traer soluciones al sector salud utilizando inteligencia artificial, a través de un asistente virtual (Notimex, 2017).

La organización cuenta con dos sedes. La oficina central se encuentra en Santa Mónica, California en EE. UU., ubicada en 730 Arizona Ave., C.P. 90401. Y la segunda sede se encuentra en Polanco, CDMX.

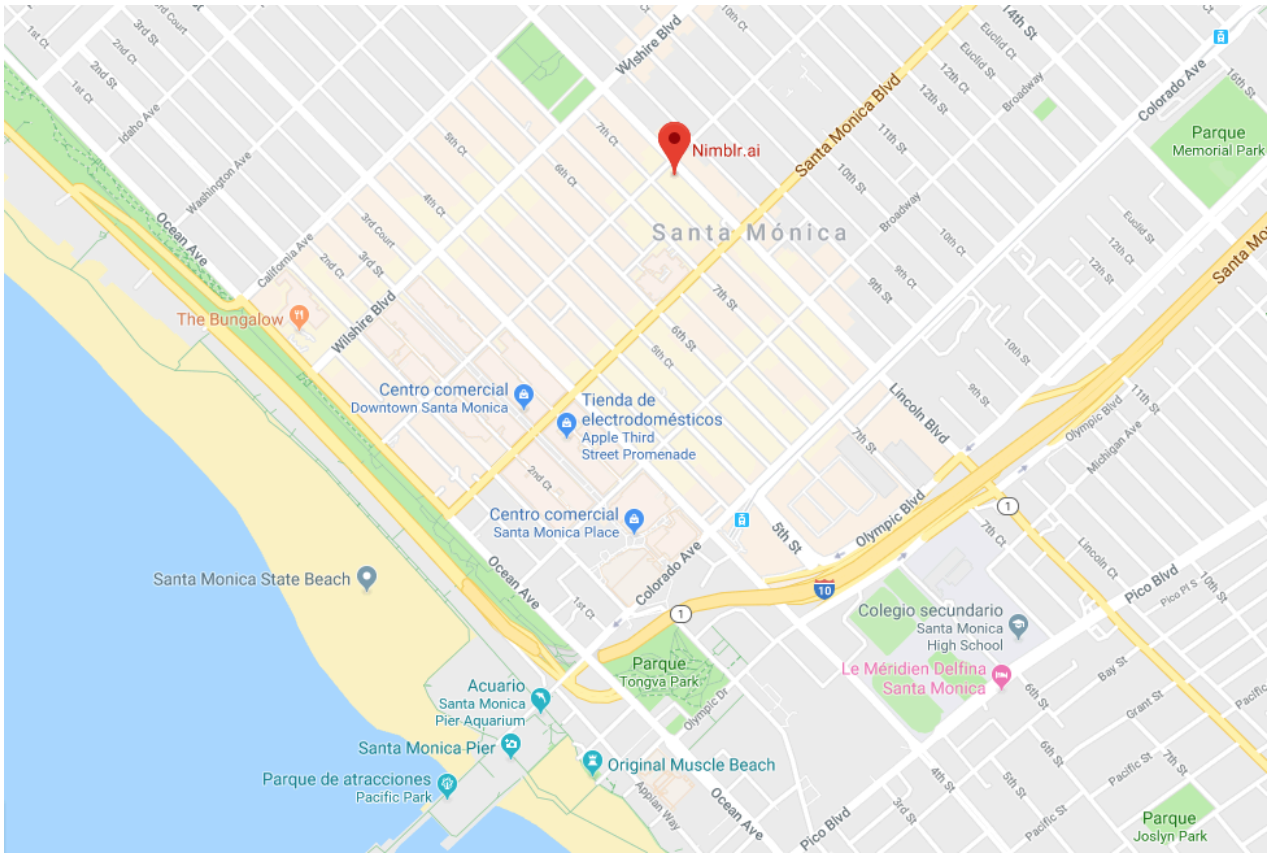

Ilustración 1 Ubicación de Nimblr Inc., Santa Mónica, CA. Fuente: Google Maps

HISTORIA

Nimblr Inc. es una empresa fundada en el año 2016 en California, EE. UU. por Juan Vera, Andrés Rodríguez, Silvanna Valencia y David Jiménez. Con una inversión inicial de $1.3 MDD (Crunchbase, 2017) y un equipo inicial de 4 personas lograron en 2 años posicionarse en el mercado mexicano y de EE. UU., con un flujo de miles de usuarios al día. Actualmente el equipo consiste en 12 personas, como se muestra en el organigrama del apartado 1.3

Dentro de los tipos de empresa Nimblr Inc. clasifica como *startup* (también llamada empresa emergente), es decir una organización que ofrece una solución inexistente en el mercado, además de tener expectativas de crecimiento rápido. Recibe capital de fondos de inversión especiales para *startups* y cuenta con un grupo de asesores especializados. El mayor reto para estas empresas es encontrar un modelo de negocio rentable, esto puede tardar varios años durante los cuales depende enteramente de inversores y/o programas de aceleramiento.

Por el hecho de ser una startup la dinámica de trabajo dentro de Nimblr Inc. es distinta a la de una empresa: la ejecución y entrega debe ser rápida, y debe ser capaz de adaptarse a los requerimientos del mercado en tiempo (casi) real. Por su potencial y buen desempeño comercial cuenta desde 2018 con el respaldo del *programa StartX* de la universidad de Stanford (Endeavor, 2018).

El programa StartX está diseñado para asesorar a emprendedores del sector salud y crear un ecosistema de soluciones en tecnologías de la información. Se realizan varios eventos a lo largo del año en California en donde los participantes pueden compartir sus experiencias y presentar avances de los proyectos. Adicionalmente se pueden obtener recursos económicos en función del crecimiento y el potencial de las soluciones.

ORGANIGRAMA

Estructuralmente Nimblr Inc. está dividida en 5 departamentos: gerencia, desarrollo, operaciones, ventas y marketing (el organigrama se muestra en la ilustración 2).

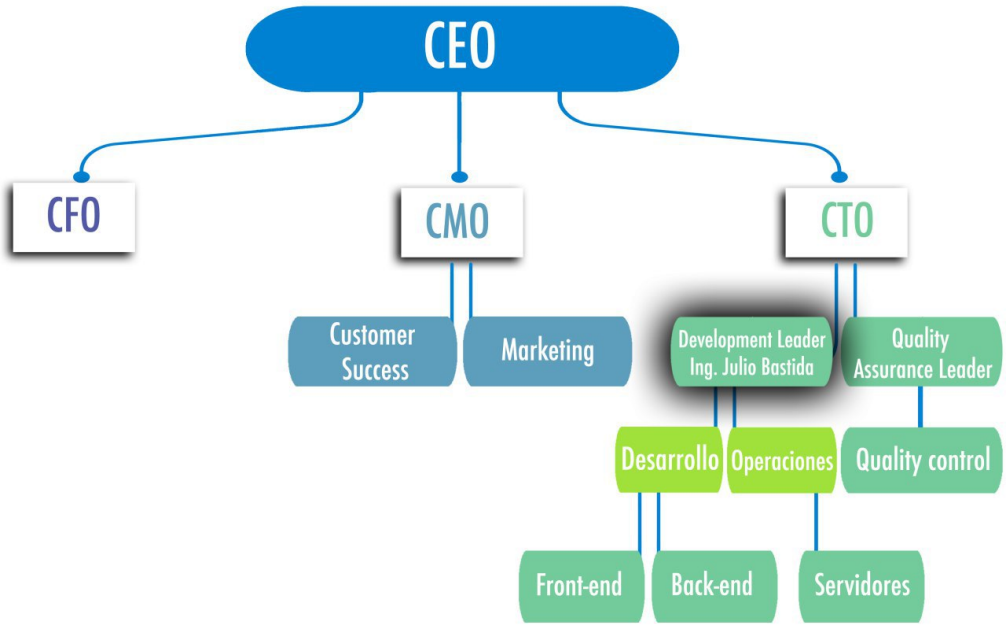

Ilustración 2 Organigrama de Nimblr Inc. Fuente: Elaboración del autor

1.1.1. Departamento de Ingeniería

El departamento de Ingeniería está encabezado por el *Chief Technology* Officer (*Jefe de Tecnología, CTO*). Él es responsable de hacer la planeación de desarrollo, la asignación de tareas y de ser mentor de los programadores más nuevos. El departamento se divide en 2 secciones:

- Desarrollo: Equipo a cargo de diseñar, implementar, probar y monitorear todas las funcionalidades del sistema.
- Operaciones: Equipo encargado de configurar y dar mantenimiento a la infraestructura del sistema, así como de crear herramientas de monitoreo automatizado y alertas.

Las pruebas automatizadas (unitarias y de integración) son una parte esencial en el proceso de desarrollo de Nimblr Inc., así como los procesos de revisión de pares y el trabajo del área de aseguramiento de calidad, de ello hablaré en el apartado 3.7.

1.1.1.1. Protocolo de incidencias

Dentro del departamento de operaciones existe un protocolo de manejo de incidencias, el cual define los procedimientos a seguir del equipo cuando ocurre un suceso, así como una clasificación. Una incidencia se define como un acontecimiento inesperado aquel que puede afectar el comportamiento normal o esperado del sistema. Esto incluye, por ejemplo: errores no manejados quienes interrumpen la ejecución de algún servicio, archivos de configuración con valores erróneos, servicios de terceros no disponibles, etc.

El protocolo asigna una prioridad a cada incidencia en función del número de usuarios afectados por la incidencia y el riesgo para el flujo de trabajo del sistema.

Se define también a un equipo responsable de controlar cualquier incidencia detectada. Controlar una incidencia significa crear un reporte y contactar a la persona adecuada para resolver la incidencia de manera oportuna.

La clasificación de incidencias está dada de la siguiente manera:

Prioridad	Descripción	Acción(es)
Alta	- Un gran número de usuarios se ve afectado por la incidencia - La incidencia afecta a alguna de las funcionalidades principales del sistema - Ejemplos: Los usuarios no pueden hacer *login*, el servidor no tiene suficiente *memoria RAM*	- Equipo responsable debe evaluar y controlar de manera inmediata - Interrumpir cualquier tarea y resolver de manera inmediata

Media	- Un solo usuario se ve afectado por la incidencia - La incidencia afecta a alguna de las funcionalidades secundarias del sistema - Ejemplos: El usuario X tiene preferencias incompatibles, el sistema tarda más de 1 minuto en mandar alertas internas	- Equipo responsable debe evaluar y controlar en un plazo máximo de 24 horas - Establecer un tiempo de solución no mayor a una semana
Baja	- La incidencia no pone en riesgo el flujo de trabajo del sistema - Ejemplos: los usuarios no pueden ver *emojis* en los reportes de actividad, la base de datos de respaldo no está disponible	- Equipo responsable debe evaluar y controlar en un plazo máximo de 72 horas - El tiempo de solución es a mediano plazo

Tabla 1 Clasificación de incidencias. Fuente: Elaboración del autor

DESCRIPCIÓN DEL PUESTO

Mi puesto inicial en la empresa fue *Software Engineer Jr. (Ingeniero de Software Jr.)* con las siguientes tareas:

- Diseñar soluciones y arquitecturas de software
- Construir *software front end* (código que se encarga de procesar información y mostrarla al usuario en el formato solicitado)

- Desarrollar conexiones a proveedores externos de servicios de calendario en línea
- Conectar *gateways (canal de comunicación)* externos de envíos de SMS
- Configurar infraestructura de nube en *Amazon Web Services (AWS)*
- Brindar soporte técnico

Durante los primeros 6 meses realicé todas estas tareas de manera eficiente y responsable. Además, aprendí a usar herramientas de software colaborativo (*git*). Mi proceso de desarrollo de software se hizo más maduro ya que debía seguir el flujo de trabajo (detallado en la siguiente sección) y las practicas definidas por el equipo, todo ello con el objetivo de entregar un producto de calidad.

A partir del sexto mes me asignaron el rol de **Desarrollador líder**, esto implicó responsabilidades adicionales:

- Planear las tareas del equipo desarrollo.
- Representar al equipo de desarrollo en las reuniones de gerentes.
- Asignar tareas.
- Monitorear la correcta ejecución del flujo de desarrollo.
- Realizar liberaciones mensuales.

