UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de Software para el Monitoreo de Medidor Eléctrico a Distancia

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

José Donaldo Fernández Montenegro

ASESOR DE INFORME

Ing. Orlando Zaldívar Zamorategui

Ciudad Universitaria, Cd. Mx., 2018

Capítulo 1

Introducción

El trabajo lo realicé en una empresa privada, encargada de elaborar software para venta a empresas gubernamentales. El proyecto trata del muestreo, almacenamiento y presentación de datos obtenidos por medidores de voltaje y corriente.

Antes que nada comenzaré por explicar qué es el proyecto y qué necesidades se buscaban satisfacer al elaborarlo.

El proyecto consistió en el desarrollo de un sistema de software que permite consultar información, tanto histórica como en tiempo real de múltiples medidores eléctricos en distintas locaciones, de manera remota y por medio de una misma interfaz.

El desarrollo de este proyecto tiene como finalidad el disminuir el tiempo y dinero que se invertía en ciertos procesos de negocio que tiene la empresa, como son:

- La supervisión de líneas eléctricas.
- Consulta de datos en tiempo real.
- Almacenamiento y consulta de datos históricos.

Este proyecto debido a los requerimientos del cliente se inicio desde cero y actualmente ya existe una primera versión en un entorno real.

Para el desarrollo de este proyecto se propuso el uso de un arquitectura orientada a servicios que permitiera distribuir por distintos dispositivos y sistemas operativos la información obtenida, tanto del muestreo de los medidores en tiempo real como la almacenada en base de datos.

1.1. Objetivo

El proyecto que se presenta a continuación tiene la finalidad de facilitar y agilizar los tiempos de consulta de datos en distintas instalaciones electricas; la implementación de este sistema representa para el cliente la disminución de tiempo en procesos referentes a corrección de fallas y mantenimiento de instalaciones eléctricas.

Este proyecto tiene como finalidad empezar a solventar la falta de implementaciones tecnológicas en el ámbito del software para esta empresa, es por esto que todas las tecnologías aquí planteadas fueron diseñadas con el propósito de que el software sea una base escalable para una futura expansión de la plataforma de software.

1.2. Descripción de la empresa

La empresa donde laboré nueve meses está enfocada al campo de las licitaciones de software; es una empresa mediana que tiene más de veinticinco empleados y más de veinte años en el ámbito del desarrollo de software y ventas al sector público en el ámbito de tecnología.

La misión de la empresa es:

“Brindar un servicio de la más alta calidad en consultoría, desarrollo, venta, mantenimiento y reparación de distintos tipos de tecnológicas tanto de software como hardware, siempre brindando la mejor solución y apoyando a nuestros clientes con cualquier reto que enfrenten”.

Actualmente el proyecto principal y más grande en el que trabaja la empresa es el descrito en este informe.

En la primera versión del proyecto se plantea el uso de una interfaz web para la presentación de los datos, sin embargo la arquitectura del proyecto está pensada a modo que en un futuro la interfaz podría estar también en dispositivos móviles.

1.3. Descripción del puesto de trabajo

El puesto por el que estaba contratado era “full stack developer”, las responsabilidades correspondientes a este puesto comprendía tareas de programación, tanto de la parte de presentación de la aplicación (interfaz de usuario) como de la parte del servidor e inclusive tareas relacionadas con la administración de la base de datos.

Capítulo 2
Requerimientos

Como parte del desarrollo de este proyecto se plantearon 9 requerimientos funcionales y 9 no funcionales. Un requerimiento es también conocido como una característica o restricción que debe tener un sistema para satisfacer a un cliente. Éstos los podemos dividir en dos: los requerimientos funcionales y los requerimientos no funcionales. A continuación veremos de qué trata cada uno de ellos y cuáles eran las necesidades del proyecto.

Los requerimientos funcionales son todas aquellas necesidades que se buscan satisfacer con la interacción de un sistema y sus usuarios (este tipo de requerimiento también es conocido como servicio), dicho de otra forma es un requisito descrito por el negocio que tiene una salida o solución esperada, puede iniciar a través de un conjunto de entradas o de una acción dada y busca dar precisión a lo que se espera que realice el software. Tener en claro los requerimientos funcionales del software da pie a que no hayan interpretaciones ambiguas que deriven en cambios en el desarrollo del sistema y que a su vez llevarán a un mayor costo en el desarrollo del producto.

Los requerimientos no funcionales describen aspectos que no están relacionados de manera directa con el buen funcionamiento del sistema. Ejemplos de este tipo de requerimiento son el tiempo de respuesta, la seguridad, tiempo de entrega, cantidad de usuarios, entre otros.

2.1. Levantamiento de requerimientos

Para el levantamiento de requerimientos fue necesario realizar entrevistas con el cliente a modo de poder entender cuáles serían las funcionalidades que tendría el sistema. A través de toda una semana de entrevistas con el cliente pudimos llegar a los siguientes requerimientos para la implementación del sistema, los cuales como ya mencionamos anteriormente dividiremos en funcionales y no funcionales respectivamente.

2.1.1. Requerimientos funcionales

Los requerimientos funcionales se muestran en la tabla 1.

Tabla 1. Requerimientos funcionales

No.	Requerimiento funcional
1	Crear un sistema de consulta para todos los datos de distintos medidores.
2	Se contará con un registro de usuarios.
3	El sistema tendrá notificaciones para cualquier eventualidad.
4	El software permitirá administrar los usuarios y notificaciones.
5	El sistema podrá imprimir los datos en tiempo real.
6	El sistema mostrará el histórico de los datos de manera opcional.
7	Se podrán consultar los datos en forma gráfica y tabular.
8	Todos los datos guardados deberán contar con metadatos que aporten detalles como la ubicación y tiempo de muestreo.
9	Los datos a mostrar deberán ser configurables.

2.1.2. Requerimientos no funcionales

Los requerimientos no funcionales se muestran en la tabla 2.

Tabla 2. Requerimientos no funcionales

No.	Requerimiento no funcional
1	El sistema deberá construirse pensando en la escalabilidad del proyecto.
2	El proyecto se implementará en primera instancia en versión web, pero en un futuro podría ser llevado a versión móvil.
3	Alta disponibilidad del sistema.
4	El sistema deberá ser compatible con el navegador Google Chrome a partir de la versión 5.0.
5	El software deberá ser capaz de atender hasta 100 usuarios al mismo tiempo.
6	El desarrollo sólo podrá ser usado desde la intranet del cliente.
7	Se deberá tener un tutorial para el sistema.
8	Se propone la utilización de un arquitectura orientada a servicios.
9	Se propone la creación de un API.

Un API significa por sus siglas en inglés *application programming interface* y es un conjunto de funciones que pueden ser utilizadas por otro software; un API permite la utilización de estas funciones sin tenerlas que programar de nuevo e inclusive sin conocer su estructura interna, debido a que sólo se realiza una llamada a la función (con o sin parámetros) y ésta nos regresa un dato o acción esperada.

2.2. Identificación de actores

Una vez que se obtuvieron los requerimientos para el proyecto y continuando con la planeación, se realizó la identificación de actores, donde un actor representa a uno de los roles que puede ser usado por un ente que interactúe con nuestro sistema.

En la tabla 3 se identificaron los actores dentro del sistema.

Tabla 3. Actores dentro del sistema

No. Actor	Tipo de Actor	Descripción
1	Usuario general	Este usuario podrá consultar los dispositivos y los datos disponibles de cada uno de ellos, también recibirá notificaciones.
2	Usuario administrador	Usuario administrador tendrá todos los privilegios del usuario general y también la capacidad de asignar nuevos dispositivos a usuarios generales, así como cambiar permisos de visualización de información en todos los dispositivos que tenga asignados.
3	Administrador	El actor administrador tiene todos los privilegios del usuario general y el usuario administrador, también tendrá la capacidad de dar de alta nuevos dispositivos y asignarlos al usuario administrador, podrá modificar todos los permisos de todos los usuarios.

2.3. Identificación de casos de uso

Para obtener los casos de uso se analizó cada uno de los requerimientos y se complementó con las entrevistas hasta determinar cuáles son los comportamientos deseados por el sistema.

Lo primero que hay que entender es ¿qué son los casos de uso?, podemos definirlos como la descripción de más alto nivel de una funcionalidad en un sistema, suele representarse comúnmente como las acciones explícitas que llevará a cabo el software en desarrollo. Es recomendado que el nombre de los casos de uso esté compuesto por un verbo en infinitivo que deje de manifiesto la acción que se quiere realizar, por ejemplo “consultar datos”.

Los casos de uso nos proporcionaron un medio para que tanto desarrolladores como usuarios, se pudieran poner en sintonía en cuanto al funcionamiento que tendría el sistema. En la tabla 4 podemos ver los casos de uso.

Tabla 4. Casos de uso

No.	Casos de uso	Descripción
1	Consultar datos	Se podrá hacer la consulta de la información.
2	Administrar información	El usuario limitará la cantidad de información disponible a otros usuarios en cada dispositivo.
3	Administrar notificaciones	Se podrán consultar las notificaciones de los dispositivos.
4	Asignar dispositivos	El usuario asignará nuevos dispositivos a otros usuarios.
5	Insertar dispositivos	Se podrán insertar nuevos dispositivos en el sistema para ser muestreados.
6	Administrar usuarios	Se podrán modificar todos los privilegios que tenga un usuario.

2.4. Maquetado del sistema

En búsqueda de dar una mejor perspectiva de los requerimientos del sistema, el cliente desarrolló una maquetación en la que resaltó los puntos visuales clave de los elementos que más valor agregan al software.

En la figura 1 se muestra cuáles eran los componentes deseados para el inicio de sesión y dentro de las figuras 2 y 3 podemos ver cuál es la estructura en la que se esperaba representar los datos de los medidores.