Como desarrollador líder fui responsable de un equipo de 4 personas con el principal objetivo de incrementar la frecuencia en la entrega de funcionalidades. Para lograr esto propuse múltiples estrategias ante el área directiva, y al ser implementadas generaron buenos resultados. Estas estrategias fueron:

- Especializar a los miembros del equipo: Orientar a cada persona a un área específica de trabajo (front end, operaciones, reportes)
- Realizar reuniones uno-a-uno: Periódicamente tener reuniones cortas con cada uno de los integrantes para conocer su estado de ánimo con respecto al proyecto
- Establecer objetivos claros a corto plazo para cada miembro: Controlar el avance del proyecto mediante tareas a corto plazo (3 a 5 días)
- Crear una estrategia de liberación de código: diseñar un nuevo proceso de liberación siguiendo principios fundamentales de ingeniería de software (Pressman, 2010)

Con estas estrategias logré subir el número de funcionalidades liberadas y al convertirse los programadores en “expertos” de su área el código generado mejoró en cuanto a eficiencia y confiabilidad.

FILOSOFÍA

Misión
Nimblr Inc. es una empresa de tecnología con la intención de innovar al sector médico en México y Estados Unidos al introducir soluciones basadas en inteligencia artificial. Queremos contribuir a implementar esta tecnología a la práctica médica para hacer los procesos de agenda de citas más eficientes y con ello mejorar la relación entre doctores y pacientes.

Visión
Dentro de 5 años ser la empresa con mayor participación en la transformación del sector médico en México y América Latina ofreciendo excelencia, calidad y servicio a nuestros clientes, empleados y socios.

Objetivo
Generar una integración de lo último en tecnología en inteligencia artificial con los proveedores de servicios de salud para resolver los problemas más habituales que surgen en esta área.

El logo de la empresa se ilustra a continuación como referencia a su identidad corporativa:

Ilustración 3 Logotipo Nimblr Inc.

PROBLEMÁTICA

Las grandes pérdidas de tiempo y dinero en clínicas y hospitales, así como las dificultades en la comunicación médico-paciente. En Estados Unidos los pacientes que no se presentan a su cita le cuestan al sistema de salud más de $150 billones de dólares anualmente (Kheirkhah, 2016). Cada espacio no ocupado se convierte en tiempo muerto para la clínica y a esto se le suman las horas de los asistentes y las recepcionistas, porque estos lo utilizan en llamadas de confirmación y recalendarización. La solución propuesta es Holly, un sistema autónomo con las tareas de comunicación doctor-paciente.

Capítulo 2
HOLLY

2.1 Chatbot

Un *chatbot* (o también llamado *bot conversacional*) es un sistema con el cuál se puede establecer una conversación vía texto o de manera oral. Los motivos y utilidades son muy variados porque son aplicados en diferentes escenarios. Una característica principal de un chatbot es generar una interacción con el interlocutor lo más social posible, es decir lo más real y acercado a una persona.

Los primeros ejemplos conocidos de chatbot datan de 1966 (Weizenbaum, 1966) y siempre ha existido un interés por ellos dentro de la computación. Su desarrollo ha ido de la mano con varios avances en la investigación de inteligencia artificial.

Los sistemas de chatbot pueden clasificarse de distintas maneras:

- Por tipo de interacción:
    - De soporte: Se encargan de dar datos de la empresa, así como de guiar a los usuarios en los procesos posibles del sistema.
    - De habilidad (*skill chatbot*): Es capaz de ejecutar órdenes (“Apaga la luz”, “Sube el volumen”), opcionalmente utiliza un contexto para darle precisión a las acciones (“las luces de la sala o de la cocina”).
    - Asistente: Su objetivo es ser un intermediario entre el usuario y un conjunto selecto de servicios (restaurantes, clínicas).

- Por alcance:
    - De propósito general: Está diseñado para poder entender y responder cualquier entrada posible. Ejemplos: Siri, Cortana, Amazon Alexa.
    - De propósito específico: Trabaja con un conjunto bien definido de temas y entradas aceptadas, por lo general trabajan con guiones (scripts).

- Por tipo de procesamiento de lenguaje:
    - Instrucciones preestablecidas: Utilizan un banco de palabras, las cuales disparan respuestas específicas, un ejemplo sería: el chatbot al detectar la palabra “mamá” dentro de una oración responde con “cuéntame más de tu familia”.

- Identificación de intenciones: Busca identificar los componentes en las oraciones de entrada (verbo, sujeto, complemento) para entender la entrada y poder dar una respuesta adecuada.

2.2 ¿Qué es Holly?

Holly es el sistema central y el producto de Nimblr Inc., este sistema tiene las siguientes funcionalidades:

- Lee, cancela y programa citas en agendas médicas.
- Envía y recibe mensajes por diferentes canales de comunicación (SMS, WhatsApp, Facebook, etc.) de doctor a paciente y viceversa.
- Envía reportes a los usuarios

Combinando estas funcionalidades Holly maximiza el tiempo del consultorio y reduce la cantidad de pacientes ausentes.

Para el consultorio médico, Holly es una asistente virtual, un empleado más a un costo muy bajo. Gracias a la arquitectura del sistema, se puede clasificar como “aplicación invisible”, es decir: ni los consultorios, ni los pacientes deben instalar nada para utilizar a Holly .

Algunas de las ventajas para los usuarios (doctores y pacientes) al utilizar a Holly son:

- Recordatorio oportuno de citas con datos de la clínica.
- Cancelar y/o reagendar citas.
- Recibir retroalimentación para la clínica.
- Notificar de cambios en la cita.
- Dar información básica de la clínica.
- Agregar citas al calendario del disipativo del paciente
- Recibir reportes periódicos con estadísticas de interés para las clínicas

Ilustración 4 Holly. Fuente: (Nimblr Inc, 2016)

El único requisito para poder utilizar a *Holly* es tener una cuenta en alguno de los proveedores soportados.

Para la comunicación con pacientes existe un módulo de chatbot el cual envía mensajes a través de distintos canales de conversación en tiempo real: SMS, WhatsApp, Facebook Messenger, Telegram, Amazon Alexa y más. El objetivo es tener una comunicación lo más natural posible, por eso el procesamiento del lenguaje natural es una parte integral de Holly, para ello se han implementado múltiples algoritmos conocidos dentro del campo de la inteligencia artificial.

El conjunto de capacidades de este módulo está bien definido, por lo que es posible controlar el flujo de las conversaciones utilizando guiones con un principio y un fin completamente demarcado. Tomando en cuenta lo anterior, el chatbot se puede clasificar como chatbot de propósito específico con procesamiento de lenguaje natural.
A nivel de implementación Holly está basada en una arquitectura de microservicios, los cuales fomentan un bajo acoplamiento entre sus diferentes módulos y da la facilidad de escalar cada uno conforme el uso lo demande.

Actualmente Holly da servicio a más de 200 clínicas y conversa con más de 6000 pacientes todos los días.

Capítulo 3
ANTECEDENTES

Para la realización de mis tareas fue necesario recurrir a las bases teóricas y prácticas obtenidas durante mi carrera en la Facultad de Ingeniería. Desde la aplicación del pensamiento lógico matemático en la resolución de problemas hasta la implementación de conceptos particulares de la teoría de la computación.

A lo largo de este capítulo presento una lista de algunos conceptos claves en mi desenvolvimiento para el mundo laboral. Estos conocimientos los obtuve en materias como: Algoritmos y estructuras de datos, Redes de datos, Administración de proyectos de software, Lenguajes de programación, Ingeniería de software, Inteligencia Artificial, Probabilidad, Estadística y Cálculo. Es notable la importancia de tener bases teóricas sólidas, así como la importancia de la práctica estructurada y continua, aspectos en los que se distingue la Facultad de Ingeniería.

De las materias mencionadas arriba las más relevantes para mi trabajo fueron sin duda Ingeniería de software y algoritmos y estructuras de datos.

Ingeniería de software incluye temas como: fases de desarrollo de software, análisis de requerimientos, metodologías de administración de proyectos, documentación, planeación y estimación de tareas, control de riesgos, estándares de la industria y presupuestos. Llevar a cabo un proyecto de software suele ser complejo debido a todas las variables controladas y no controladas involucradas en el proceso, para minimizar los riesgos y aumentar la probabilidad de éxito es necesario aplicar estas herramientas de la ingeniería de software de manera correcta y además apegarse a los estándares y buenas prácticas del área. Con estas herramientas pude definir los procesos de trabajo más adecuados para realizar mis tareas y generar métricas de calidad de mis resultados.

En algoritmos y estructuras de datos aprendí: análisis de complejidad de algoritmos; teoría de computación; algoritmos de búsqueda, ordenamiento y recursivos; así como

implementación de árboles, listas ligadas, pilas, colas, índices, mapas y conjuntos. La importancia de conocer estos temas radica en la creación de código eficiente, conocido y bien probado y le dan al programador las herramientas para crear nuevas estructuras y métodos de manipulación óptimos.

Estas dos materias son una piedra angular en la formación de un ingeniero de software ya que abarcan lo técnico-científico y lo administrativo de manera formal.

3.1 Servicios Web

En la arquitectura cliente-servidor un servicio web es un servicio de software ejecutado de manera remota para obtener información de un sistema externo (W3C, 2004) y que opcionalmente puede recibir parámetros de entrada. Es posible pensar en un servicio web como una función remota. Un servicio web debe soportar por lo menos un protocolo de comunicación, siendo el más común HTTP y por lo menos un formato de respuesta (XML, HTML, JSON, texto plano). Comúnmente se denomina *Application Programming Interface (API)* a una colección de servicios web con un propósito en común, aunque el término API se utiliza en otros contextos como: el conjunto de métodos de una biblioteca externa, el código base de una plataforma de desarrollo, las funciones disponibles de un sistema operativo.

Los servicios web se pueden implementar de muchas maneras, pero al final se elige aquella cuya resolución del problema sea óptima. Ejemplos notables de arquitecturas en servicios web: *Simple Object Access Protocol (SOAP, Protocolo simple de acceso a objetos),* REST*, Remote Procedure Calling (RPC, Procedimiento remoto de llamada).*

Servicio web *REST*: es una arquitectura para la implementación de servicios web basado en la definición del estándar HTTP. *REST* introduce el concepto clave de ***recurso*** (Lucchi, Millot, & Elfers, 2008), de manera simple un recurso es información alojado en otro sistema y representa el estado de una entidad: un carro, una casa, un usuario.

Las principales características de esta arquitectura son:

- Aplicación sin estado: el servidor nunca guarda información acerca del estado del cliente, esto garantiza operaciones idempotentes, es decir se pueden ejecutar de manera continua y siempre obtener el mismo resultado.

- Interfaz uniforme: los clientes hacen peticiones haciendo referencia a recursos y no a funcionalidades, por eso las urls encontradas en estas arquitecturas se ven así:
    - /coche/4/comprar
    - /usuario/1001/habilitar

- Cacheability: los clientes pueden decidir almacenar algunas de las respuestas de la API de su lado, siempre y cuando sea apropiado para el servidor. Guardar este tipo de información puede incrementar la velocidad general de ambos sistemas (cliente y servidor).

Como mencioné anteriormente, REST se basa por completo en el estándar HTTP, esto se refiere a la existencia de un “idioma” bien conocido el cual facilita crear y utilizar APIs. Por ejemplo, cada verbo HTTP implica una operación distinta:

- GET: solicita información de un conjunto o un conjunto de recursos
- POST: crea un nuevo recurso con la información provista en la petición
- PUT: reemplaza un recurso con la información provista en la petición
- PATCH: actualiza un recurso con la información provista en la petición
- DELETE: elimina uno o varios recursos

HTTP contempla otros verbos, aunque estos son lo más utilizados.

También son de gran importancia los códigos de respuesta pues permiten la correcta automatización en la comunicación de sistemas:

- 200 - 299: la petición se ejecutó exitosamente
- 400 - 499: la petición no se puede ejecutar porque la información provista es incorrecta
- 500 - 599: la petición no se puede ejecutar debido a un error interno del sistema

Actualmente la plataforma tiene integraciones con más de 20 APIs, una de ellas es la de Google Calendars. A continuación, se encuentra parte de la documentación de esta API

Events

For Events Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/calendar/v3, unless otherwise noted		
delete	`DELETE /calendars/calendarId/events/eventId`	Deletes an event.
get	`GET /calendars/calendarId/events/eventId`	Returns an event.
import	`POST /calendars/calendarId/events/import`	Imports an event. This operation is used to add a private copy of an existing event to a calendar.
insert	`POST /calendars/calendarId/events`	Creates an event.
instances	`GET /calendars/calendarId/events/eventId/instances`	Returns instances of the specified recurring event.
list	`GET /calendars/calendarId/events`	Returns events on the specified calendar.
move	`POST /calendars/calendarId/events/eventId/move`	Moves an event to another calendar, i.e. changes an event's organizer. **Required query parameters:** `destination`
patch	`PATCH /calendars/calendarId/events/eventId`	Updates an event. This method supports patch semantics. The field values you specify replace the existing values. Fields that you don't specify in the request remain unchanged. Array fields, if specified, overwrite the existing arrays; this discards any previous array elements.
quickAdd	`POST /calendars/calendarId/events/quickAdd`	Creates an event based on a simple text string. **Required query parameters:** `text`
update	`PUT /calendars/calendarId/events/eventId`	Updates an event.
watch	`POST /calendars/calendarId/events/watch`	Watch for changes to Events resources.

Ilustración 5 API de operaciones con citas (Google Inc, 2018)

3.2 Microservicios

La arquitectura de microservicios consiste en estructurar los módulos de un sistema como servicios independientes cada uno con un propósito específico (Dragoni, y otros, 2017). Los sistemas diseñados de esta forma tienen un bajo acoplamiento en el nivel de abstracción más alto y es sencillo agregar o modificar a sus partes. Debido al nivel de separación de los componentes también es posible escalarlos de manera individual (alta

granularidad). Los componentes pueden comunicarse entre sí utilizando algún protocolo preestablecido (como HTTP) para completar sus tareas.

Es deseable para una arquitectura de microservicios tener las siguientes características (Amazon Web Services, Inc, 2016):

- Descentralización: no existe un sistema central controlador, ni un esquema unificado de datos. Es decir, cada microservicio interpreta el modelo de datos como más le convenga.
- Independencia: un microservicio puede cambiar o ser reemplazado sin afectar al resto de los microservicios. Por ejemplo, si un microservicio falla los demás pueden seguir trabajando.
- Responsabilidad única: cada microservicio está enfocado en resolver una sola tarea o sólo un tipo de tareas.
- Caja negra: los detalles de implementación no son conocidos fuera de cada microservicio, cada uno decide el nivel de visibilidad mediante una interfaz
- Poliglota: cada microservicio está constituido por un conjunto diferente de herramientas, como un lenguaje de programación, frameworks, bibliotecas externas. De esta manera los desarrolladores tienen la libertad de siempre usar la herramienta correcta para el trabajo.

En algunos casos se tiene además una interfaz de acceso (API), el cual combina las funcionalidades de los módulos para así poder generar la información solicitada desde algún sistema exterior.

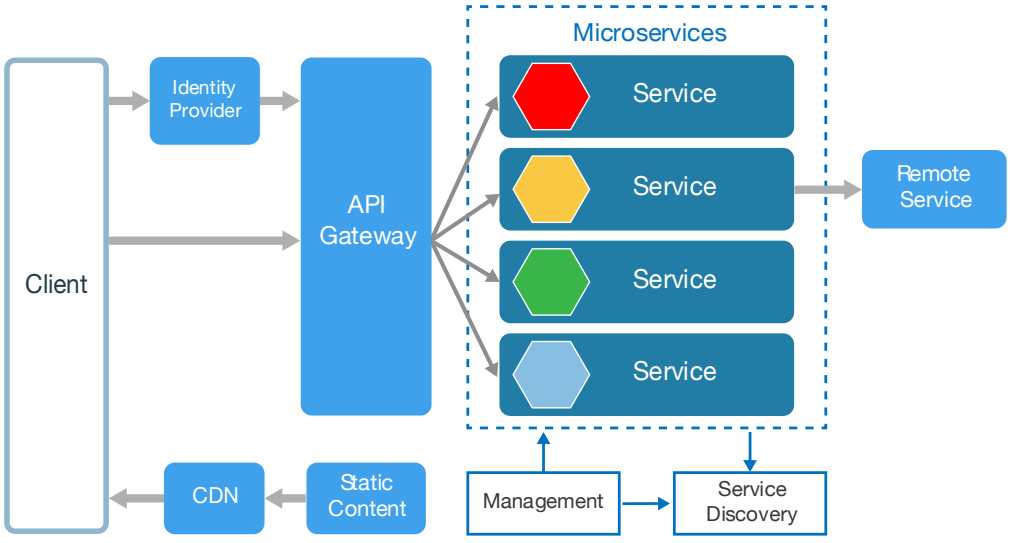

Ilustración 6 Ejemplo de arquitectura de microservicios (Microsoft, 2018)

La plataforma de Nimblr Inc. está compuesta de microservicios, por ejemplo: mensajería, lectura/escritura de agendas, generación de reportes, procesamiento de lenguaje natural. Actualmente cada desarrollador es responsable de un microservicio diferente y cuando se necesita un nuevo microservicio el equipo decide el conjunto de tecnologías a utilizar y se asigna un nuevo responsable.

3.3 Patrones de diseño

Los patrones de diseño son buenas prácticas para seguir en el desarrollo de software. Estas prácticas se han discutido desde hace ya varias décadas y tuvieron su auge en 1994 con el libro "*Design Patterns: Elements of Reusable Object-Oriented Software*" de la 'Banda de los 4' (Gamma, 1994). La idea detrás de adoptar estos patrones es generar código mantenible y escalable, de esta manera se pueden evitar los problemas más comunes en proyectos de software con muchos componentes.

Los patrones de diseño que utilicé fueron:

- Fachada: consiste en enmascarar un subsistema (o subconjunto de operaciones) complejo mediante una interfaz "fachada". Esta interfaz agrega un nivel de abstracción con operaciones sencillas de entender (Erich, y otros, 1994)

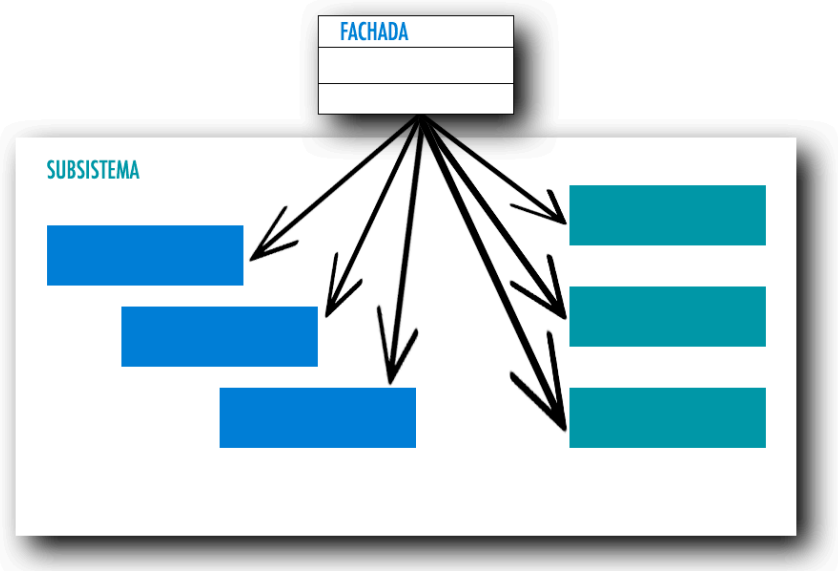

Ilustración 7 Patrón de diseño (Fachada). Fuente: Elaboración del autor

- Inyección de dependencias: en este patrón se tiene un servicio constructor o inyector encargado de crear, configurar y conectar a los elementos del sistema (Kocsis & Swan, 2017). El objetivo de esto es dejar la responsabilidad de la configuración de objetos en una entidad separada de la lógica de negocios, y facilitar la manera de selección con las implementaciones a usar en tiempo de ejecución (polimorfismo).

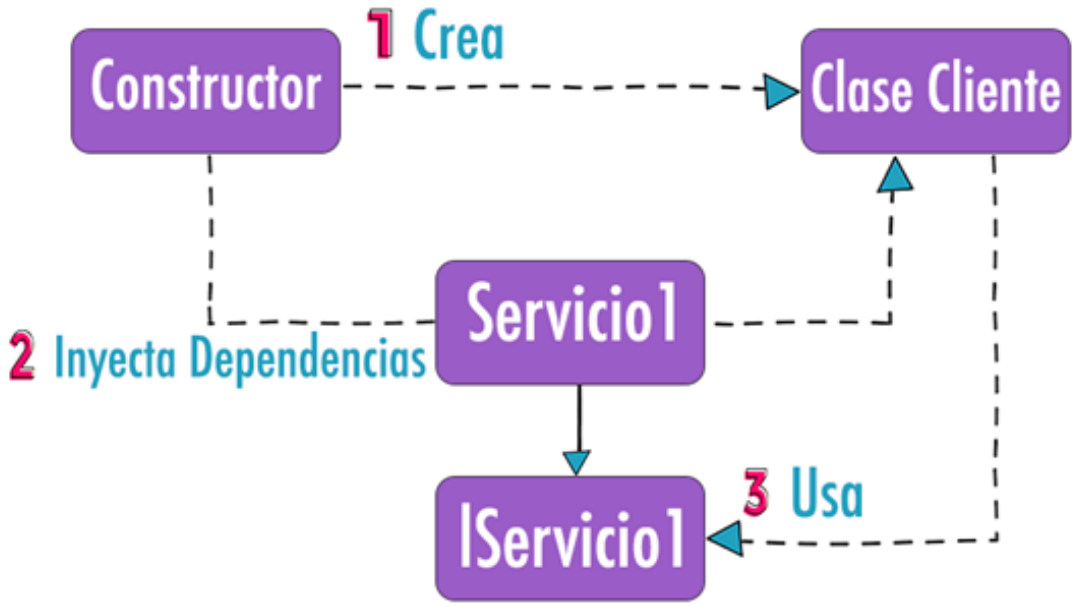

Ilustración 8 Inyección de dependencia. *Fuente: Elaboración del autor*

3.4 Programación funcional

El paradigma de programación funcional fue propuesto desde el nacimiento de la computación, con base en la definición matemática teórica de lo que es una función: una relación entre elementos de uno o más conjuntos, con su mayor utilización en trabajos de investigación académicos, aunque existen también ejemplos de sistemas empresariales con base en este paradigma (AutoCAD es uno de ellos) (Hughes, 1989).

El mayor contraste con la programación imperativa es la manipulación del estado en un sistema. En el paradigma imperativo de las asignaciones son operaciones primarias para cambiar el estado del sistema, en cambio en la programación funcional se hace gran énfasis en la inmutabilidad de la información y las estructuras. Esto se refleja en el código mediante el principio de las funciones cuyo dictamen es no alterar nunca los parámetros dentro una función.

A continuación, se muestra la diferencia entre una función regular y una función pura:

```
function porEscalar(vector, escalar) {
    vector.x = vector.x * escalar;
    vector.y = vector.y * escalar;
}
```

Ilustración 9 *Función no pura – modifica los argumentos.* Fuente: Elaboración del autor

```
function porEscalar(vector, escalar) {
    let resultado = {
        x: vector.x * escalar,
        y: vector.y * escalar
    };
    return resultado;
}
```

Ilustración 10 Función pura – coloca el resultado en un objeto nuevo. Fuente: Elaboración del autor

En las siguientes imágenes se muestra la diferencia entre código imperativo y código declarativo

```
let arr = [1,2,3,4,5];
let suma = 0;
for (let i = 0; i < arr.length ; i++) {
    if (arr[i] % 2 !== 0) continue;
    suma += arr[i];
}
```

Ilustración 11 Ejemplo de código imperativo para sumar los números pares de un arreglo. Fuente: Elaboración del autor

```
let arr = [1,2,3,4,5];
let suma = arr.filter(n => n % 2 !== 0)
              .reduce((n, acc) => n + acc, 0);
```

Ilustración 12 *Ejemplo de código declarativo para sumar los números pares de un arreglo. Fuente: Elaboración del autor*

Los beneficios de utilizar este paradigma son: código más legible y conciso (estructura declarativa), ejecución predecible (para las mismas entradas siempre se obtendrán los mismos resultados) y funciones reutilizables (funciones con responsabilidad genérica definida).