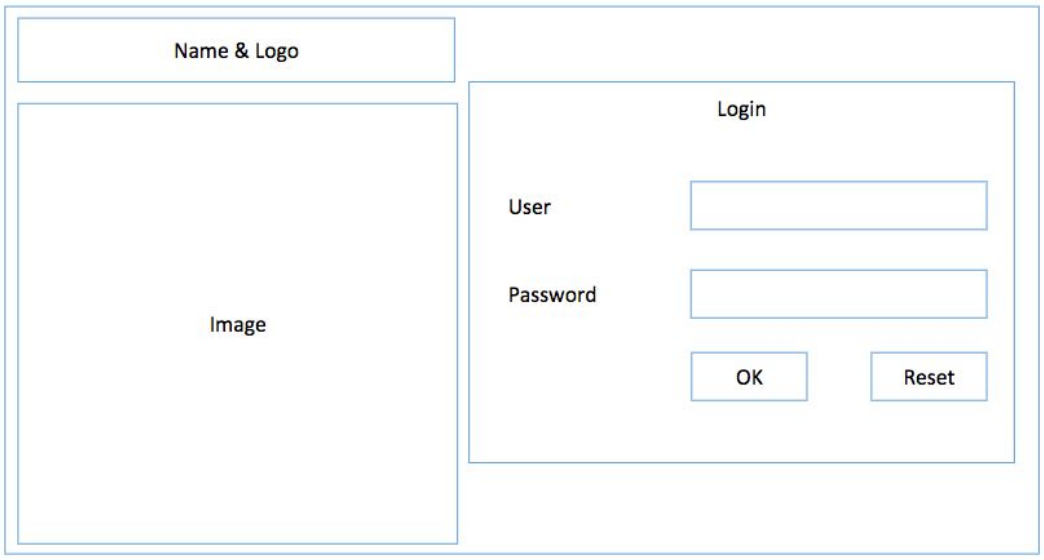


Figura 1. Maquetado del sistema

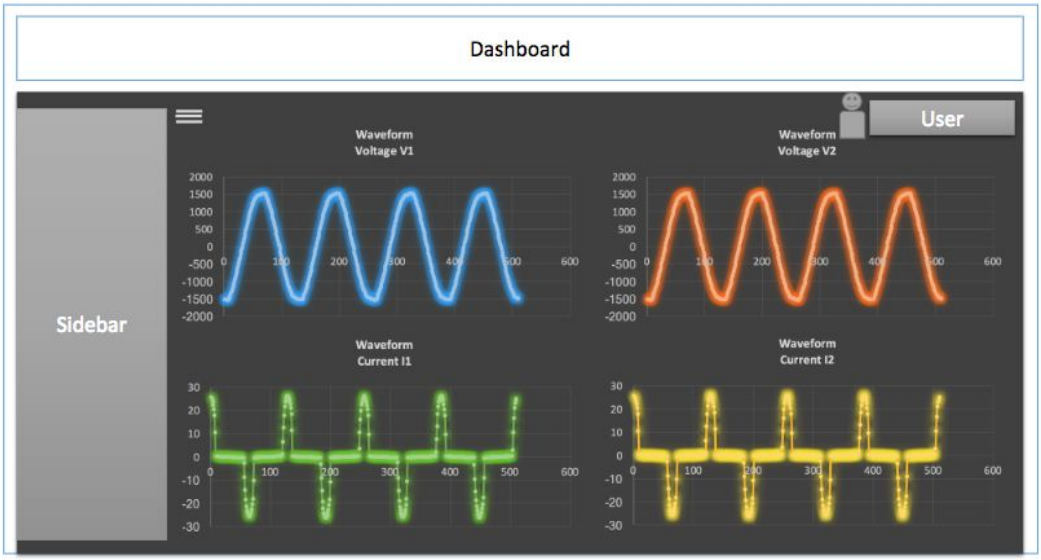


Figura 2. Maquetado del sistema

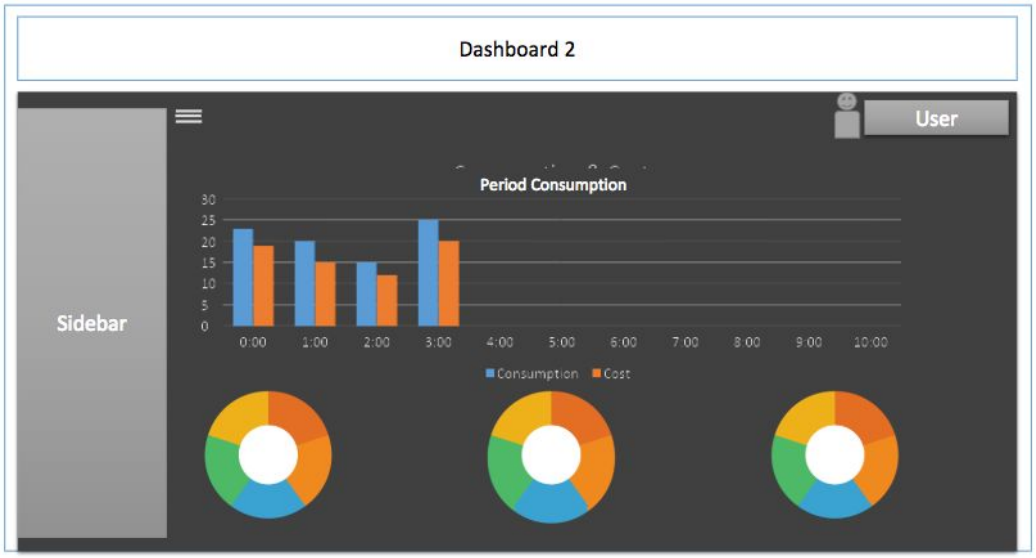


Figura 3. Maquetado del sistema

Capítulo 3

Metodología para el desarrollo

Para el proceso de desarrollo de software fue necesario adoptar una metodología que nos permitiera tener un esquema de trabajo organizado y constante, que a su vez fuera adaptable a cambios durante el proceso de elaboración; pero sobre todo que estuviera enfocado en los aspectos que dan valor al software por encima de cualquier tipo de documentación; para esto se propuso la implementación de una metodología ágil en el desarrollo del proyecto.

3.1. Desarrollos ágiles

Antes de continuar, definiremos las metodologías de desarrollo ágiles. Son aquellos donde se busca que a partir de un desarrollo iterativo e incremental se acelere el proceso de elaboración de un producto de software (estas metodologías también son aplicables a otros ámbitos), buscando no enfocarse en la documentación, si no en el software y el valor que aporta al cliente. Para esto se tienen los siguientes principios de las metodologías ágiles que se deben seguir durante la implementación del desarrollo, como se muestra en la tabla 5.

Tabla 5. Manifiesto ágil

Principio	**Descripción**
Participación del cliente	El cliente del proyecto tiene que estar involucrado de manera constante en el proceso de desarrollo, analizando los requerimientos del sistema y evaluando los resultados de cada iteración.
Entrega incremental	El software se desarrolla en ciclos, donde en cada ciclo se tendrá algo que entregar.
Personas, no procesos	El desarrollo debe estar enfocado en el equipo y en sus formas de trabajo y no la imposición de procesos formales.
Aceptar el cambio	El diseño, planeación e implementación del software deben de tener en cuenta que pueden haber cambios en cualquier parte del proceso.
Mantener la simplicidad	Se busca siempre mantener el proceso de desarrollo lo más simple posible.

Una de las razones por las cuales se pudo tomar la decisión de implementar una metodología de desarrollo ágil fue la experiencia previa que se tenía por parte de la empresa en el uso de éstas.

3.2. Marco de trabajo scrum

Scrum es un marco de trabajo enfocado en el desarrollo ágil, muy orientado al desarrollo de software (scrum no se limita únicamente a este campo de aplicación, pero generalmente es el ámbito donde más se encuentra).

Scrum consiste en el desarrollo de un producto (en este caso de software) junto con un equipo durante varios ciclos iterativos; un ciclo iterativo se refiere a una cantidad de tiempo constante donde un equipo tendrá que entregar finalizadas determinadas tareas previamente seleccionadas que representarán un avance parcial del proyecto y que demostrará el valor obtenido por el producto durante este ciclo, se avanzará de manera consecutiva por medio de ciclos hasta finalizar el proyecto.

3.2.1. Equipo de scrum

Para el desarrollo de scrum es necesario tener al menos tres distintos roles dentro de cualquier equipo de scrum, cada uno de éstos con una serie de actividades específicas.

- Producto owner: Este rol dentro del scrum representa a los interesados en el proyecto, ayudando con los requerimientos, dando un punto de vista y apoyando con la retroalimentación.

- Scrum master: Quien toma este rol está enfocado en administrar todo el proceso de scrum (administración de ceremonias, análisis de avance, etc).

- Equipo de desarrollo: Como su nombre lo dice este rol se enfoca al desarrollo del producto en todas sus facetas (extracción de requerimientos, implementación y pruebas).

3.2.2. El sprint

En scrum a cada ciclo de trabajo se le conoce como sprint. Un sprint es el tiempo dado a un equipo de trabajo para realizar cierto cantidad de tareas, al inicio de cada ciclo se plantean las tareas que serán resueltas en dicho intervalo de tiempo, las tareas son obtenidas de los casos de uso o de historias de usuario (priorizando la realización de las que tengan mayor valor para el negocio). A cada tarea se le asigna un valor de complejidad con respecto a las demás tareas donde a mayor nivel de complejidad implicaría mayor tiempo de desarrollo, un sprint puede tener una duración variable entre dos y seis semanas, pero por lo general es recomendable que duren cuatro semanas. Este valor se deberá seleccionar dependiendo del ritmo de trabajo del equipo y la meta que se busque.

3.2.3. Ceremonias de scrum

Dentro de scrum existen cuatro tipos de juntas (ceremonias):

- Sprint planning: Esta junta se debe llevar a cabo al inicio de cada sprint donde se planeará cuáles serán las tareas a llevar a cabo durante este periodo de tiempo, durante la junta se contestarán toda clase de dudas referentes al ciclo que está por comenzar.

- Daily meeting: Esta junta se realiza todos los días del sprint y procura no tener una duración mayor a 15 minutos (se recomienda que se tome de pie con el fin de minimizar la duración), durante esta junta se plantean 3 preguntas: ¿qué se ha hecho?, ¿qué se hará? y ¿se tiene algún impedimento para realizar el trabajo que estamos haciendo?, la finalidad de esta junta es

agilizar el trabajo en equipo, aumentar la productividad y buscar el compromiso por parte de todos los integrantes.

- Sprint review**:** En esta junta se habla de los logros del equipo en el sprint, así como una demostración del trabajo realizado, a veces no es posible hacer demostraciones, pero se deberán mostrar los logros obtenidos durante el sprint.

- Sprint retrospective: Esta junta está diseñada para que todos los miembros del equipo puedan reflexionar sobre la forma en que se hizo el sprint pasado y poder mejorar en los siguientes ciclos a modo de evitar lo incorrecto y reforzar lo correcto.

3.3. Scrum dentro del proyecto

Ahora que ya explicamos cuáles son los fundamentos del scrum, hablaremos de cómo fue la implementación dentro del proyecto, para esto dividiremos el proceso en diez pasos:

- El **primer paso** dentro de la planeación fue escoger a un product owner, esta persona era un empleado proveniente de la entidad gubernamental que solicitó el producto, dicha persona tenía claro qué es lo que se quería hacer, lo que es posible de hacer y también poseía una buena noción de los riesgos y recompensas que se obtendrían con el desarrollo del proyecto.

- El **segundo paso** fue integrar la primera parte del equipo, un scrum master quien ayudaría con todo el planteamiento y proceso del scrum y a un analista de negocio (business analyst) quien por su parte sería el encargado de crear una abstracción del proyecto y validarla con el cliente.

- El **tercer paso** tiene como finalidad obtener una bitácora del producto mediante casos de usos e historias de usuario, en otras palabras es una lista con los requerimientos existentes para el proyecto, donde todos los elementos de esta lista están priorizados, según agreguen valor al producto final.

- El **cuarto paso** buscó integrar al resto del equipo, una vez que los requerimientos del producto estaban claros y verificados por el product owner, se reclutó a un equipo de desarrolladores y un tester (verificador) que seríamos los encargados del desarrollo y la planeación técnica del proyecto.

- El **quinto paso** consistió en hacer la planeación técnica del proyecto, a pesar de que se tenía una idea general, fue un proceso donde se examinó la bitácora del producto, los requerimientos y los alcances que buscaba el producto, a partir de esto se realizó una sugerencia técnica especificando las tecnologías que debían ser usadas para el óptimo desarrollo del proyecto.

- El **sexto paso** es estimar la bitácora del producto, esta actividad buscó evaluar cada uno de los puntos de la bitácora dependiendo del esfuerzo que tomará realizar cada tarea; para asignar el esfuerzo que requeriría cada una de las tareas se usó como referencia la serie de Fibonacci 1,2,3,5,8,13,21 donde el valor 1 representa una tarea que requiere poco esfuerzo y el valor 21 representa mucho esfuerzo. Se suele utilizar esta serie porque la diferencia que hay entre números es lo suficientemente amplia como para poder diferenciar la cantidad de esfuerzo que se asigna a cada tarea.

- El **séptimo paso** consistió en realizar la planeación del primer sprint, para este momento se tomaron de la bitácora del producto las actividades de mayor valor para el negocio y se incluyeron en el primer sprint, se sumaron los puntos de esfuerzo (del sexto paso) y se estipuló que el tiempo de cada sprint sería de dos semanas, se incluyeron tantos puntos de esfuerzo como el equipo creía posible comprometerse a realizar durante el primer sprint y con eso inició la ejecución.

- El **octavo paso** se asignaron los estados dentro de los que puede estar cada tarea durante el desarrollo, la cantidad de estados es dependiente de cada proyecto, pero dentro de éste se definieron siete posibles estados que puede tener cada tarea; cada uno de ellos representa qué avance tiene cada tarea dentro del ciclo de desarrollo. Podemos ver cada uno de los estados en la tabla 6.

Tabla 6. Estados de tarea en scrum

Tarea en bitácora del producto	Tareas con esfuerzo estimado	Tareas listas para el desarrollo	Tareas en desarrollo	Tareas listas para testing	Tareas en testing	Tareas terminadas

- En el **noveno paso** se inicia el desarrollo del software y se da seguimiento a todo el sprint por medio de las ceremonias, hasta realizar el siguiente sprint.

- El **décimo paso** consiste en recabar la información obtenida del sprint e iniciar la planificación del siguiente sprint.

3.3.1. Bitácora del producto para scrum

La bitácora del producto es una parte fundamental para el desarrollo del scrum; una bitácora del producto es una lista de tareas que se deben realizar para satisfacer los requerimientos del producto, cada una de estas tareas contará con criterios de aceptación que se validaron previamente con el cliente y serán la forma de dividir el trabajo entre el equipo de desarrollo.

Para realizar una bitácora del producto, lo primero que se debe de identificar son las épicas. Estas son tareas muy generales de lo que se debe de hacer para la elaboración del sistema. Ejemplos de épicas son los siguientes: “Crear inicio de

sesión", "Ver gráficas en el sistema", etc. A partir de las épicas se crearán tareas más pequeñas que podrán ser divididas entre el equipo de desarrollo, por ejemplo "Crear pantalla de inicio de sesión", "Crear Funciones del servidor para inicio de sesión".

En la tabla 7 se muestran las épicas del proyecto.

Tabla 7. Épicas del sistema

No.	Épicas
1	Diseñar interfaz del sistema.
2	Crear registro de usuarios.
3	Crear inicio de sesión.
4	Modelar interfaz para la visualización de datos.
5	Implementar la navegación en el sistema.
6	Elaborar el sistema de administración.
7	Implementar un sistema de notificaciones.
8	Obtener información del medidor.
9	Procesar información del medidor.
10	Crear base de datos.
11	Insertar en base de datos.
12	Diseño e implementación de API.
13	Conexión remota entre dispositivos de consulta y medidor.
14	Creación de tutorial.

Cada una de estas épicas se dividió en tareas compactas, bien definidas y que podrían ser asignadas por el scrum master entre el equipo de desarrollo durante los sprints. Cada tarea tenía criterios de aceptación validados por el cliente que se deben de cumplir al momento de realizar la tarea.

3.3.2. Herramientas para scrum

Para llevar el control del desarrollo de scrum se buscó la implementación de un software que pudiera ayudar en todos los pasos de administración de scrum (al administrar la bitácora del producto, ver el estado de las tareas, llevar el seguimiento del trabajo, etc).

En primera instancia se utilizó una herramienta llamada Trello que permitió llevar a cabo el seguimiento del estado de las tareas; sin embargo el software no cumplia con los requermientos necesarios para un funcionamiento óptimo.

En las figuras 4 y 5 podemos ver cómo es la herramienta y los distintos estados que podía llegar a tener cualquiera de las historias dentro de ella.

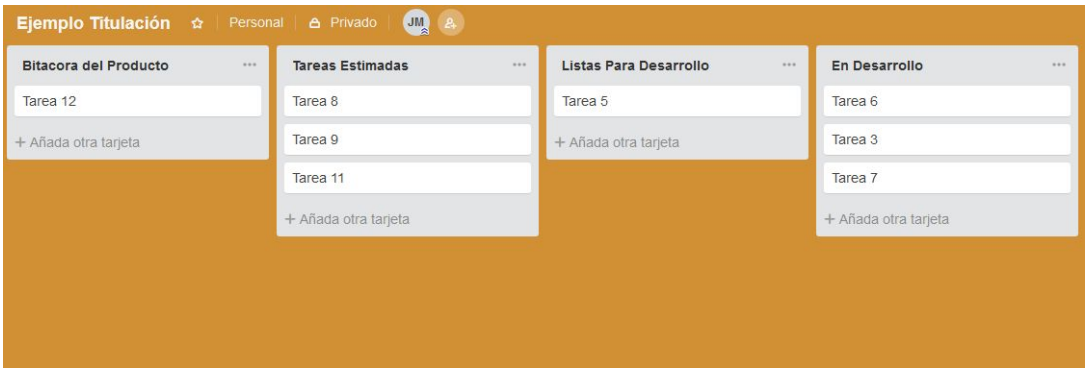


Figura 4. Ejemplo de Trello

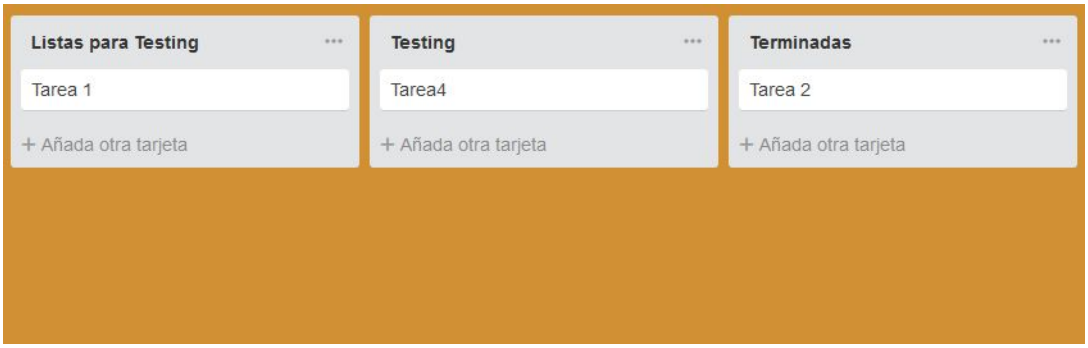


Figura 5. Ejemplo de Trello

La principal ventaja del uso de Trello es:

- No suma ningún costo al proyecto gracias a que la herramienta es gratuita.

Sin embargo, las desventajas que acarrea el uso de esta herramienta son las siguientes:

- No permite llevar un seguimiento constante del trabajo.
- No permite crear información detallada de cada tarea.
- No permite guardar históricos de tareas.
- No da gráficas del avance del equipo.
- No permite administrar calendarios.

Tras hacer un análisis de las ventajas y desventajas de la herramienta Trello, se decidió cambiar de software y se comienza con el uso de JIRA como herramienta para scrum.

En las figuras 6 y 7 se muestran las pantallas relacionadas con el software donde podemos ver los distintos estados que podía llegar a tener cualquiera de las tareas, en esta pantalla se agregó un estado extra llamado UET. Este estado es el *user acceptance testing* y es usado por el product owner ya que cuando una tarea ha llegado a este punto es porque el usuario validará el resultado final del proyecto.

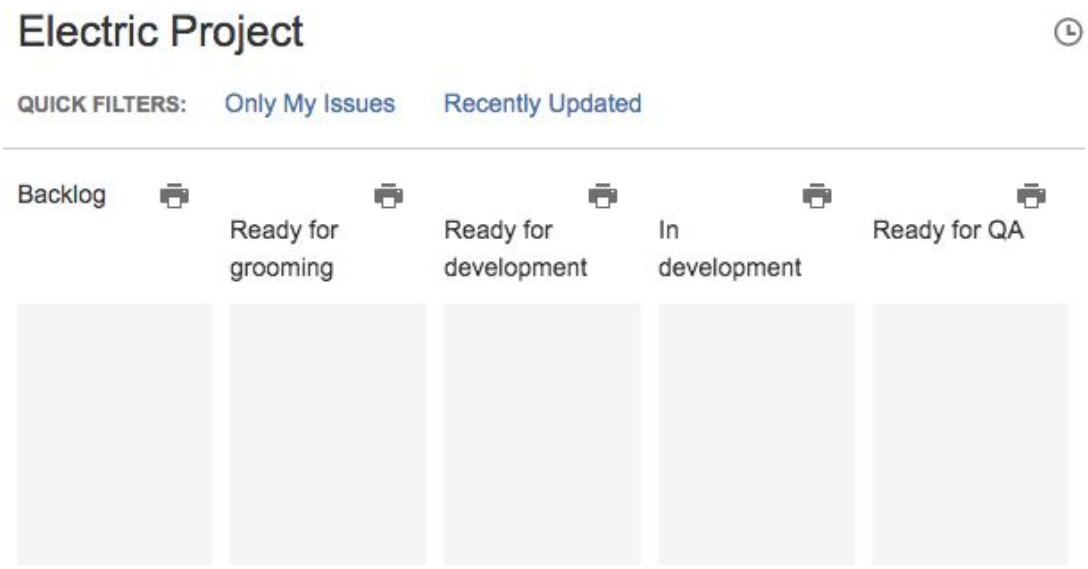


Figura 6. Ejemplo de JIRA

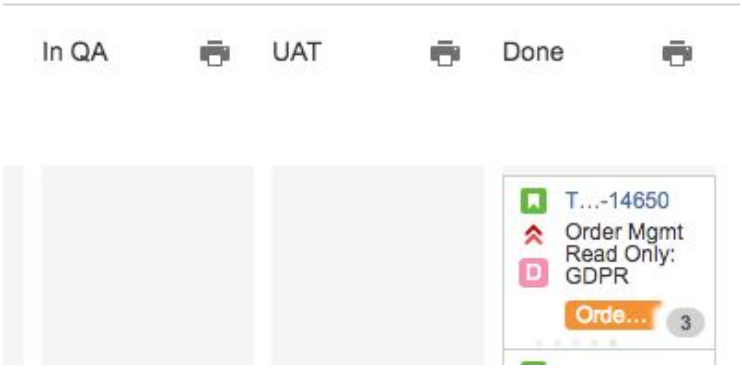


Figura 7. Ejemplo de JIRA

Una de las principales ventajas de JIRA es la generación automática de reportes y gráficas a partir del seguimiento de las tareas dentro de la plataforma, esta clase de gráficas ayudan al equipo a saber cuál es el avance que tiene un equipo y también permiten mejorar su capacidad de trabajo en cada iteración de scrum.

En la figura 8 podemos ver uno de los ejemplo de reportes que nos da JIRA.

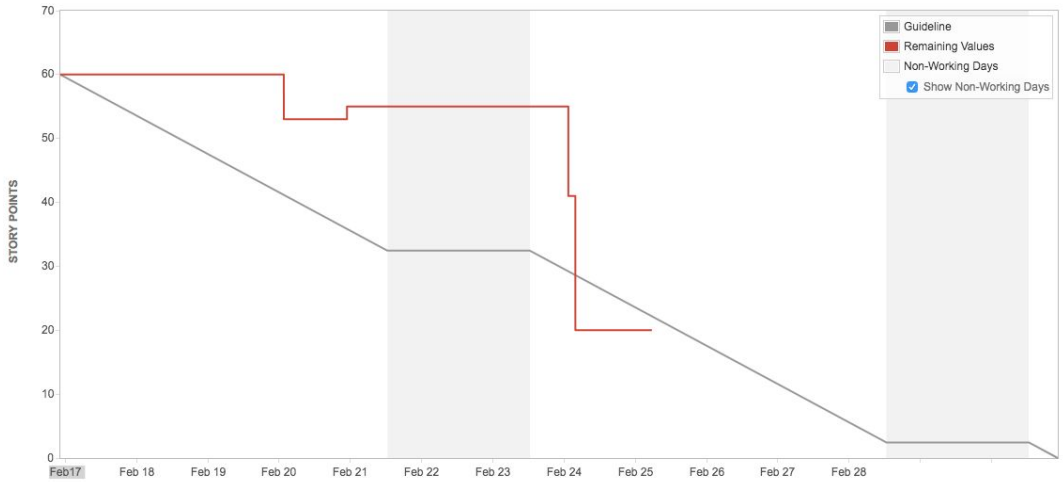


Figura 8. Burndown chart

Las principales ventajas del uso de JIRA son:

- Buen seguimiento de tiempos para el proyecto, tarea y persona.
- Reportes de todos los actores del proceso.
- Integración con tecnologías como controladores de versión.
- Datos procesados en gráficas.
- Seguimiento de errores.

Las principales desventajas:

- La curva de aprendizaje es grande para ser una herramienta de apoyo.
- El equipo puede ser reacio al flujo de trabajo de JIRA.
- El software tiene costo.

Podemos decir que tras la aplicación del software JIRA el monitoreo de los sprint y el scrum mejoró debido a la optimización en tiempo de procesos de planeación y

seguimiento de tareas, todo gracias a las facilidades del programa y a sus múltiples funciones en el seguimiento del trabajo.

3.3.3. Burndown chart

Como se ve en la figura 8, dentro de scrum existen gráficas que nos permiten llevar un seguimiento de manera sencilla sobre cuál es el avance que se está teniendo en cada sprint; en la figura 8 vemos una gráfica llamada burndown chart y dentro de ella una línea de color gris que representa el avance esperado por el scrum master en cada día que tiene el sprint y otra línea de color rojo donde vemos el avance real que se tiene en el sprint.

En el eje de las abscisas se mide el tiempo que se tiene restante para el sprint y en el eje de las ordenadas la cantidad de puntos restante para el mismo, como se ve en la gráfica de la figura 8 las estimaciones no se suelen seguir al pie de la letra, pero lo importante es que podemos comparar el avance real contra el estimado de manera sencilla.

Estos datos son útiles para poder identificar avances lentos, bloqueos en el desarrollo o cualquier factor que esté afectando de manera directa a la maquila del software.

3.4. Aplicación de scrum en el desarrollo

A continuación daré una breve descripción de lo que fue el desarrollo del proyecto. Una parte esencial en la elaboración de proyectos de gran tamaño es la planeación y gracias a la buena planeación que se tuvo antes de iniciar el trabajo se pudo disminuir el desperdicio de tiempo y recursos que, por ende, redujo la cantidad de presupuesto invertido en el desarrollo.

Para el desarrollo del proyecto fueron necesarios 16 sprints. A esto se le sumó el tiempo de instalación y lanzamiento del proyecto en un entorno real.

3.4.1. Equipo de desarrollo

Para el desarrollo de este sistema fue necesario un equipo de 8 personas con los roles enlistados en la tabla 8.

Tabla 8. Equipo de scrum

Rol	**Número de participantes por rol**
Scrum master	1
Product owner	1
Analista de requerimientos	1
Desarrollador	4
Tester	1

3.4.2. Flujo del sprint

El flujo de cada uno de los sprints, como vimos en la metodología, se llevó a cabo de la siguiente forma con cada historia.

1. Se plantean los requerimientos en la bitácora del producto en forma de tareas.
2. Cada una de las tareas se estima dependiendo de su grado de dificultad.
3. Inicia el desarrollo de cada uno de los requerimientos.
4. Al término de las historias se encolan para ser verificadas.

5. Se verifica que cada una de las tareas cumpla con los requerimientos impuestos por el usuario.
6. Se procede a la validación con el cliente para comprobar que la funcionalidad cumpla con lo que se esperaba.
7. Cuando el cliente aprueba las tareas éstas se dan por terminadas.

3.4.3. Desarrollo de sprints

Durante cada uno de los sprints se realizaron distintas tareas y hubo algunos desarrollos que tardaron más de un sprint debido a la complejidad de las tareas, a razón de que no todos los integrantes del equipo de scrum tenían las mismas habilidades y en busca de agilizar el trabajo se implementó la realización de tareas en paralelo a través de los sprints.

A continuación, en la tabla 9 podemos ver los tópicos trabajados durante cada uno de los sprints.

Adicional al tiempo de desarrollo se tomó un tiempo para la implementación del sistema en los equipos de hardware. Este trabajo no estuvo dentro de ningún sprint debido a que no era necesario todo el proceso del sprint, ni requería el esfuerzo de todos los participantes para completar este proceso.