Aunque la estructura general de la plataforma está definida por clases, objetos y métodos, dentro de mis algoritmos me apegue a los principios de la programación funcional utilizando métodos declarativos como: *filter (filtro), map (mapa)* y *reduce (reducir).* De modo que di preferencia al uso de estructuras inmutables a través de métodos auxiliares para clonar objetos y creando funciones con un solo propósito.

3.5 Cola con prioridad

La cola es una de las estructuras más conocidas y utilizadas en el mundo de la programación. El concepto es muy sencillo: una colección en donde se pueden insertar elementos (tamaño dinámico) para después consumirlos siempre en el mismo orden de recepción. En algunas bibliografías se encuentran referencias a esta estructura con los términos First-In,First-Out y First-Come,First-Served. (Sedgewick & Wayne, 2016)

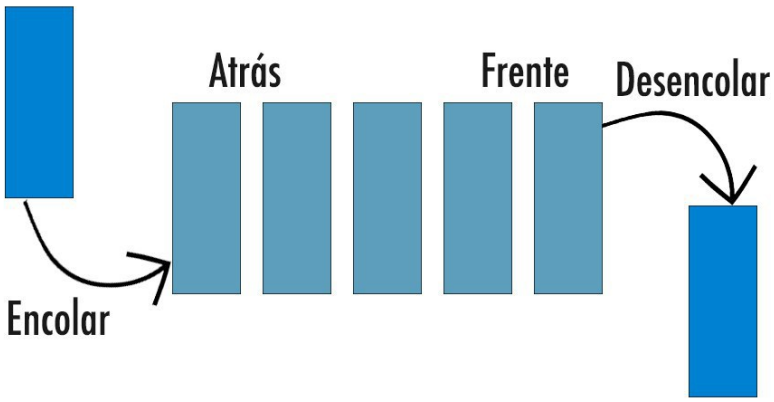

Ilustración 13 *Cola con Prioridad. Fuente: Elaboración del autor*

Una cola con prioridad trabaja bajo el mismo principio de la cola tradicional. Con la característica adicional de otorgar una prioridad a cada nodo, el cual es un número entero asignado al momento de ingresar. (Cormen, Leiserson, Rivest, & Stein, n.d.)

Para consumir un elemento en la cola se toma en cuenta la subcola de mayor prioridad y se selecciona el primer elemento insertado. La cola con prioridad es útil cuando se requiere tener un control de tareas paralelas, pero existen algunas con preferencia y siempre se ejecutan primero.

Para la implementación es común utilizar un árbol binario especial llamado montículo (*heap* en inglés). En un montículo, el valor de los nodos hijo deben ser siempre mayor al valor del nodo padre, gracias a esta condición se vuelve muy sencillo determinar cuál es el siguiente elemento por consumir.

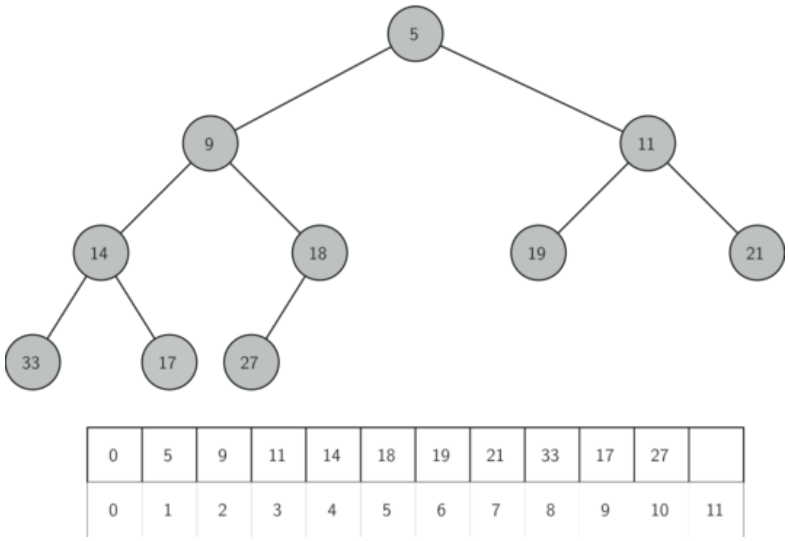

Ilustración 14 Organización de nodos en árbol. Fuente: (Bradfield School of computer science)

La complejidad de las distintas operaciones a hacer con la cola se muestra a continuación en notación O grande:

Operación	Complejidad
Búsqueda	O(1)
Eliminar elemento	O(log n)
Insertar	O(log n)

Tabla 2 Complejidad de operaciones para cola con prioridad. Fuente: Elaboración del autor

En algunas API se establecen límites en cuanto al número de peticiones permitidas por segundo, por esto fue necesario tener una estructura la cual permitiera tener control de la cantidad y el orden de las peticiones salientes.

3.6 Administración y procesos de desarrollo

Scrum (Academy, 2015): es un marco de trabajo de administración de proyectos principalmente utilizado dentro del mundo de desarrollo de software (también está siendo adoptado en otras áreas). Una característica importante de *scrum* es la forma en la entrega de soluciones al cliente. El producto se construye en varias fases incrementales, en contraste con el desarrollo clásico en donde el producto se entrega completo después de un periodo prolongado. El desarrollo incremental permite al cliente dar retroalimentación y cambiar los requerimientos para que el producto final siempre resuelva sus necesidades.

Scrum define 3 roles:

- *Product owner (Dueño del producto):* representa los intereses de los usuarios (o de algún cliente en particular). Define las necesidades a cubrir del producto y vigila el proceso de desarrollo. Esta persona puede ser parte de la organización del cliente o puede ser un integrante de la empresa desarrolladora. A diferencia de otros modelos de desarrollo, scrum necesita al PO involucrado en todas las fases del proyecto para garantizar que se cumplan las expectativas del usuario final en el proyecto.
- *Scrum team (Equipo Scrum)*: equipo de desarrollo ejecutor del proyecto. Desde la perspectiva de *scrum* el equipo de desarrollo debe de hacer las estimaciones de todas las tareas (en contraste a otros marcos de trabajo en donde el gerente o el cliente definen los tiempos).
- *Scrum master (Maestro Scrum)*: es el encargado de mantener la comunicación con el product owner y de supervisar el avance del scrum team. Además, debe de planear las iteraciones del proyecto.

Además de estos 3 roles, scrum define sprints: periodos cortos (2 a 6 semanas) en donde el equipo de desarrollo debe completar un conjunto de tareas para avanzar hacia el producto final.

El scrum máster es el encargado de planear los sprints y asegurarse de completarlos correctamente. Al inicio de cada sprint el scrum team selecciona las tareas a realizar en

función de la duración de éste haciendo una estimación conjunta para cada tarea. Al finalizar el sprint, el product owner revisa el estado del producto y da retroalimentación, la cual se utiliza para realizar la planeación del siguiente sprint.

Scrum también contempla dentro del proceso de planeación la estimación de tiempo de tareas, para esto existen varias técnicas ágiles (Grenning, 2002), comúnmente se utilizan las siguientes:

- Póker de planeación: se elige una tarea y cada miembro del equipo de desarrollo escribe de manera secreta en un papel el número de horas estimadas para esa tarea. Cuando todos los desarrolladores tienen una estimación la revelan simultáneamente y discuten hasta llegar a un consenso de la estimación final.
- Masa relativa: el equipo ordena las tareas por complejidad. Se comienza seleccionando la tarea más sencilla y la más compleja, todas las demás tareas se ordenan de manera relativa a estas.

En el caso de Nimblr Inc. el trabajo por parte de la administración y procesos de desarrollo se rige por Scrum. Esto permite revisar y actualizar el plan de trabajo de los desarrolladores de manera rápida, además de tener una orientación hacia objetivos específicos. Los roles en el equipo son:

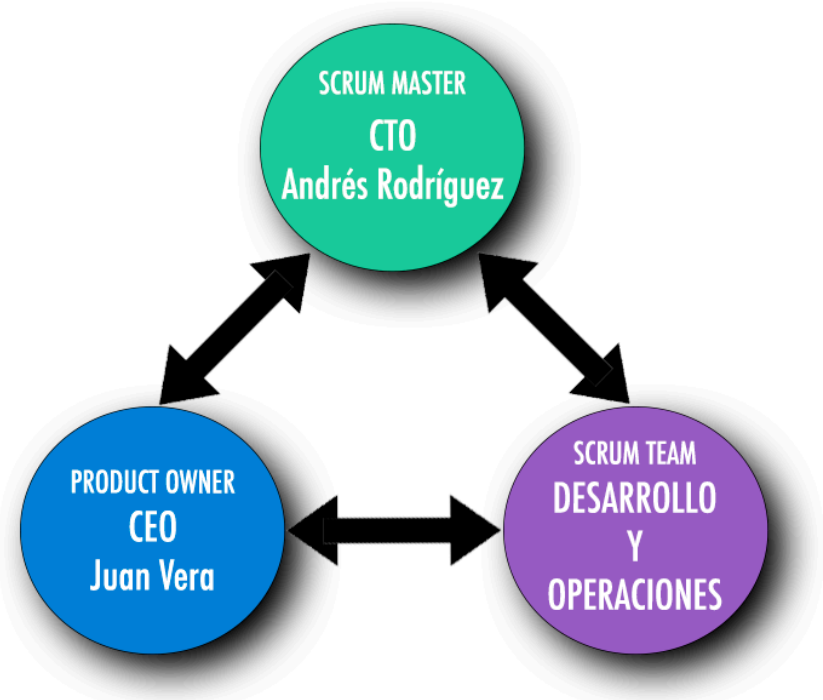

Ilustración 15 Roles de equipo con Scrum en Nimblr Inc. Fuente: Elaboración del autor

El siguiente diagrama muestra el flujo de desarrollo de software dentro de la organización con la metodología de *Scrum*:

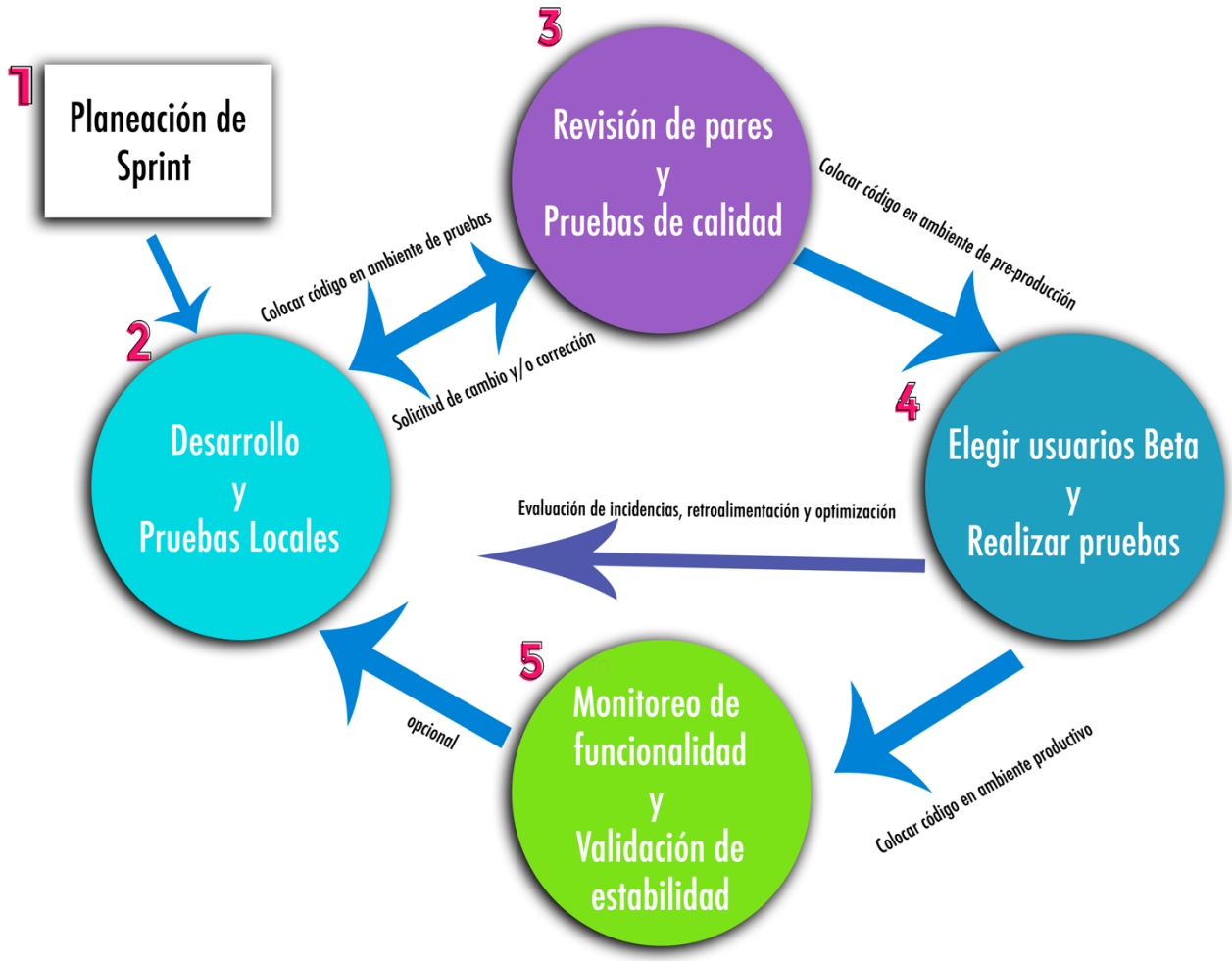

Ilustración 16 Diagrama de flujo de desarrollo de software con Scrum en Nimblr Inc. Fuente: Elaboración del autor