Tabla 9. Ejecución de sprints

Número de sprint	Temas del sprint
1	- Planificación de forma de trabajo
2	- Implementación de Base de Datos - Implementación de Servidor Web - Implementación de Ambiente de Trabajo - Implementación Ambiente de Pruebas
3	- Configuración de la Base de Datos - Implementación de Ambiente de Trabajo
4	- Desarrollo de API (Registro) - Desarrollo de Lector Medidor
5	- Desarrollo de API (Log in) - Desarrollo de Lector Medidor
6	- Desarrollo de API (Implementación de Permisos) - Desarrollo de Lector Medidor
7	- Desarrollo de API (Implementación de Permisos) - Desarrollo de Lector Medidor
8	- Desarrollo de API (Funciones de Datos) - Desarrollo de Lector Medidor
9	- Desarrollo de API (Funciones de Datos) - Desarrollo de Lector Medidor
10	- Desarrollo de API (Funciones de Datos) - Desarrollo de Lector Medidor
11	- Desarrollo de API (Funciones de Datos) - Desarrollo de Lector Medidor
12	- Desarrollo de API (Funciones de Datos) - Desarrollo de Lector Medidor
13	- Desarrollo de Interfaz Gráfica - Implementación API
14	- Desarrollo de Interfaz Gráfica - Implementación API
15	- Desarrollo de Interfaz Gráfica - Implementación API
16	- Desarrollo de Interfaz Gráfica - Implementación API

Capítulo 4

Diseño de software

Una vez que se supo cuál era la metodología, el marco de trabajo y se tenían claros los requerimientos del sistema se dio inicio a la fase de diseño. Durante esta fase se buscó especificar a nivel técnico el funcionamiento del software, desde cómo interactúan los usuarios con el sistema hasta la forma en que están relacionados los componente físicos y lógicos que comprenden el proyecto.

Para el diseño del software es crucial tomar en cuenta los requerimientos que se recaudaron previamente, con éstos se pueden hacer sugerencias de tipo técnico y tener una idea clara de cuáles serán los alcances reales del proyecto, siempre tomando en cuenta variables como lo son las limitaciones tecnológicas, de tiempo y de presupuesto.

El diseño del software nos dará como resultados la arquitectura del software, la estructura de los datos, el detalle procedimental y las interfaces. El diseño interpreta los requerimientos para lograr una representación del software, a partir de la cual se podrá desarrollar el sistema.

4.1. Arquitectura del software

La arquitectura del software se define como una serie de abstracciones que nos proporcionan un marco de referencia para desarrollar un sistema de software; dicho de otro modo son una serie de formas y guías que nos sirven para llevar a cabo un proyecto de software, indican el funcionamiento general y la interacción de las distintas partes del sistema.

La arquitectura de software nos brinda los planos y guías bien definidas para la ejecución del desarrollo de software, del mismo modo que los arquitectos lo hacen con los trabajadores de la construcción.

4.1.1. Arquitectura SOA

Dentro de la computación existen distintos tipos de arquitectura de software, cada uno de ellos útil dependiendo de las distintas situaciones.

Para este proyecto usamos una arquitectura SOA (*Service Oriented Architecture*) o Arquitectura Orientada a Servicios. A continuación explicaremos en qué consiste y por qué se decidió usar este tipo de implementación en el proyecto.

SOA, como lo dice su nombre, está orientada a desglosar las actividades de un sistema en distintos servicios, donde cada servicio es una representación de una actividad. Estos servicios se proveen a otros agentes por medio de un canal y este canal suele ser la red por medio de protocolos HTTP.

4.1.1.1. Manifiesto SOA

SOA se rige por un conjunto de principios que son descritos dentro del manifiesto SOA y con ellos se busca entender cuáles son las ventajas y razones por las que deberíamos usar una arquitectura SOA.

Tomando en cuenta el manifiesto SOA y algunos de sus principios más relevantes:

- Priorizar lo que le da valor al negocio.
- Buscar metas del proyecto sobre metas individuales.
- Buscar la flexibilidad en el desarrollo.
- Busca la escalabilidad.

Podemos decir que lo que se busca es la utilización de funcionalidades individuales, bien estructuradas, organizadas y uniformes para crear un proyecto, que permita el rápido desarrollo, así como obtener ventajas en el largo plazo para escalabilidad o mantenimiento.

4.1.1.2. Componentes de SOA

Dentro de SOA existen dos roles principales: el proveedor de servicio y el consumidor del servicio:

- El consumidor del servicio: Es aquel que interactúa con el proveedor de servicios, pueden ser aplicaciones web, aplicaciones móviles, etc.

- El proveedor de servicios: Son todos los servicios que se encuentran dentro de la arquitectura.

Estos dos roles dentro de la arquitectura SOA interactúan entre ellos por medio de un canal de comunicación que proporciona una conexión entre los dos participantes, ésta puede ser de distintos tipos (inalámbrica, cableada, etc.) siempre y cuando se llegue a una conexión punto a punto, la forma más común de lograr este tipo de interacción es por medio de protocolos de comunicación HTTP que permiten a dos puntos conectados intercambiar información por un mismo canal.

En la figura 9 podemos ver que el diagrama se divide en consumidor del servicio, canal de comunicación y el proveedor de servicios, donde el consumidor del servicio consulta y recibe datos desde el proveedor de servicios (servidor) y éste a su vez envía datos, recibe instrucciones e interactúa con la base de datos para el funcionamiento del sistema, todo a través de un canal de comunicación que para este caso es la red y se hace por medio de protocolos HTTP.

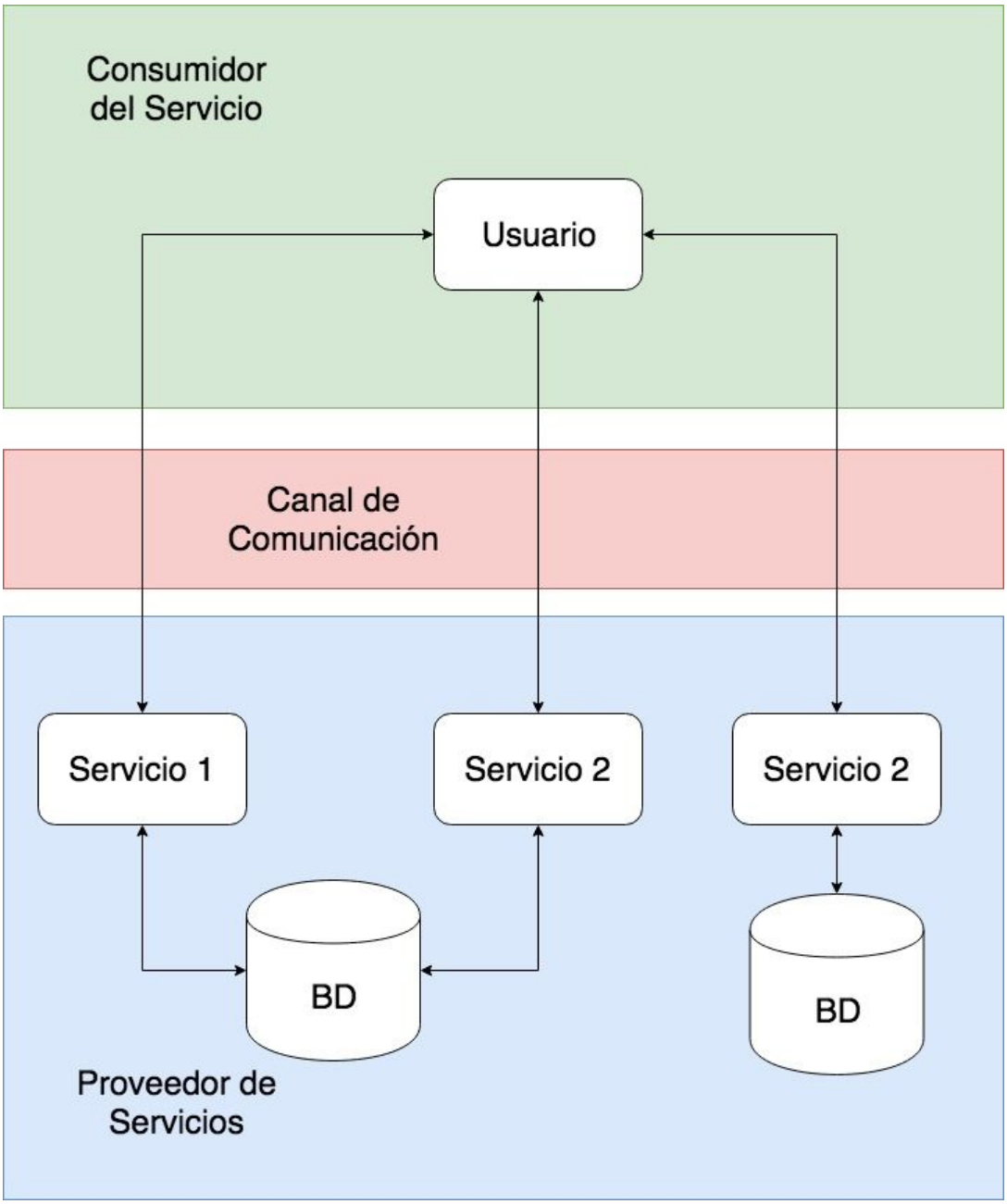


Figura 9. Diagrama SOA

El diagrama de SOA de la figura 9 aparte de ejemplificar el funcionamiento de esta arquitectura también nos da una perspectiva general de cómo fue implementado el proyecto y cuáles son los flujos de comunicación en el sistema.

Ahora que se tiene claro qué es una arquitectura SOA hay que entender cuáles fueron las principales ventajas de la arquitectura y las razones por las que fue implementada en el proyecto.

- Los requerimientos del proyecto fueron una de las principales razones para la utilización de la arquitectura, la escalabilidad dentro de SOA es sencilla porque el desarrollo contiene funciones bien definidas y aisladas.

- SOA puede consumirse desde distintas plataformas por medio de un API. Como lo indica el proyecto, a pesar de que en la primera versión sólo existe una página web, en futuras versiones se busca la expansión a dispositivos móviles.

- El mantenimiento tanto correctivo como perfectivo son sencillos gracias al grado de abstracción que tienen las funciones en este tipo de arquitectura.

- Asegura un buen rendimiento, confiabilidad y disponibilidad dentro de la aplicación.

4.1.2. Justificación de tecnología en la arquitectura

Durante la implementación de la arquitectura hay que hacer la planeación no sólo de cada componente, también de la tecnología que tendrá cada uno de éstos. Dentro del desarrollo de la arquitectura mostraré cuál fue la tecnología utilizada en cada uno de los distintos componentes, así como la justificación de por qué se decidió utilizar cada tecnología.

Los criterios para la elección de cada tecnología fueron los siguientes:

- Que la tecnología estuviera acorde a las necesidades de la arquitectura.

- Usar tecnológicas actuales con librerías que faciliten el desarrollo del proyecto.

- Se dispusiera de conocimientos para el desarrollo en las tecnologías seleccionadas.

4.1.3. Modelos de arquitectura

Para poder representar de manera precisa la arquitectura del sistema y sus componentes se desarrollaron ayudas gráficas que representan la implementación, lógica, física y funcional de la aplicación con al finalidad de facilitar el entendimiento entre la planeación y el desarrollo.

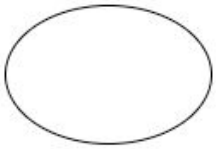
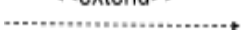
4.1.3.1. Vista de escenarios

La vista de escenarios fue implementada por medio de un diagrama de casos de uso donde se buscó definir el comportamiento del sistema sin entrar a detalle con la estructura interna del proyecto.

Para este diagrama hay que especificar quiénes interactúan con cada uno de los casos de uso que se definieron en los requerimientos y también hay que retomar la definición de actores planteada en la misma fase. Se definió que existirán tres tipos roles de usuarios y seis casos de uso, cada uno de esos roles de usuario serán nuestros actores dentro del diagrama de casos de uso y dependiendo del rol se tendrá una interacción con cada uno de los casos de uso.

En la tabla 10 mostraremos cuál es la notación usada dentro del diagrama de casos de uso:

Tabla 10. Notación para diagramas de casos de uso

Simbolo	Elemento que representa
	Marco para delimitar el sistema.
	Este símbolo ilustra a un caso de uso.
	Símbolo que refiere a los actores del sistema.
	Interacción entre actores y casos de uso.
	Este tipo de unión indica que se necesita alguna acción extra para la realización de los casos de uso.
	Esta unión indica que se hará uso de alguna acción complementaria a esta acción.

4.1.3.1.1 Diagrama de casos de uso

En la figura 10 podemos ver la representación del diagrama de casos de uso.