3.7 Desarrollo orientado a pruebas

Las pruebas automatizadas son una parte fundamental del desarrollo de software ya que dan un grado de confiabilidad en el código presentado, además representan el comportamiento esperado de los componentes del sistema. El flujo clásico de desarrollo es de la siguiente manera:

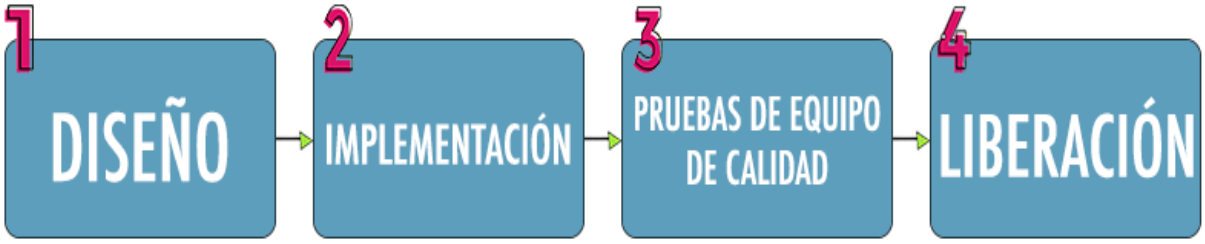

Ilustración 17 Flujo de desarrollo "clásico". Fuente: Elaboración del autor

Esta forma de trabajo es la más común entre los programadores, pero tiene complicaciones. Es difícil pensar en todos los casos de uso al momento de implementar y normalmente el programador sólo contempla aquellos casos de uso más sencillos, solamente para regresar al código una vez que el equipo de calidad determina un código está incompleto o incorrecto.

En la metodología del desarrollo orientado a pruebas el flujo se ve así:

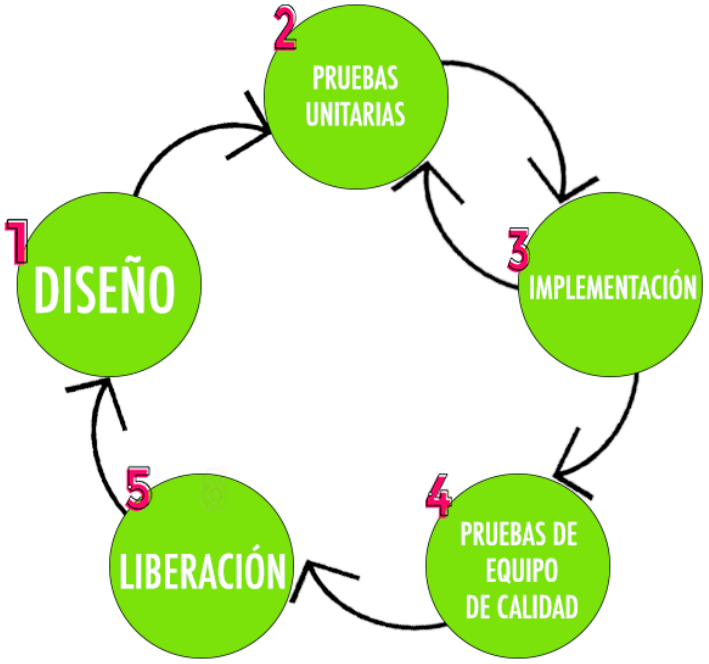

Ilustración 18 Proceso de desarrollo de código. Fuente: Elaboración del autor

El único cambio con respecto al flujo original es la fase de pruebas antes de la implementación. El propósito para el programador es pensar en la mayor cantidad de

casos de uso posibles y los escriba en pruebas unitarias. Al tener las pruebas unitarias se implementa el código con el objetivo de ejecutar exitosamente. Este es un proceso iterativo, porque normalmente es necesario modificar la implementación varias veces hasta llegar a ejecutar exitosamente las pruebas unitarias.

Las pruebas unitarias existentes del sistema funcionaron como guía para agilizar el desarrollo de código. En los casos donde fue necesario agregar nuevas funcionalidades en Holly siempre se comienza por la implementación de pruebas unitarias suficientes para asegurar la calidad del código. Las pruebas unitarias se usan también para obtener parámetros de calidad de software como:

- Cobertura de código: indica el valor del porcentaje de código ejecutado durante la elaboración de pruebas unitarias. Un valor alto indica que la mayoría de las ramas de ejecución han sido probadas por lo menos una vez. Este es un ejemplo de una gráfica de cobertura de código a través del tiempo

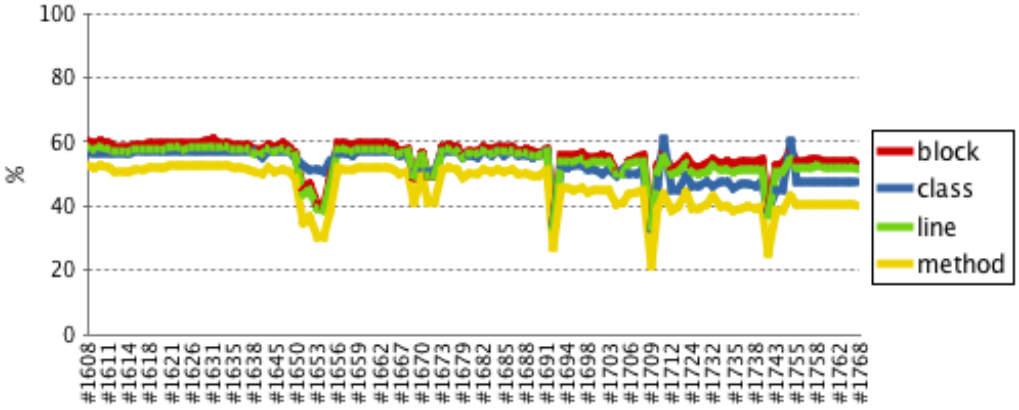

Ilustración 19 Reporte de cobertura de código. Fuente: (Campbel, 2012)

- Eficiencia: las herramientas de pruebas unitarias también muestran los bloques de código con tiempos de ejecución más altos. Gracias a esto se pueden diseñar pruebas de estrés con grandes volúmenes de información y encontrar aquellas partes del código causales de cuellos de botella en la ejecución del sistema.

De manera adicional a las pruebas en el equipo seguimos las siguientes prácticas de aseguramiento de calidad de software:

- Múltiples entornos de pruebas: antes de liberar una nueva funcionalidad al ambiente de producción se coloca en un servidor de pruebas en donde el equipo de QA está constantemente probando los casos de uso de la plataforma.
- Revisión de pares: todo el código debe ser revisado y aprobado en su totalidad por 2 desarrolladores distintos al responsable, con esto disminuyen la cantidad de errores y se optimiza el tiempo de todo el equipo.
- Pruebas de caja negra: el equipo de aseguramiento de calidad prueba todas las funcionalidades sin tener conocimiento de la implementación.

Capítulo 4
Reporte: Electronic Health Record

Un EHR es un sistema de almacenamiento de datos médicos y clínicos en formato digital (Gunter, 2005). Los hospitales y clínicas utilizan a los EHR para llevar el historial médico de pacientes, generar recetas certificadas, verificar información de seguros médicos y administrar citas. En Nimblr Inc. estuve a cargo de realizar la integración de Holly con un EHR que a lo largo de este capítulo llamaré el ***proveedor***.

4.1 Integración con EHR

El *proveedor* opera dentro de EE. UU., se encuentra dentro de los 10 sistemas electrónicos de salud más utilizados en este país con más de 19 mil usuarios. Por su presencia dentro de los principales proveedores de servicios digitales del área de salud en EE. UU. los beneficios de realizar la integración son: adquirir conocimiento técnico y de mercado, expandirse comercialmente y dar a conocer los servicios de Nimblr Inc. con más personas del sector.

El proceso de integración se dividió en varias tareas:

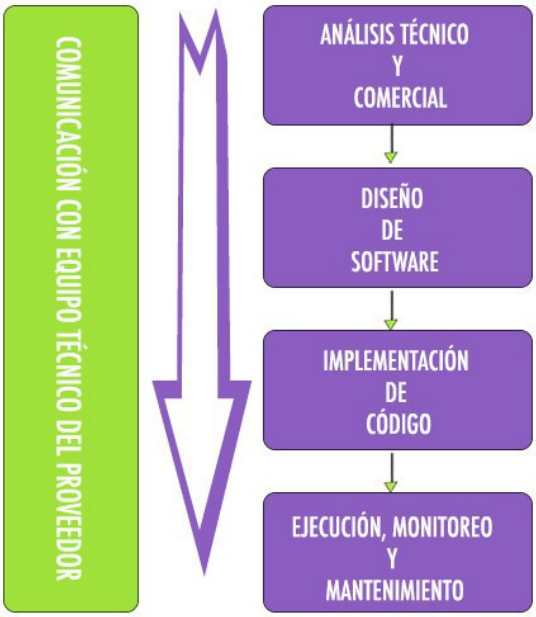

Ilustración 20 Flujo de trabajo para implementación con el proveedor. Fuente: Elaboración del autor

Además, a solicitud del *proveedor* se debió completar un proceso de certificación de *Health Insurance Portability and Accountability Act (Ley de Responsabilidad y Portabilidad del Seguro de Salud, HIPAA)* que es la regulación de la información digital en el sector salud en EE.UU., esto lo detallo de manera más extensa en el punto 4.1.7.

Para garantizar un proceso ordenado seguí las prácticas del marco de trabajo SCRUM: juntas diarias de reporte, entregas bimensuales funcionales, evaluación conjunta de subtareas y ajustes en la planeación basados en requerimientos. Esto ayudó al equipo a estar siempre informado acerca del avance de mis tareas y a poder tomar decisiones de planeación en conjunto. Además, con esto tenía objetivos claros con tiempos definidos viables y por lo tanto los riesgos se redujeron sustancialmente.

4.1.1 Comunicación con equipo técnico del proveedor

El proveedor tiene un proceso definido para todas las aplicaciones cliente que deseen ofrecer un producto a sus clientes. El proceso es estricto: contempla fechas de entrega, revisión de avances, pruebas en tiempo real y entrega de documentación detallada en periodos establecidos por el equipo técnico. Participé en llamadas para verificar el avance de la integración, mantuve comunicación con el equipo mientras se realizó la revisión de toda la integración, y posterior a ella sostuve varias sesiones hasta resolver las dudas del equipo del proveedor.

La comunicación fue un proceso paralelo a todos los demás descritos a continuación porque el proveedor requería tener llamadas semanales inicialmente y mensuales al final.

4.1.2 Análisis técnico y de negocio

La primera parte de la integración consistió en realizar un análisis de la interfaz de comunicación existente para determinar: viabilidad, dificultad y tiempo. Para consultar la documentación y guías fue necesario solicitar un usuario al equipo técnico del proveedor. La plataforma tiene varias interfaces de comunicación incluyendo HTTP y *Health Level Seven (*nivel de salud siete, HL7*)*. La interfaz HTTP es de tipo REST y cuenta con los métodos necesarios para la integración con Holly. Debido al soporte existente y facilidad de desarrollo elegí utilizar esta interfaz.