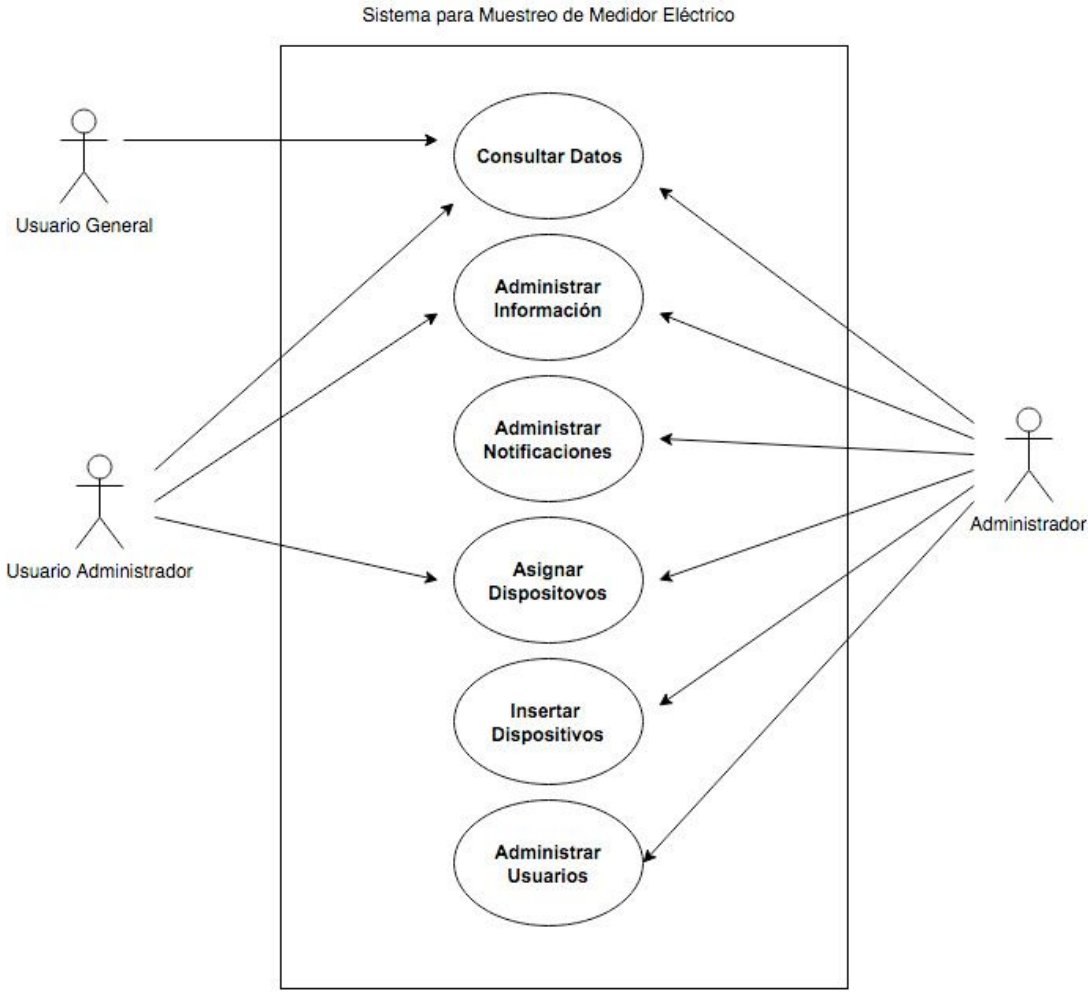


Figura 10. Diagrama de casos de uso

4.1.3.2. Vista lógica

La vista lógica busca describir la estructura y funcionalidad de un sistema en lo que a software se refiere. Para el proyecto se utilizó un diagrama de clases que permitió realizar una representación gráfica de la interacción de los distintos componentes de software en el sistema.

Dentro de un diagrama de clases existen distintos elementos de los cuales hablaremos a continuación:

Una clase es una representación de un elemento dentro del software, está compuesta por tres elementos: nombre de la clase, atributos y los métodos de la clase.

Entendamos por atributos a la información que necesita una clase para su funcionamiento; esta información la puede recibir de un agente externo o la puede tener definida por defecto.

En el diagrama de clases, la forma de denotar una clase es como se muestra en la figura 11:

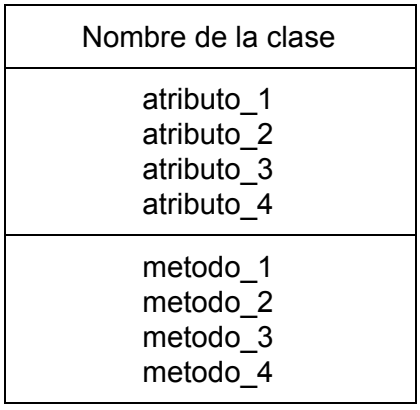


Figura 11. Representación de una clase

Los atributos de una clase pueden tener tres distintos grados de visibilidad:

- Público (public): Indica que el atributo será visible para todos los elementos propios o ajenos a una clase, su representación es el símbolo **+**.

- Privado (private): Denota que el atributo sólo es accesible por los métodos dentro de su clase, su representación es el símbolo **-**.

- Protegido (protected): Señala que el atributo sólo es accesible por otros métodos y también por los métodos de clases que hereden de ésta, su representación es el símbolo **#**.

La forma en que se usan los atributos dentro de clase es la siguiente:

Grado de visibilidad (+,-,#) Nombre : Tipo de Dato [= Valor Inicial]

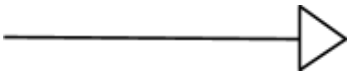
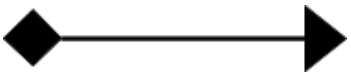
[ ] - Los elementos entre corchetes cuadrados son opcionales.

Los métodos tienen los mismos distintos grados de visibilidad que los atributos.

- public**:** su representación es el símbolo **+**.
- private**:** su representación es el símbolo **-** .
- protected: su representación es el símbolo **#**.

Para el diagrama de clases se tienen distintos tipo de relaciones entre clases y cada una de ellas usa su propia simbología, en la tabla 11 la podemos ver.

Tabla 11. Relaciones en diagrama de clases

Tipo de Relación	Símbolo	Definición
Asociación		Se usa cuando dos clases interactúan entre sí como parte de la funcionalidad del sistema.
Especialización		Se usa cuando se busca indicar que una clase es subclase de otra, esto significa que hereda atributos y métodos dependiendo del nivel de visualización de cada clase.
Agregación		Este tipo de relación indica que para la existencia de una clase es necesaria la existencia de otra.

Dentro del diagrama de clases existe la multiplicidad y se indica en cada extremo de la relación; existen los siguiente tipos de relación:

- Uno a muchos.
- 0 a muchos.
- Muchos a muchos.

Cabe resaltar que muchos de los elementos teóricos de este diagrama, pudieran no aparecer porque las reglas de negocio iniciales no dan información suficiente para una planeación completa y minuciosa del diagrama de clases; sin embargo se debe intentar crear una representación que diste lo menos posible de la realidad.

4.1.3.2.1. Diagrama de clases

En la figura 12 podemos ver la representación del diagrama de clases.

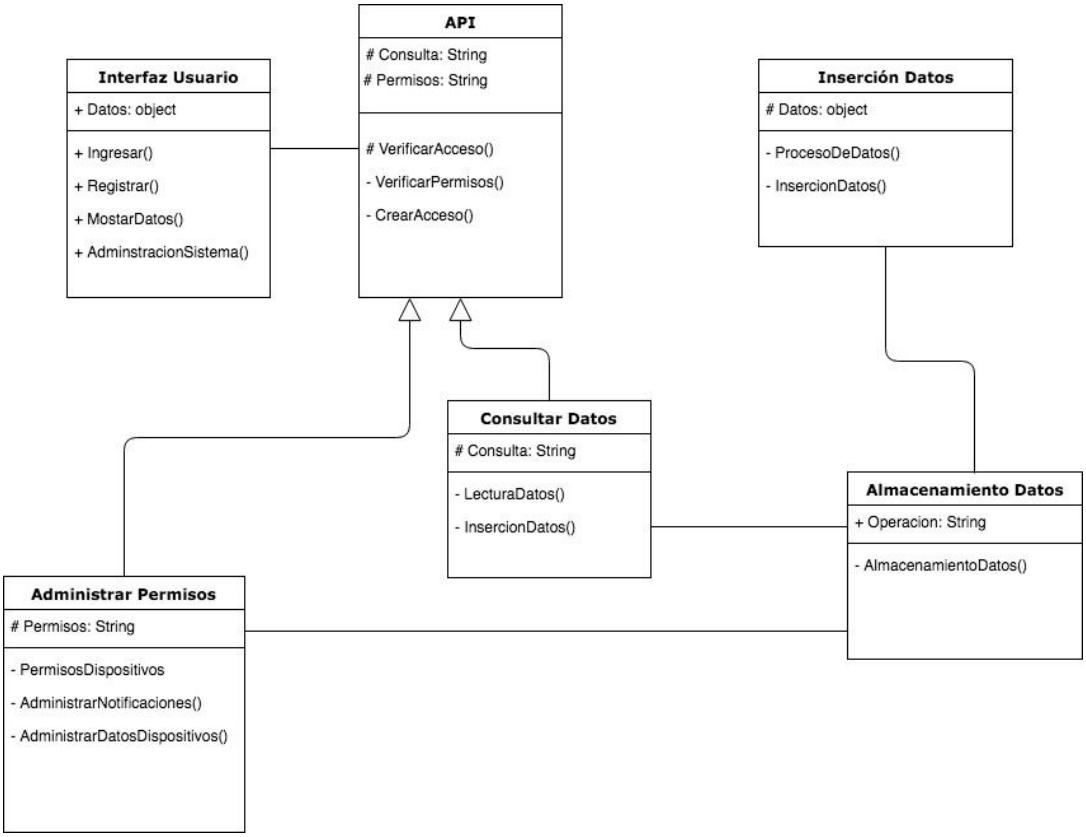


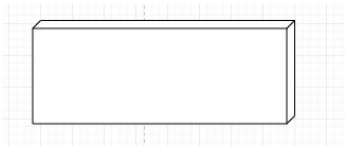
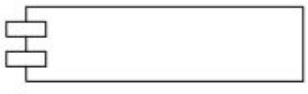
Figura 12. Representación de una clase

4.1.3.3. Vista fisica

La vista física busca dar una perspectiva de cómo se distribuye el software en el hardware del sistema, así como representar la interacción entre los distintos componentes de hardware.

Dentro de un diagrama de despliegue se tiene los elementos de la tabla 12.

Tabla 12. Simbología en vista física

Elemento	Representación	Descripción
Nodo		Un nodo es un componente de hardware.
Componente		Un componente es una instancia de software que se encuentra dentro de un nodo.
Union entre nodos		Denota una comunicación entre componente de hardware por medio de algún canal de comunicación.
Union entre componentes		Representa comunicación entre software por medio de algún protocolo de comunicación.

4.1.3.3.1. Diagrama de despliegue

En la figura 13 podemos ver la implementación mediante el diagrama de despliegue.

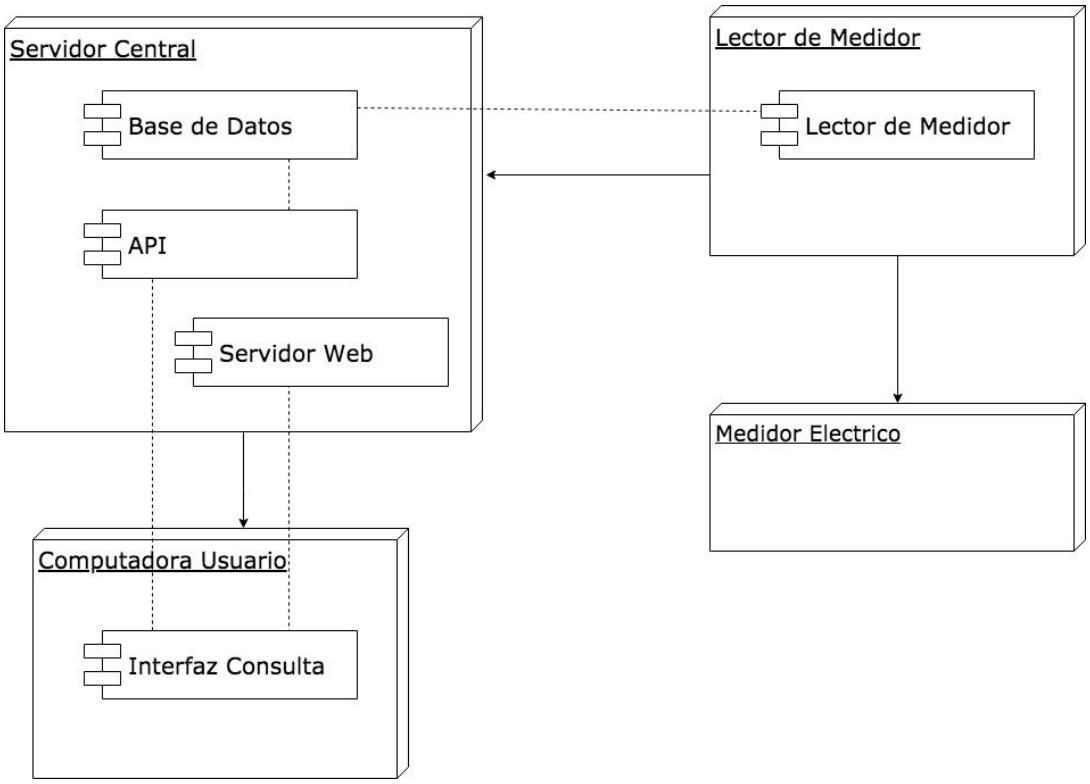


Figura 13. Diagrama de despliegue

4.1.3.4. Tecnología para lectura de medidor

Para la implementación del software de lectura del medidor se utilizó el lenguaje de programación Python, éste es un lenguaje de programación interpretado y que respeta el paradigma de programación orientado a objetos.