La integración es viable sólo cuando la interfaz cuenta por lo menos con las siguientes operaciones:

- Listar citas por rango de fechas
- Buscar citas por id
- Crear citas
- Actualizar citas
- Buscar paciente por id o nombre
- Listar doctores

Después de revisar toda la documentación pude confirmar que contaba con las funcionalidades mínimas necesarias y encontré también operaciones necesarias para realizar funcionalidades adicionales.

La seguridad de la API de *el proveedor* está dada por 2 protocolos:

- *SAML (Autenticación):* este protocolo se utiliza en el proceso de donde la clínica otorga el acceso a la plataforma externa (en este caso Nimblr Inc.). El objetivo principal del protocolo SAML es garantizar la autenticidad de la identidad de todos los participantes.

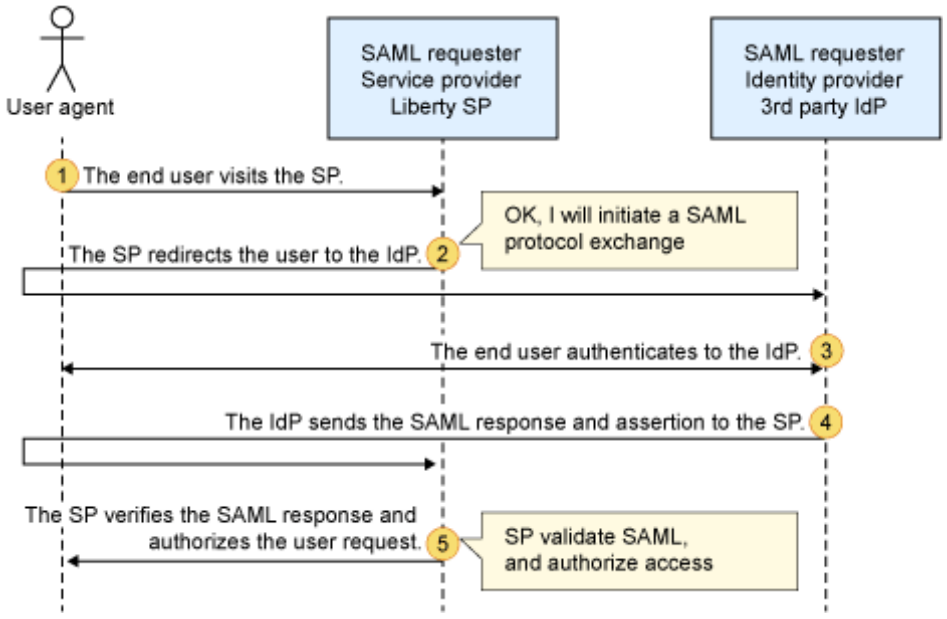

Ilustración 21 Diagrama de secuencia de autenticación SAML. Fuente: (IBM, 2018)

- *OAuth 2.0 (Autorización*): todas las peticiones a la API deben llevar en el encabezado un token de seguridad. Este token es generado por un servicio de autorización e identifica a la entidad solicitante de la información, además el token tiene un tiempo de vida después del cual es necesario solicitar uno nuevo para realizar más peticiones.

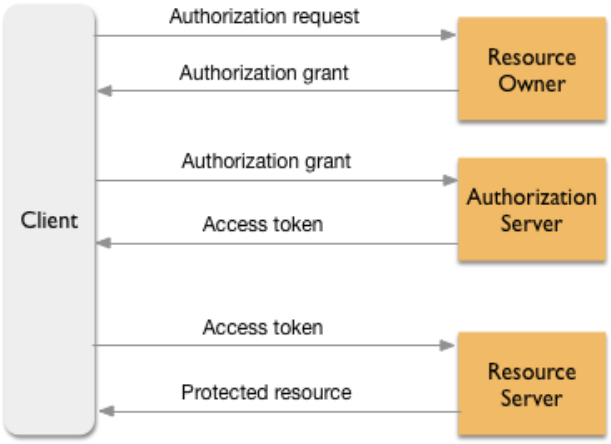

Ilustración 22 Diagrama de secuencia de autorización OAuth 2.0. Fuente: (apigee)

La plataforma de Nimblr Inc. cuenta con un módulo de autorización y renovación de credenciales OAuth 2.0, por ello no es necesario considerar desarrollo de código para esta parte. Sin embargo, no existía ningún módulo compatible con SAML, entonces fue necesario contemplar tiempo de desarrollo para esto en la planeación de la integración.
Por lo anterior determiné una integración viable al tomar en cuenta la experiencia con integraciones anteriores y la documentación disponible.

El siguiente paso fue realizar un análisis de como esta integración podía funcionar con el modelo de negocio de la empresa, para esto se llevaron a cabo algunas juntas con el CEO y el CTO en donde yo les transmití lo que había encontrado durante mi investigación para desarrollar una estrategia comercial. Se concluyó que la integración era viable del lado comercial.

Por último, organicé una junta de planeación con el equipo de ingeniería para determinar las subtareas, estimar los tiempos de ejecución y asignar a una persona responsable por cada una. Para la estimación el equipo utilizó la técnica de póker de planeación. A continuación, se muestra la tabla con las subtareas acomodadas por prioridad, tiempo y persona responsable:

	Prioridad	Horas	Responsable
Diseño			
Documentación UML	Regular	16	Desarrollador 1
Implementación			
Módulo de autenticación SAML	Alta	40	Desarrollador 2
Operaciones mínimas de manejo de eventos	Alta	120	Desarrollador 1
Operaciones opcionales de manejo de eventos	Baja	32	Desarrollador 1
Revisión de pares	Regular	8	Desarrollador 3
PRUEBAS			
Banco de pruebas unitarias	Alta	40	Desarrollador 1
Pruebas en stg	Alta	24	QA
Pruebas en prl	Regular	16	QA
CERTIFICACIÓN			
Proceso de certificación	Alta	120	Desarrollador 1
Total:		416	

Tabla 3 Estimación de tiempos de desarrollo. Fuente: Elaboración del autor

En total se estimaron 416 horas y 4 personas involucradas para completar la integración. Tomando en cuenta el sprints de 2 semanas, las subtareas se repartieron en los siguientes 6 sprints.

Después de obtener el permiso de la gerencia proseguí a la fase de diseño de la integración.

4.1.3 Diseño de software

Para diseñar la integración utilicé como referencia las historias de usuario definidas en la documentación, así como un conjunto de interfaces existentes en el sistema (patrón de diseño de fachada), las cuales definen las acciones ejecutadas por Holly durante su ejecución. También fue necesario considerar las pruebas unitarias y de integración, las cuales definen el comportamiento mínimo esperado de cualquier integración.

El siguiente es uno de los casos de uso definidos en la documentación:

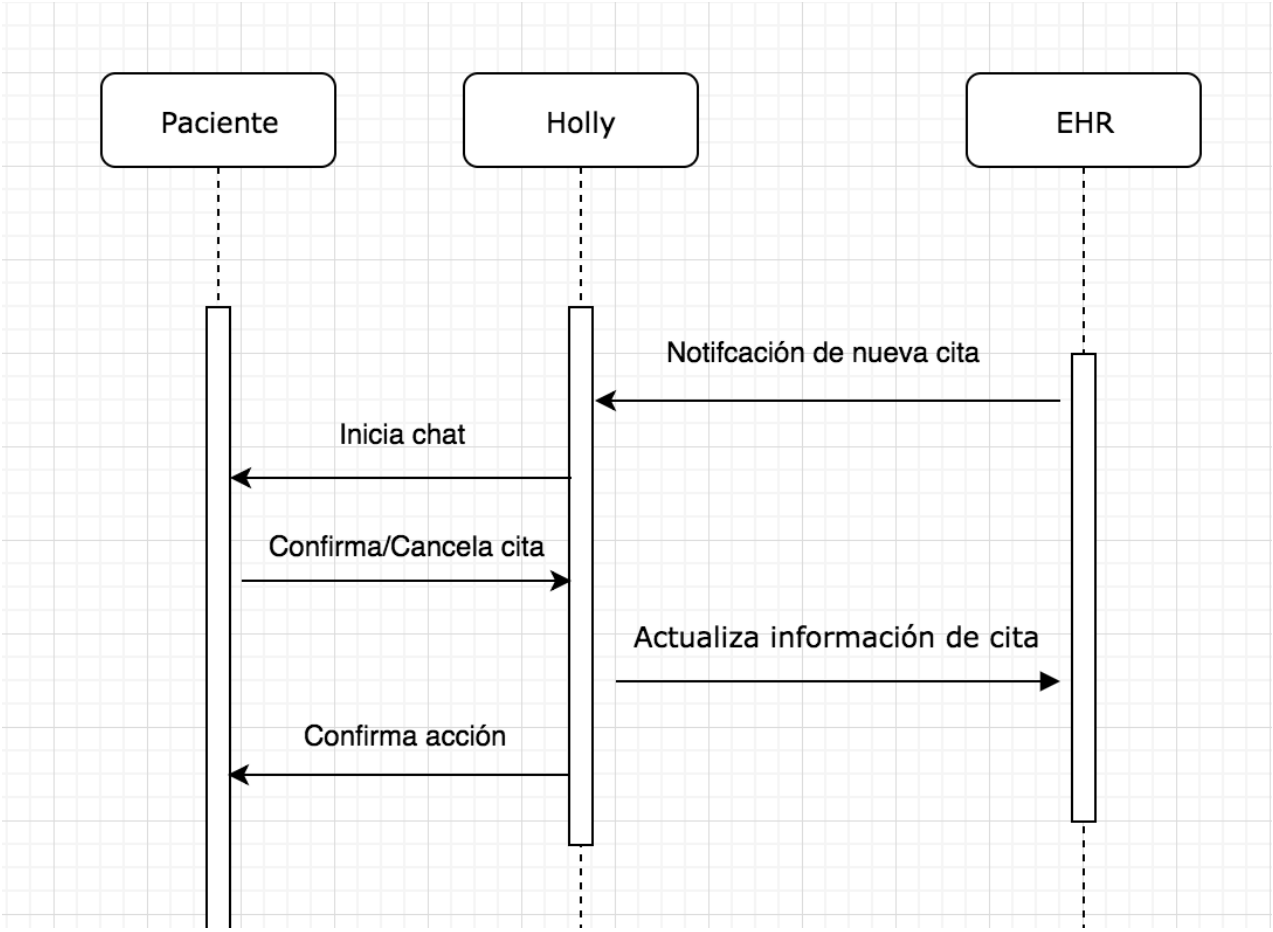

Ilustración 23 Diagrama de secuencia "Confirmación". Fuente: Elaboración del autor

Al tener este marco de referencia mi trabajo consistió en:

- Encapsular los métodos de la interfaz del *proveedor*: los métodos requeridos por Holly no siempre tienen una contraparte en las interfaces de los proveedores. Es necesario establecer las combinaciones de acciones necesarias para obtener los

resultados deseados por nuestro sistema. Por cuestiones de integridad en la información estas combinaciones deben de ser atómicas.

- Diseñar algoritmos: al establecer el flujo de las operaciones internas identifiqué rutinas ya utilizadas anteriormente, como: la autenticación de cliente, mapeo de modelos, homogenización de fechas, etc. También fue necesario diseñar una función recursiva y adicionalmente usar la técnica de memorización para reducir el número de llamadas asíncronas HTTP.

4.1.4 Implementación

Durante la implementación utilicé el proceso de desarrollo orientado a pruebas (*TDD* por sus siglas en inglés) para así garantizar el cumplimiento del código con el comportamiento mínimo esperado por el sistema.

Basándome en el proceso mencionado en la sección 3.7, la producción de software consistió en hacer una implementación del algoritmo planteado en la fase de diseño (primera aproximación), elegir un subconjunto del banco de pruebas y verificar las ejecuciones exitosamente; si fallaba alguna de las pruebas entonces regresaba a ajustar el código y repetía esto hasta terminar de ejecutar todas con éxito.

Este proceso alargó la duración de la fase de implementación en comparación al flujo clásico, en donde el programador sólo se enfoca en el camino de ejecución ideal, pero se garantiza un código con el funcionamiento mínimo esperado (calidad de software). Para asegurar la calidad de la implementación establecí un parámetro de cobertura de código mínima de 80%.