Las principales razones por las cuales fue utilizado este lenguaje de programación para la interacción del medidor eléctrico y sistema fueron las siguientes:

- Python cuenta con librerías (MODBUS) que permiten la fácil interacción entre el dispositivo eléctrico y el software.

- Python puede utilizar tanto librerías como funciones nativas para facilitar el procesamiento y consulta de datos del dispositivo eléctrico.

- Python tiene librerías que facilitan las operaciones con la base de datos.

4.1.3.5. Tecnología en API

Para el caso de este proyecto el API funciona del siguiente modo: Por medio de la interfaz gráfica se hacen peticiones vía HTTP al API y ésta devolverá por el mismo canal que recibe la petición una respuesta procesada en el servidor, podemos ver el ejemplo en la figura 14.

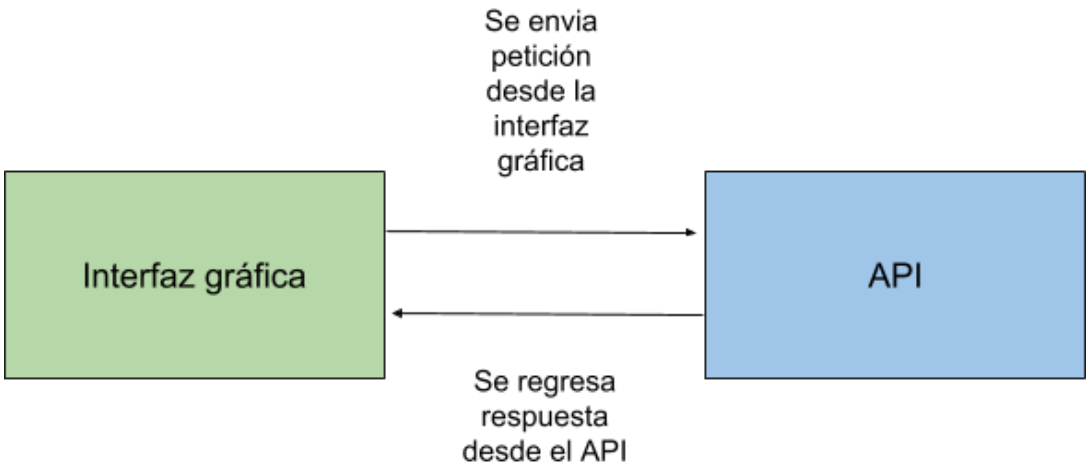


Figura 14. Funcionamiento del API

Para el desarrollo del API se utilizó el lenguaje de programación NodeJS, éste es un lenguaje de programación enfocado a la creación de API's.

Las principales razones por las cuales se utilizó este tipo de implementación fueron las siguientes:

- Librerías disponibles para la implementación de API's.

- Buen lenguaje de programación para el servidor y permite habilitar la interfaz de consulta en distintos tipos de hardware (PC y dispositivos móviles).

4.2. Detalle procedimental

El detalle procedimental nos da una perspectiva de cómo será el funcionamiento de nuestro proyecto.

Dentro del sistema existen siete actividades principales que debe cumplir el sistema:

1. Inicio de sesión y registro de usuarios: Esta actividad la realizan todos los usuario del sistema y sirve para que los usuario puedan acceder a la información.

2. Ver medidores disponibles: Cada usuario que entra al sistema podrá ver la cantidad de medidores sobre los que tiene permiso de consulta.

3. Muestra de datos disponibles: El sistema mostrará a los usuario los datos que tienen permisos de visualizar dentro de cada medidor que tengan asignado.

4. Mostrar notificaciones: El sistema mostrará notificaciones y alertas dentro del sistema a todos los usuarios, éstas serán generales o referentes a un dispositivo que haya sido asignado.

5. El sistema permitirá tanto al usuario administrador como al administrador la asignación de dispositivos a los demás usuarios.

6. El administrador y el usuario administrador podrá restringir la cantidad de datos que visualizan los usuarios asignados a cada dispositivo.

7. El administrador podrá dar de alta nuevos medidores.

En el diagrama de la figura 15 podemos ver el flujo que tienen estas acciones dentro del sistema.

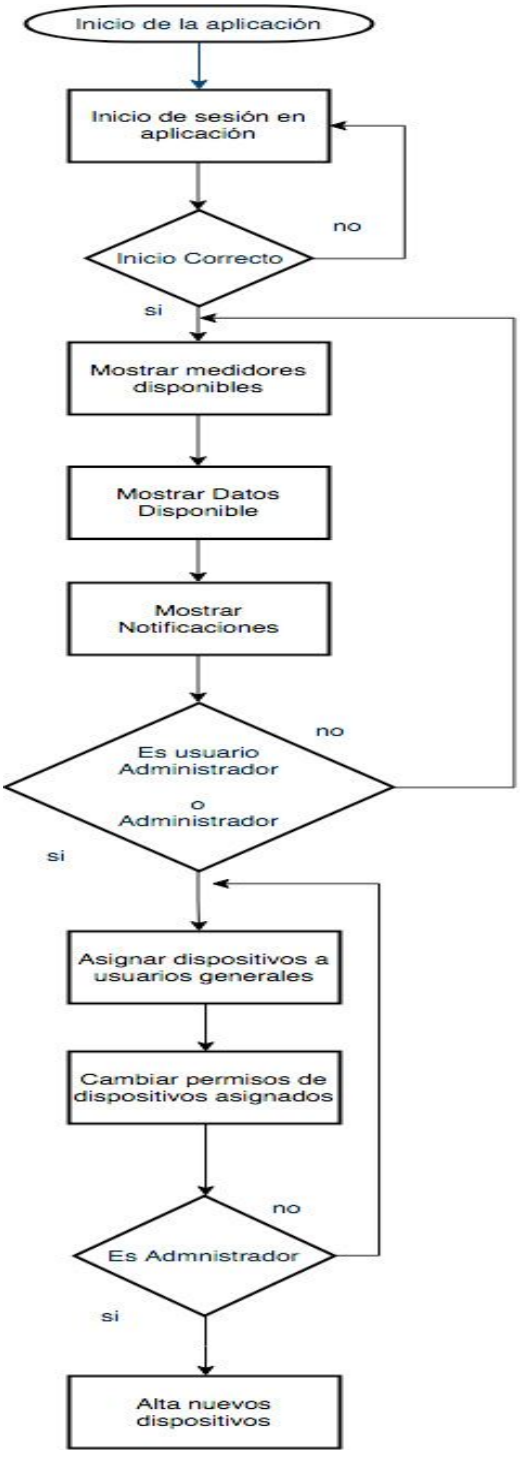


Figura 15. Flujo del sistema

4.3. Estructura de datos

El modelado de datos dentro de un sistema de software es crucial en el desarrollo de cualquier plataforma. Un buen modelado brinda facilidades para el diseño de la arquitectura y es indispensable para un desarrollo de software sin contratiempos.

A continuación en la figura 16 se muestra el diagrama entidad relación y en la figura 17 el modelo relacional utilizado durante el proyecto, con la finalidad de esclarecer cómo es el manejo de los datos dentro del sistema.

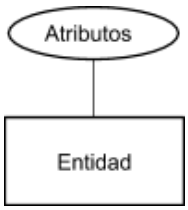
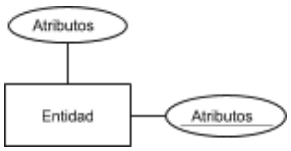
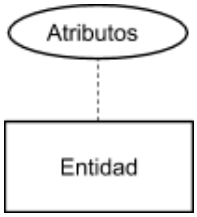
El modelo nos ayudó a dejar en claro cuáles son las relaciones que tienen los distintos datos del sistema y también nos ayuda a tener una perspectiva clara de qué y cómo es lo que se guarda en la base de datos. Todo esto ayudó a encontrar cuál era el camino técnico para resolver problemas durante la implementación del código.

4.3.1. Diagrama entidad relación de la base de datos

El diagrama entidad relación nos permite plantear de manera conceptual el diseño de una base de datos, permitiéndonos tener los primeros indicios de cómo será el diseño final de nuestra base de datos.

En la tabla 13 podemos ver la notación que se ocupó para el diagrama entidad relación.

Tabla 13. Simbología del diagrama entidad relación

Elemento de la simbología	Símbolo	Descripción
Entidad		Representación de una entidad.
Atributo		Los atributos se representan por medio de óvalos que se unen por una línea con la entidad.
Llave primaria (pk)		Los atributos subrayados representan al atributo que será la llave primaria de la entidad.
Atributos opcionales		Aquellos atributos unidos a la entidad por medio de una línea punteada serán atributos opcionales.
Unión 1:M		Este elemento representa la unión 1:M entre dos entidades.
Unión 1:1		Ésta es la representación 1:1 entre dos entidades.
Unión N:M		Representación de la unión M:N entre dos entidades.

El diagrama entidad relación se muestra en la figura 16.

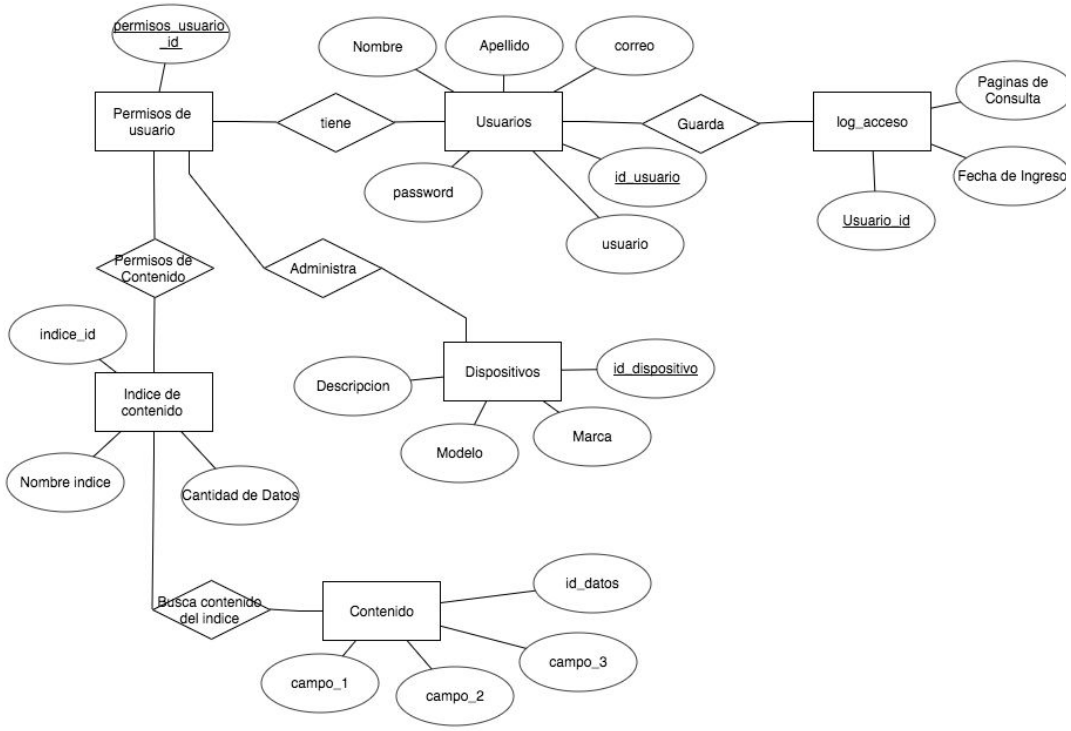


Figura 16. Diagrama entidad relación

El diagrama entidad relación nos permite representar las entidades relevantes de cualquier sistema y su interacción con las otras de manera sencilla, para el caso del proyecto el tener este diagrama nos permitió elaborar un mejor modelo relacional que estuviera acorde a las ideas aquí planteadas.

Este diagrama nos permitió tener una idea clara de qué entidades se necesitarían en nuestro sistema a partir de los requerimientos a solventar, una vez que se tuvieron claras las principales entidades se realizó el proceso de identificar cómo se relacionarán unas entidades con otras, posteriormente se adicionaron los principales atributos de cada entidad al diagrama. El diagrama entidad relación nos permite tener una idea clara de cómo será el diseño de una base de datos y a partir de éste refinar el concepto para obtener un diagrama relacional implementable en una base de datos.

4.3.2. Modelo relacional de la base de datos

El modelo relacional de la base de datos se muestra en la figura 17.