Debido a la madurez del sistema, y la experiencia previa con otros proveedores el banco de pruebas consistía en más de 60 pruebas unitarias, así como 20 pruebas de integración. Al terminar el proceso de implementación agregué 15 pruebas unitarias de casos especiales, las cuales no se habían contemplado antes.

En total esta fase duró 2 meses, durante los cuales hice múltiples presentaciones internas al equipo técnico y de ventas, además de 2 presentaciones al equipo del *proveedor*. El objetivo de las presentaciones internas fue obtener retroalimentación del equipo con base en el comportamiento y las funcionalidades esperadas, con esta retroalimentación se planeaba el siguiente conjunto de tareas para la siguiente entrega. De las presentaciones con el proveedor, la primera fue para comprobar la viabilidad de la

integración en el aspecto técnico y comercial, finalmente la segunda fue para realizar pruebas de estrés con monitoreo sobre la implementación final.

Siguiendo la filosofía Scrum “Software sobre documentación extensiva” el esfuerzo de esta fase se dividió 90% en generación de código y el otro 10% en documentación. Generé un documento exclusivo de la integración del *proveedor* en donde expliqué a grandes rasgos el encapsulamiento de las operaciones de la API y los posibles riesgos en el futuro, también contiene pseudocódigo de los algoritmos de mayor complejidad dentro de la implementación.

Uno de los problemas durante la implementación fue ajustar el ciclo de vida de las citas en el proveedor al ciclo de vida utilizado en nuestra plataforma: El proveedor tiene un proceso de creación de citas completamente distinto al de los demás proveedores ya que es necesario obtener información adicional en varios servicios web. Por ello fue necesario entonces cambiar la estructura del modelo de citas interno de la plataforma, y replicar el cambio en el resto de los microservicios, de tal manera la información necesaria se propagaría a lo largo de todo el flujo de comunicación.

El cambio en la estructura interna del modelo de citas también ocasionó cambios en la arquitectura del repositorio de datos con el propósito de tener un modelo más flexible y así facilitar más cambios en el futuro.

4.1.5 Ejecución, monitoreo y mantenimiento

Con la implementación finalizada, estable y aprobada por el *proveedor*, el siguiente paso en el proceso fue colocar el código en el ambiente de preproducción, y habilitar al usuario piloto, para lo cual tuve una llamada para guiar y monitorear el proceso completo de manera exitosa. A partir de ese momento trabajé con información real y en alto volumen.

Durante esta fase me dediqué a monitorear todas las interacciones con los pacientes del usuario y ajustar la configuración del usuario, de acuerdo con lo observado en el transcurso de los días; de esta forma el valor obtenido por Holly pudo ser el máximo.

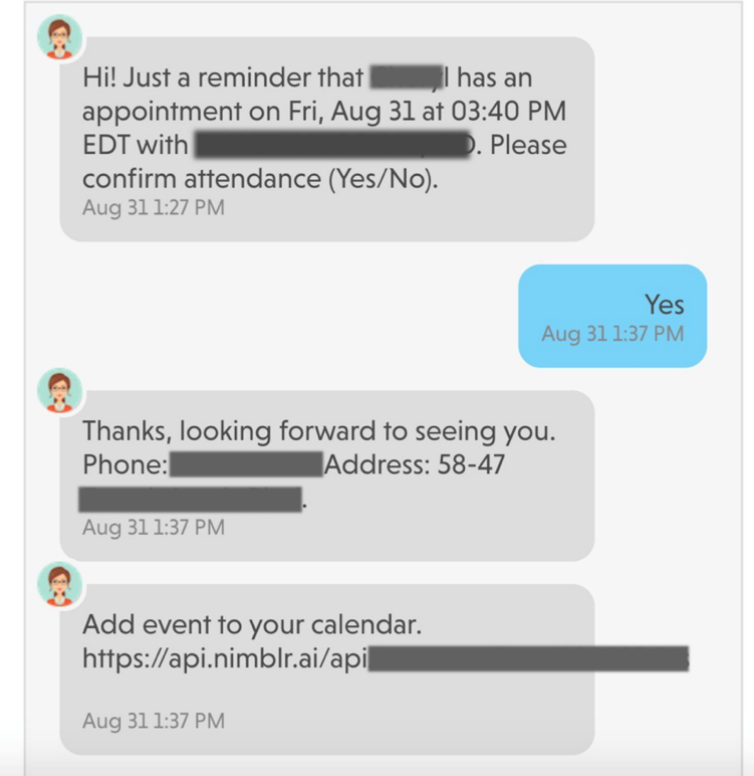

Ilustración 24 Conversación de Holly con un paciente dell proveedor. Fuente: Elaboración del autor

Como era de esperarse ocurrieron incidencias durante la ejecución de la integración, las cuales fueron corregidas en poco tiempo. Todas las incidencias se clasificaron como bajas, porque no fueron un riesgo para la operación general del sistema y no afectaban a la operación de ningún usuario.

Durante el monitoreo de la integración noté el número de peticiones por segundo, y estás crecieron rápidamente a medida que el sistema encontraba más citas.

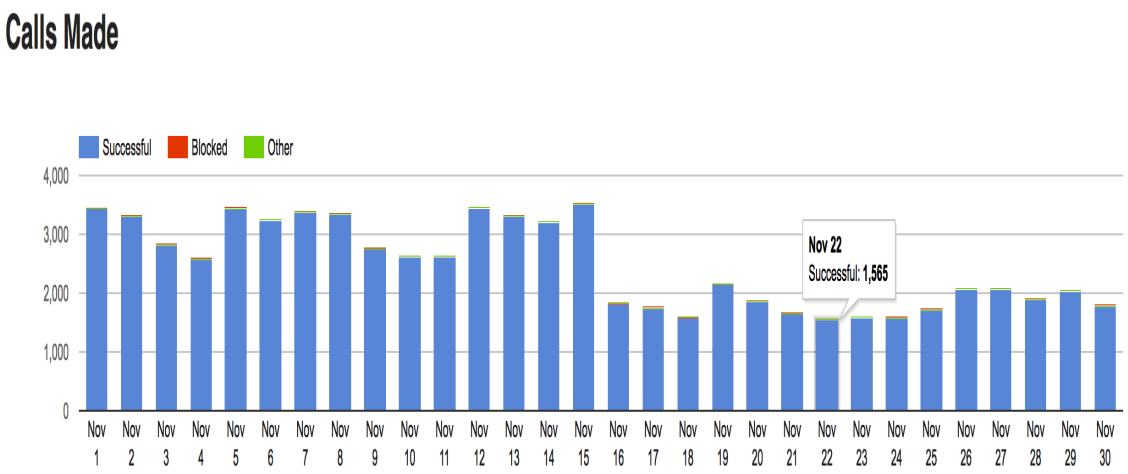

Ilustración 25 Número de llamadas HTTP a la API de el proveedor durante el mes de Noviembre. Fuente: Elaboración del autor

Esto representó un problema porque existe un límite en el número de peticiones por segundo para la interfaz del *proveedor*, entonces implementé una estructura de cola con prioridad. De esta manera limité la cantidad de peticiones paralelas y además se respetó el orden de ejecución de las peticiones. El pseudo código de la cola se muestra a continuación:

Insertar elemento

1. Recorrer el árbol hasta encontrar un nodo que tenga ninguno o un solo nodo hijo

```
FUNCTION encuentra_nodo_disponible(nodos):
    hijos = []
    por cada nodo en arreglo de nodos:
        IF NOT nodo -> izquierdo OR NOT nodo -> derecho
            return nodo
        INSERT nodo -> izquierdo IN hijos
        INSERT nodo -> derecho IN hijos
    RETURN encuentra_nodo_disponible(hijos)
```

2. Insertar el nuevo nodo (el lado izquierdo tiene preferencia)
3. Reacomodar los valores para preservar el orden del árbol

```
FUNCION insertar(valor):
    crear nodo nuevo con valor
    padre = encuentra_nodo_disponible(arreglo con nodo raiz)
    IF NOT padre -> izquierdo
        padre -> izquierdo = nuevo
    ELSE
        padre -> derecho = nuevo
    mientras árbol este desordenado:
        IF padre -> valor > nuevo -> valor
            intercambia padre con nuevo
```

Consumir elemento

1. El siguiente elemento es la raíz del árbol, almacenar en variable temporal y eliminar nodo raíz
2. Encontrar el último elemento del árbol y colocarlo como nueva raíz del árbol (el procedimiento es similar a encontrar el nodo disponible más próximo)
3. Reacomodar árbol (el procedimiento es similar a reacomodar al insertar)

Después de implementar y agregar la cola con prioridad a la integración logré mantener el número de peticiones limitado a 5 por segundo (el límite impuesto por el proveedor), y al mismo tiempo me aseguré de que las operaciones con mayor prioridad (actualizar información de cita) se ejecutaran siempre primero.

4.1.7 Certificación HIPAA

Al tener una integración operando con múltiples usuarios, el último paso del proceso fue completar un proceso de certificación como entidad HIPAA, a través de una compañía auditora elegida por el equipo del *proveedor*.

HIPAA es una ley vigente en EE. UU., la cual cubre todo lo relacionado a la protección de la información (digital y física) en el sector salud (Centers for Medicare & Medicaid Services, 2018). Además, define las penalizaciones y multas para las instituciones o personas que no cumplan con esta ley.

Cualquier sistema electrónico que reciba, procese y/o envié información catalogada como protegida bajo la ley HIPAA debe estar sujeto a una serie de políticas y procedimientos, ellos garantizan el manejo correcto de esta información. Además, se debe someter a procesos de auditoría de manera regular con terceros designados quienes verifican el correcto cumplimiento de los procesos de seguridad definidos.

Para cumplir con el proceso de auditoría documenté las políticas y procedimientos más adecuados para Nimblr Inc. tomando en cuenta los servicios de infraestructura con los que cuenta, junto con los requerimientos marcados dentro de la ley. Definí 2 tipos de procesos:

- Administrativos: Estos incluyeron capacitación de personal, protocolos de autorización de acceso a la información, protocolos de contratación de personal, definición de roles con responsabilidad dividida, protocolos de auditorías internas y protocolos de control de incidencias.

- Técnicos: Fueron las tareas operativas como programación de respaldos automáticas de información protegida, definición de usuarios con acceso restringido, registro de accesos, canales de comunicación cifrados, programación de actualizaciones de seguridad regulares a todos los sistemas y habilitación de cifrado en cualquier sistema de almacenamiento.

Mi responsabilidad fue planear y garantizar procesos definidos con el apoyo de la sección de operaciones y la gerencia de la empresa. Comencé con la documentación de políticas hasta la implementación y auditoría de ellas, el proceso duró 1 mes.

4.1.8 Evaluación de proceso

Al finalizar la integración tuve una reunión de retrospectiva con todos los participantes del proceso. Las conclusiones de la reunión fueron:

- Los tiempos estimados se cumplieron con una desviación máxima del 33%, se muestran en la *Tabla 4*
- El proceso de certificación no se estimó correctamente
- La experiencia anterior acumulada permitió la realización de algunas tareas en menor tiempo del estimado
- La baja cantidad de incidencias en la fase de pruebas fue un buen indicador de la madurez del proceso de desarrollo

	Horas estimadas	Horas Reales
Documentación UML	16	12
Módulo de autenticación SAML	40	48
Operaciones mínimas de manejo de eventos	120	136
Operaciones opcionales de manejo de eventos	32	24
Revisión de pares	8	8
Banco de pruebas	40	48

unitarias		
Pruebas en ambiente de pruebas	24	24
Pruebas con usuario piloto	16	16
Proceso de certificación	120	148
Total:	416	464

Tabla 4 Comparativa de tiempos estimados contra tiempos reales. Fuente: Elaboración del autor

La conclusión general obtenida del proceso de retrospectiva fue la identificación del nivel de madurez del equipo de desarrollo, el cual fue satisfactorio al poder planear y ejecutar proyectos a largo y mediano plazo.

El proyecto se encuentra ahora listo para ser explotado por el área comercial y generar un retorno de inversión para la empresa. De acuerdo con ellos las proyecciones muestran la integración de 400 a 500 nuevos clientes de la plataforma para los próximos 6 meses.