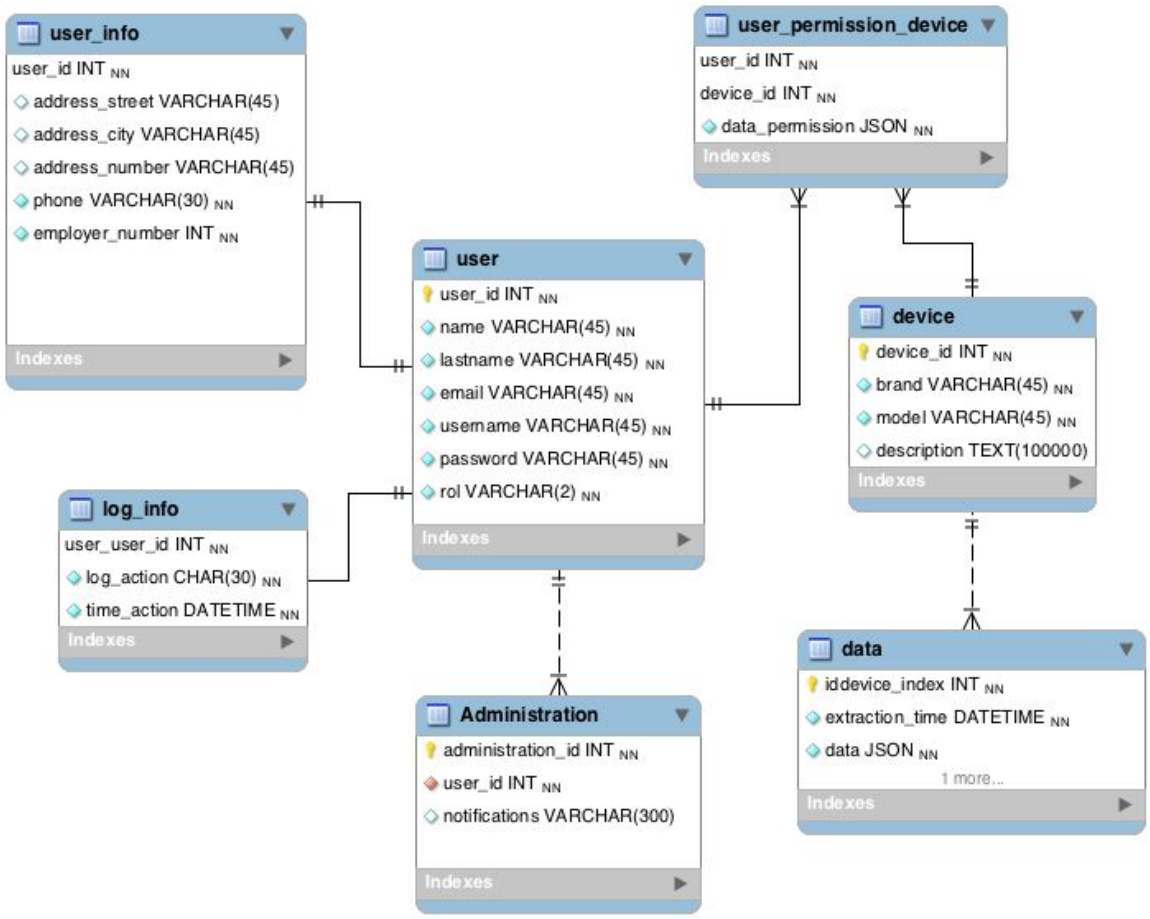


Figura 17. Modelo de la base de datos

Con respecto al diseño de la base de datos, vale la pena resaltar que la base de datos está normalizada hasta la tercera forma normal, lo que significa que cumple con los siguientes atributos:

- Primera forma normal: Se cumple la primera forma normal cuando un atributo no puede tomar más de un valor de una tupla, dicho de otro modo, se encuentra en su primera forma normal si no permite redundancias.

- Segunda forma normal: Una base de datos se encuentra en segunda forma normal si cumple con la primera forma normal y elimina todas las dependencias parciales dentro de una tabla (elementos que no dependen de la llave primaria para identificarlos).

- Tercera forma normal: Para llegar a esta forma normal se deben cumplir las dos primeras formas normales y a parte todos los atributos de una entidad dependen funcionalmente de su llave foránea.

4.3.3. Tecnología en la base de datos

Para la base de datos dentro del sistema se utilizó el manejador de bases de datos PostgreSQL. Este manejador de base de datos utiliza el modelo relacional.

Las principales razones por las que fue utilizado esta manejador de bases de datos fueron las siguientes:

- El software no tiene ningún costo de implementación (descarga e instalación).
- Se puede hacer uso del lenguaje SQL para el desarrollo dentro de la base de datos.
- La base de datos soporta el tipo de dato JSON.
- Tiene soporte para el almacenamiento de datos de tipo espacial.

A pesar de las grande ventajas que tiene este manejador de bases de datos también tiene algunas desventajas que se presentaron durante el proyecto:

- El soporte de la base de datos tiene un precio elevado (optimización y mantenimiento).
- La configuración de la base de datos puede llegar a ser complicada.

4.4. Interfaces

Las interfaces dentro del proyecto nos ayudan con la interacción entre el usuario y el sistema (lenguaje de acción) y en segunda instancia también nos sirve como una presentación del sistema para el usuario.

La interfaz de consulta fue implementada en un navegador web y se utilizaron distintos lenguajes de desarrollo:

- HTML
- CSS
- JavaScript

Estos lenguajes son los principales en el desarrollo de aplicaciones y páginas web; al día de hoy prácticamente todas la páginas que se encuentran en Internet tienen su interfaz desarrollada de manera directa o indirecta con estos lenguajes.

El desarrollar las interfaces en estos lenguajes de programación permitió la compatibilidad del proyecto con cualquier computadora que tuviera un explorador de internet instalado, con estas tecnologías logramos solventar cualquier problema relacionado con las distintas formas que se necesitaban para poder brindar acceso a la información de manera amigable y entendible.

4.4.1. Tecnología en servidor web

Para la implementación del servidor web se utilizó la tecnología XAMPP. Es un paquete de software libre que consiste en la implementación de un servidor web, junto con un manejador de bases de datos; para el caso del proyecto sólo se hizo uso del servidor web Apache.

4.4.2. Construcción del sistema

El API del sistema fue desarrollado en el lenguaje de programación NodeJS, con la ayuda del framework SailsJS. Este framework nos ayuda a implementar un API con un patrón de diseño preestablecido, con el cual la elaboración se volvió más sencilla, rápida y ordenada.

A continuación en la figura 18 se muestra parte del controlador del API enfocado a la administración de datos del usuario.

```
var UserController = {

   index: function(req, res) {

       User.findAll(function(err, users) {

       });
   },

   new: function(req, res) {
       res.view();
   },

   create: function(req, res) {
       var params = User.extend(req.query || {}, req.params || {}, req.body || {});

       User.create(params, function userCreated(err, createdUser) {

       });
   },

   show: function(req, res) {

       var id = req.param('id')

       if (!id) return res.send("ID no especificado.", 500);

       User.find(id, function userFound(err, user) {

           res.view({
               user: user
           })
       });
   },
```

```
    edit: function(req, res) {
        var id = req.param('id');

        if (!id) return res.send("ID no especificado", 500);

        User.find(id, function userFound(err, user) {

            res.view({
                user: user
            })
        });
    },

    update: function(req, res) {

        var params = User.extend(req.query || {}, req.params || {}, req.body || {});
        var id = params.id;

        if (!id) return res.send("ID no especificado.", 500);

        User.update(id, params, function userUpdated(err, updatedUser) {

        });
    },


    destroy: function(req, res) {
        var id = req.param('id');
        if (!id) return res.send("Id no especificado", 500);

        User.find(id, function foundUser(err, user) {

            if (!user) return res.send("No existen usuario con ese ID.", 404);

            User.destroy(id, function userDestroyed(err) {

            });

        })
    }

};

module.exports = UserController;
```

Figura 18. Representación del controlador de usuario

4.4.3. Pruebas en el sistema

Dentro del desarrollo del proyecto es necesario realizar validaciones y verificaciones de lo que se está realizando, para este proyecto existían dos fases de validación con el usuario y una de verificación.

Los procesos de validación se hacían en dos partes del proceso de desarrollo:

- La primera validación se hace al tomar los requerimientos para el proyecto, se recaudan las necesidades del cliente y se estipulan criterios de aceptación donde se estipula que debe hacer el software.

- La segunda validación la hace el cliente, comprueba el producto terminado con respecto a cada uno de los criterios de aceptación, si se cumplen todos el cliente da por terminado ese desarrollo.

El proceso de verificación se realiza durante la implementación del código y se hace de dos formas distintas:

- La primer forma de verificación fue de forma automática, donde desde el código y gracias a librerías especiales para testing se hacen pruebas de caja negra. Estas pruebas consisten en verificar que el resultado de una sección de código sea el esperado.

- La segunda forma de verificación fue por medio de pruebas manuales, este tipo de pruebas intenta emular el uso cotidiano que se le dará al sistema y busca defectos de cualquier tipo. Las pruebas manuales requieren a una persona que interactúe con el sistema buscando errores o defectos en todos las posibles combinaciones de uso que se pueden dar en el sistema.

Capítulo 5

Resultados, impacto y conclusiones

5.1. Resultados

El desarrollo de este proyecto dio como resultado un sistema para el monitoreo y consulta de medidores eléctricos a distancia que al día de hoy está en funcionamiento y que sentó las bases para la optimización de procesos por medio de software.

Al hacer la comparación de los requerimientos planteados contra los resultados obtenidos se puede decir que el proyecto se cumplió de forma satisfactoria, en las tablas 14, 15, 16, 17, 18, 19, 20 y 21 podemos ver la comparativa de los requerimientos funcionales contra los resultados obtenidos.

Tabla 14. Resultado de requerimiento funcional

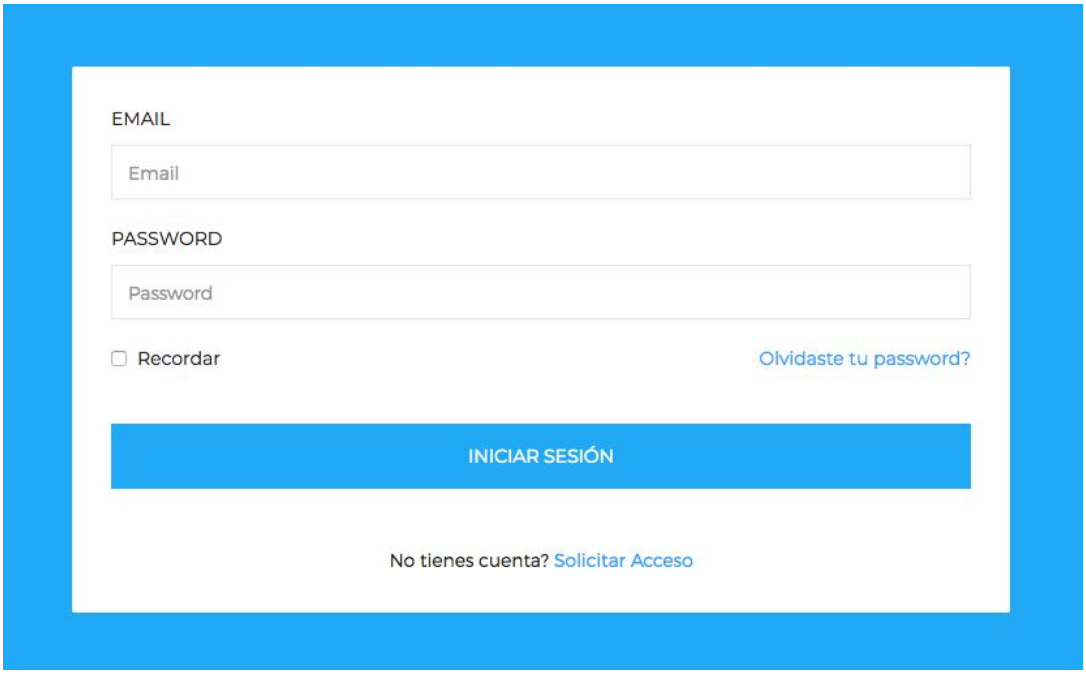
Requerimiento funcional	**Resultado**
Crear un sistema de consulta para todos los datos de distintos medidores.	Este requerimiento se cumplió completamente, el sistema permite consultar datos de múltiples medidores remotos.
	

Tabla 15. Resultado de requerimiento funcional

Requerimiento funcional	**Resultado**
Se contará con un registro de usuarios.	El resultado obtenido fue el deseado, se creó un registro de usuario.