Capítulo 5
RESULTADOS Y CONCLUSIONES

Formar parte del área de sistemas de Nimblr Inc. ha sido una gran oportunidad de crecimiento en el ámbito profesional e interpersonal. Constantemente tengo acercamiento con tecnologías nuevas y todos los días me enfrento a retos técnicos y de otros ámbitos en donde puedo aplicar lo aprendido en la Facultad de Ingeniería.

En muchos de los retos encontrados ha sido al final mi formación escolar la principal herramienta para terminarlos, ya sea al poder aplicar conceptos básicos de manera conjunta, al ingeniar soluciones complejas (pero elegantes) con las herramientas disponibles, al poder encontrar y adquirir conocimiento nuevo o incluso al realizar una tarea en equipo aprovechando las fortalezas de cada integrante.

La realización de la certificación HIPAA significó otro reto, porque involucró muchas actividades relacionadas con diferentes áreas de conocimiento y de distintas dificultades: crear filtros para los mensajes salientes, crear documentación de los procedimientos del equipo de desarrollo, crear estrategias de almacenamiento de registros a largo plazo o establecer planes de capacitación continua en materia de seguridad para los empleados de la empresa.

El poder completar exitosamente el proyecto de integración fue un paso muy importante en mi carrera, por el conocimiento adquirido y por los beneficios que esto trajo para la empresa, así como para los usuarios de la plataforma. En estos momentos más de 1500 usuarios hacen uso de esta integración diariamente.

Al inicio parecía una tarea titánica en donde no tenía muy claro la dirección tomar, pero al final sólo tuve aplicar lo que ya conocía. En especial fue difícil por ser una tarea a largo plazo completamente nueva para mí. El mayor recurso con el que conté fue la experiencia de los programadores senior, la documentación de la plataforma y el apoyo de mis colegas.

Una de las cosas más valiosas aprendidas en la **Facultad** fue a siempre realizar las cosas de manera correcta y de la mejor manera posible. Después de más de un año en Nimblr Inc. he logrado entender la importancia de tener procesos de desarrollo definidos para lograr esto. Marcos de trabajo como Scrum contemplan un conjunto de procesos fundamentados en años de experiencia de cientos de personas, además de haber sido probados y mejorados a través del tiempo. Conocer y seguir estos

procesos me ha dado las herramientas para: evaluar la viabilidad de proyectos de software, crear planes de trabajo, crear software con calidad empresarial y evaluar resultados.

El objetivo final de todo esto es alcanzar la excelencia como ingeniero de software y como alumno egresado de la UNAM. Como alumno egresado de la máxima casa de estudios tengo el compromiso de siempre realizar el mejor trabajo posible siguiendo sus principios más característicos: honestidad, perseverancia, inteligencia, humildad y solidaridad.

Realizar el mejor trabajo incluye crear sistemas de calidad, trabajar siempre al marco de la ley, extender la mano a aquellos que lo necesiten, consultar a quienes cuentan con más experiencia y tomar decisiones en equipo. Más allá de la parte técnica, más allá de la teoría computacional nuestro trabajo como ingenieros es traer soluciones a los problemas que en un inicio parecen imposibles, soluciones eficientes con un buen uso de los recursos disponibles, soluciones escalables y aptas para trabajar por mucho tiempo bajo distintas condiciones.

Dentro de la página web de la **Facultad de Ingeniería** se encuentra como “Perfil de egreso del ingeniero en computación” (UNAM, 2019) lo siguiente:

> “Los egresados de la Facultad de Ingeniería deberán poseer: capacidades para la innovación, potencial para aportar a la creación de tecnologías y actitud emprendedora, con sensibilidad social y ética profesional; y con potencialidad y vocación para constituirse en factor de cambio.”

Desde mi punto de vista la Facultad de Ingeniería y la Universidad te dan todo lo necesario para tener la base de cómo construir el perfil mencionado arriba. Comenzando desde la División de Ciencias Básicas, en donde se empieza a adquirir esta habilidad para resolver cualquier tipo de problema. Después en las materias propias de la carrera se obtiene un contexto del papel de la automatización y los sistemas en el mundo moderno. Finalmente, con las diferentes opciones en los módulos de salida permiten profundizar en algún área de la computación con una orientación ya más profesional.

Me gustaría resaltar de la Facultad de Ingeniería: los excelentes profesores, el balance entre teoría y práctica (característico del buen ingeniero), la diversidad de conocimiento y el excelente acervo en las bibliotecas. Y como mención especial a las materias de humanidades incluidas en el plan de estudios: Literatura, Economía, Ética, Filosofía, Recursos de México. Estas materias, son parte de la misma

formación integral necesaria para un ingeniero. En el mundo uno debe interactuar con todo tipo de personas con distintas formaciones y es vital esa parte “humana”, la cual permite conectar con los demás. Es necesario ir más allá del código y los algoritmos. Estas materias te hacen ir más allá de eso.

Cabe mencionar por supuesto a los muchos programas de becarios existentes, los cuales dan la oportunidad a los alumnos de obtener conocimientos nuevos. Así como de practicar la aplicación de dichos conocimientos y crear lazos con compañeros con el mismo deseo de aprender. Como ejemplo de estos programas están: *Programa de Tecnología en Cómputo (PROTECO), Unidad de Servicios de Cómputo Académico (UNICA), Unidad de Servicios de Cómputo Administrativos (USECAD), Sociedad de Alumnos de Ingeniería en Computación (SAIC), UNAM Mobile, Laboratorio de Investigación para el Desarrollo Académico (Linda),* entre otros.

Como ex becario de PROTECO me queda claro también ahora (estando en el ámbito profesional) la verdadera importancia de ellos. Además de la preparación técnica avanzada, también obtuve una excelente habilidad de investigación y de transmisión de conocimientos, así como de liderazgo y trabajo en equipo, en su momento los consejos del creador y piedra angular de PROTECO el Ing. Heriberto Olguin. Es por eso que considero esta formación (y la de la propia facultad) como fundamental dentro de mi formación como ingeniero.

También me gustaría contar las áreas de oportunidad de la Facultad de Ingeniería, las cuáles he detectado en mis 4 años de experiencia laboral. La primera de ella es sobre las tecnologías empleadas por los profesores, pues algunas de ellas son obsoletas y no corresponden al estado actual del campo de trabajo. Esto por supuesto se puede explicar por el rápido avance que siempre ha caracterizado a esta área. Pero, aun tomando esto en cuenta me parece de suma importancia buscar una solución a esto.

Como estudiante de la generación 2011 curse el plan de estudios 2010 (UNAM, 2009), a mi parecer le hacían mucha falta más temas en el área de arquitecturas de sistemas y de sistemas web. Con respecto a esto, cabe mencionar el plan de estudios 2016 (UNAM, 2015), ya que cuenta con una mejora en la adición de la materia obligatoria de “Sistemas distribuidos”, también como punto a favor esta la nueva materia “Programación orientada a objetos”.

Es notable la existencia de algunas omisiones en contenidos de materias profesionales, las cuales seguramente desaparecerán con el tiempo, para seguir

reconociendo los grandes aciertos de las mejoras, porque las cosas buenas sobrepasan enormemente a las deficiencias mencionadas.

En conclusión, considero mi desempeño laboral bastante bueno, ya que soy capaz de llevar a cabo proyectos complejos en sus diferentes ámbitos. Gran parte de esto es gracias a la excelente formación que obtuve con los profesores de la Facultad de Ingeniería de la UNAM.

Bibliografía

Academy, A. (2015). *Scrum Master Certified Workbook Student.*

Amazon Web Services, Inc. (2016, Diciembre). *Microservices on AWS: AWS Whitepaper.* Retrieved Enero 10, 2019, from AWS: https://docs.aws.amazon.com/aws-technical-content/latest/microservices-on-aws/microservices-on-aws.pdf?icmpid=link_from_whitepapers_page

apigee. (n.d.). *Introduction to OAuth 2.0.* Retrieved from apigee: https://docs.apigee.com/api-platform/security/oauth/oauth-introduction

Baghdadi, Y. (2012). A methodology for web services-based SOA realisation. *International Journal of Business Information Systems, 10*(3), 264-297. Retrieved 1 10, 2019, from https://inderscienceonline.com/doi/10.1504/ijbis.2012.047531

Bradfield School of computer science. (n.d.). *Priority Queues with Binary Heaps.* Retrieved from Practical Algorithms and Data Structures: https://bradfieldcs.com/algos/trees/priority-queues-with-binary-heaps/

Campbel, R. (2012, Enero 17). *Emma Plugin for Jenkins.* Retrieved from cloudbees: https://www.cloudbees.com/blog/emma-plugin-jenkins-easy-code-coverage-reports

Centers for Medicare & Medicaid Services. (2018, Septiembre). *HIPAA BASICS FOR PROVIDERS.* Retrieved from Centers for Medicare & Medicaid Services: https://www.cms.gov/Outreach-and-Education/Medicare-Learning-Network-MLN/MLNProducts/Downloads/HIPAAPrivacyandSecurity.pdf

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (n.d.). *Introduction to Algorithms .* MIT Press and McGraw-Hill. Retrieved 1 11, 2019, from http://mitpress.mit.edu/books/introduction-algorithms

Crunchbase. (15 de Junio de 2017). *Seed Round - Nimblr.ai*. Obtenido de Crunchbase: https://www.crunchbase.com/funding_round/nimblr-ai-seed--183276dc

Dragoni, N., Giallorenzo, S., Lluch-Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. *arXiv: Software Engineering*, 195-216. Retrieved 1 10, 2019, from https://arxiv.org/abs/1606.04036

Endeavor. (11 de enero de 2018). *StartX de Stanford selecciona a Nimblr.ai para su programa de aceleración*. Obtenido de Endeavor: https://www.endeavor.org.mx/sala-de-prensa/startx-de-stanford-selecciona-a-nimblr-ai-para-su-programa-de-aceleracion

Erich, Gamma, Richard, Helm, Ralph, Johnson, . . . Vlissides. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley. Retrieved 1 10, 2019

Gamma, E. e. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Westford, Massachusetts: Pearson Education.

Google Inc. (2018, Julio 25). *Google Calendar API reference*. Retrieved Enero 10, 2019, from https://developers.google.com/calendar/v3/reference/

Grenning, J. (2002, Abril). *Wingman Software.* Retrieved from https://wingman-sw.com/papers/PlanningPoker-v1.1.pdf

Hughes, J. (1989). Why Functional Programming Matters. *Computer Journal, 32*(2). Retrieved 1 11, 2019, from http://www.cse.chalmers.se/~rjmh/Papers/whyfp.html

IBM. (2018, Diciembre 14). *SAML 2.0 Web Browser Single-Sign-On*. Retrieved from IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/cwlp_saml_web_sso.html

Kheirkhah, P. e. (14 de Enero de 2016). *Prevalence, predictors and economic consequences of no-shows.* Recuperado el 10 de Enero de 2019, de BMC Health Services Research: https://doi.org/10.1186/s12913-015-1243-z

Kocsis, Z. A., & Swan, J. (2017). Dependency Injection for Programming by Optimization. *arXiv: Artificial Intelligence*. Retrieved 1 10, 2019, from http://dblp.uni-trier.de/db/journals/corr/corr1707.html

Lucchi, R., Millot, M., & Elfers, C. (2008). Resource Oriented Architecture and REST. Retrieved 1 10, 2019, from http://inspire.jrc.ec.europa.eu/reports/ImplementingRules/network/Resource_orientated_architecture_and_REST.pdf

Microsoft. (2018, Noviembre 12). *Microservices architecture style*. Retrieved Enero 10, 2019, from https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Nimblr Inc. (2016). *Home.* Retrieved from Nimblr Inc: https://nimblr.ai/home

Notimex. (8 de Agosto de 2017). *Holly, el primer asistente virtual para el sector médico en México*. Obtenido de Excelsior: https://www.excelsior.com.mx/nacional/2017/06/08/1168664

Pressman, R. S. (2010). Principles that guide practice. En R. S. Pressman, *Software Engineering: A practitioner's approach* (págs. 113-115). Nueva York: McGraw-Hill.

Sedgewick, R., & Wayne, K. (2016). *Computer Science: An Interdisciplinary Approach*. Retrieved 1 11, 2019, from https://amazon.com/computer-science-interdisciplinary-robert-sedgewick/dp/0134076427

W3C. (2004, Febrero 11). *Web Services Architecture*. Retrieved from W3C Working Group: https://www.w3.org/TR/ws-arch/