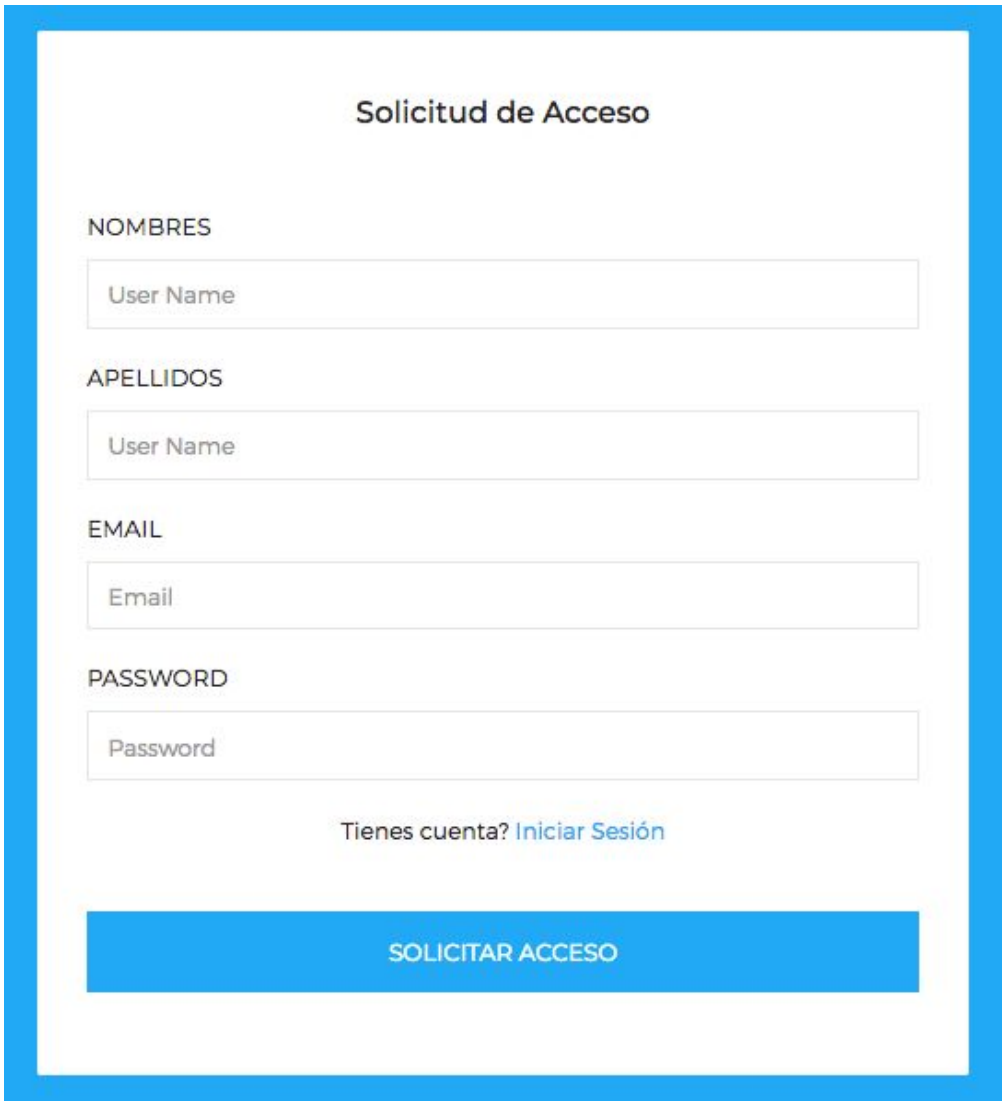


Tabla. 16. Resultado de requerimiento funcional

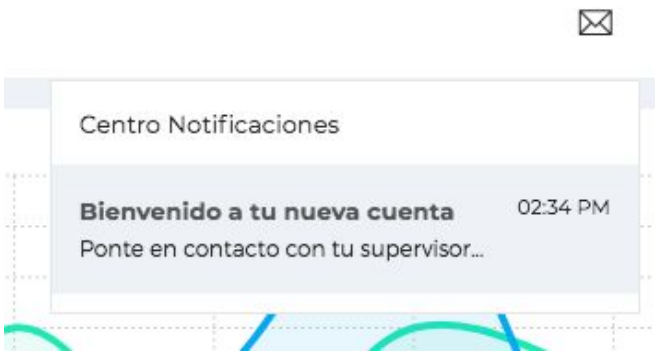
Requerimiento funcional	Resultado
El sistema tendrá notificaciones para cualquier eventualidad.	Las notificaciones fueron implementadas para todos los usuario del modo deseado.
	

Tabla 17. Resultado de requerimiento funcional

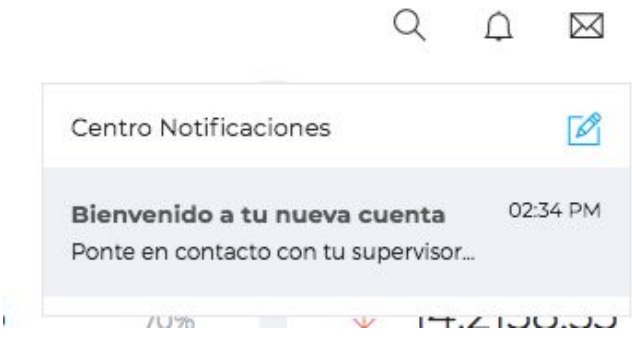
Requerimiento funcional	Resultado
El software permitirá administrar los usuarios y notificaciones.	Este requerimiento tuvo cambios durante el proyecto pero al final se cumplió de forma satisfactoria, ya que se podía administrar usuarios, permisos y notificaciones.
	

Tabla 18. Resultado de requerimiento funcional

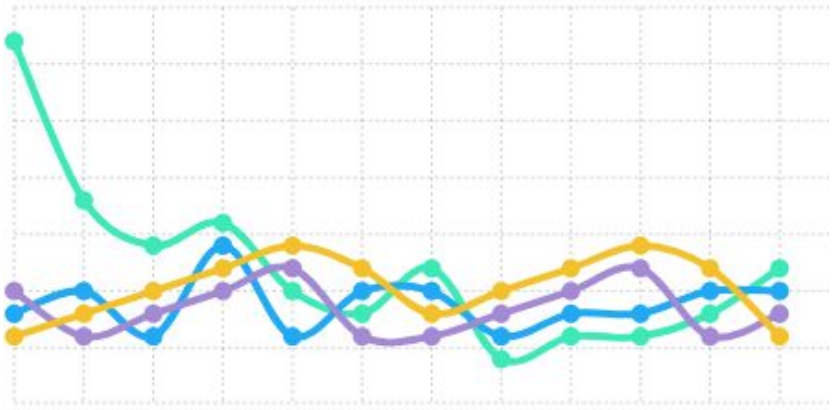
Requerimiento funcional	**Resultado**
El sistema podrá mostrar los datos en tiempo real.	Se cumplió plenamente.
	

Tabla 19. Resultado de requerimiento funcional

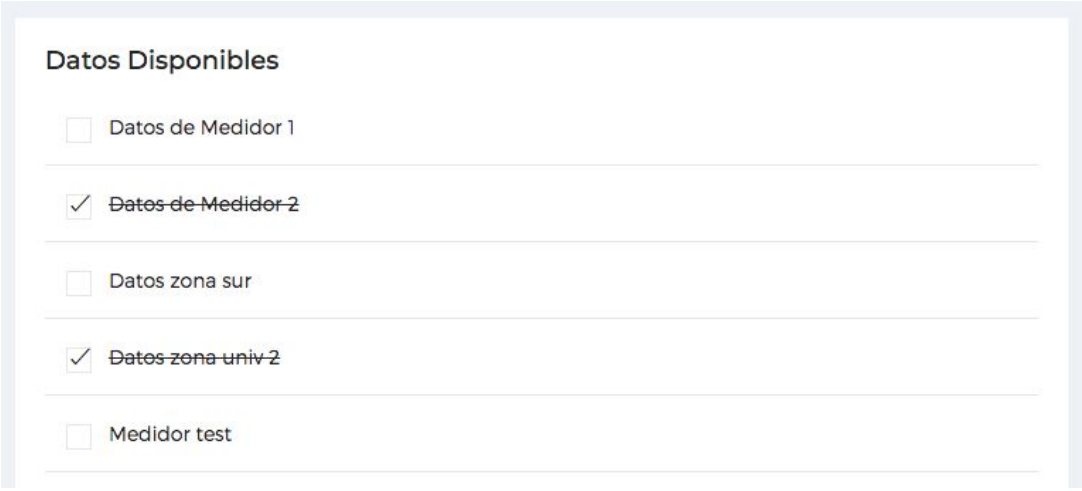
Requerimiento funcional	**Resultado**
El sistema mostrará el histórico de los datos de manera opcional.	Se logró de manera parcial, esto es porque los históricos de los datos son limitados y sólo se alcanza a tener un una cantidad limitada de datos pasados.

Tabla 20. Resultado de requerimiento funcional

Requerimiento funcional	Resultado
Se podrán consultar los datos en forma gráfica y tabular.	El sistema permite esta funcionalidad.

Tabla 21. Resultado de requerimiento funcional

Requerimiento funcional	Resultado
Los datos a mostrar deberán ser configurables.	Se cumplió el requerimiento, se pueden administrar los datos a mostrar por parte de los usuarios con mayor nivel de permisos.



En cuanto a los resultados de los requerimientos no funcionales, los resultados son los que podemos observar en la tabla 22.

Tabla. 22 Resultados no funcionales

Requerimiento no funcional	**Resultado**
El sistema deberá construirse pensando en la escalabilidad del proyecto.	El proyecto fue pensado para la futura escalabilidad, por eso se utilizó la arquitectura SOA y se implementó con tecnologías pensadas en una futura escalabilidad.
El proyecto se implementará en primera instancia en versión web, pero en un futuro podría ser llevado a versión móvil.	Se pensó en esta posible situación y por eso se implementó la arquitectura SOA.
Alta disponibilidad del sistema.	El sistema tiene un alto índice de disponibilidad, sin embargo aún está sujeto al funcionamiento de factores externos.
El sistema deberá ser compatible con el navegador Google Chrome a partir de la versión 5.0.	Se logró plenamente.
El software deberá ser capaz de atender hasta 100 usuarios al mismo tiempo.	El software fue sometido a una prueba de estrés que superó los 100 usuarios.
El desarrollo sólo podrá ser usado desde la intranet del cliente.	Se cumplió.
Se deberá tener un tutorial para el sistema.	Se elaboró un tutorial escrito para el uso del sistema.
Se propone la utilización de un arquitectura basada en servicios.	Se cumplió con la implementación de una arquitectura SOA.
Se propone la creación de un API.	Se creó un API como parte de la arquitectura SOA.

5.2. Impacto

El impacto que tuvo el desarrollo de este software fue muy significativo para el cliente debido a que redujo de manera visible el tiempo, el dinero y el esfuerzo que tomaban los siguientes procesos:

- Consulta de medidores.
- Notificación sobre fallas.
- Validación de datos.
- Detección de errores.
- Revisión de mantenimiento.
- Verificación de disponibilidad.

Para el caso de la consulta de medidores el tiempo que tomaba este proceso a técnicos era de aproximadamente una hora al día más los costos de transporte en algunos casos, gracias a la implementación de este software sólo se necesita acceso a una computadora conectada a la intranet para realizar la revisión diaria de los medidores.

La notificación sobre fallas fue un nuevo proceso que se implementó durante este proyecto, antes para reportar una falla en un medidor o bien en el elemento que muestrea, tenía que existir una notificación por parte de un agente externo, ahora el mismo sistema cuenta con funciones que notifican a los usuarios asignados a un dispositivo en caso de que exista alguna irregularidad.

Gracias a que todos los datos están siendo censados desde el momento en que son leídos del medidor se valida el correcto funcionamiento tanto del medidor como de lo muestreado.

La detección de errores generó un alto impacto para el cliente, gracias al software las fallas son notificadas con mayor velocidad y por ende son atendidas en menor tiempo.

La revisión de mantenimiento es un proceso que se llevaba a cabo con todos los medidores debido a la falta de un registro de datos históricos, ahora que se tienen se puede saber cómo ha sido el funcionamiento de los medidores y decidir a partir de esto si es necesario realizar o no un mantenimiento.

La verificación de disponibilidad dentro de cada medidor es esencial, porque los datos deben de poder ser consultados en todo momento, con esta verificación se puede detectar de manera inmediata si se perdió la comunicación con un medidor y dar alerta de esto.

A partir del desarrollo de este software no todo fue la optimización de procesos, también se generaron nuevos elementos que generan valor para todo aquel que use la plataforma, como lo son:

- La consulta de históricos.
- La visibilidad de la información.
- La disponibilidad de la información.

Al día de hoy los resultados que se han logrado gracias a la implementación de este sistema han sido claros, la inversión que se hizo para realizar este sistema de software se ha visto reflejada en variables dentro del negocio como los son: el ahorro de tiempo, el ahorro de dinero y la mayor productividad en la empresa.

El proyecto ha logrado trascender y ser considerado como exitoso dentro de la compañía debido a las mejoras que ha generado y al efecto positivo que ha tenido. Este sistema es también el primer paso en la búsqueda de innovar varios procesos dentro de la compañía del cliente y ha servido como aliciente para buscar realizar nuevos proyectos e incluso realizar mantenimientos perfectivos a éste.

En conclusión el sistema de monitoreo de un medidor eléctrico a distancia ha sido un éxito debido a su efectividad y a los mejores que ha brindado de manera general a la compañía.

5.3. Conclusiones

El trabajo que realicé durante este periodo de tiempo es la primera versión de un proyecto que al día de hoy continúa en desarrollo, esto en búsqueda de la implementación de una segunda versión para lograr una mayor satisfacción por parte de los clientes. En el trabajo aquí descrito se concentra el desarrollo de cómo fue el proceso de elaboración completo de un proyecto de ingeniería enfocado al software.

Durante este proyecto se creó un sistema que permite el monitoreo remoto de dispositivos de hardware con la finalidad de reducir tiempos en los procesos del cliente. Al día de hoy se puede decir que tras haber finalizado el proyecto y probarlo en un ambiente real, los resultados han sido exitosos y se cumplió el objetivo que se buscaba con este sistema de software.

Tras la implementación, puedo obtener conclusiones en dos ejes centrales:

- La primera de ellas es la importancia de la metodología. Durante mi carrera universitaria recibí dos cursos relacionados con ingeniería de software donde aprendí las bases teóricas de este tópico y las complementé con las experiencias de mis profesores. Sin embargo, una vez que tuve que aplicar todos esos conocimientos logré dimensionar la importancia de estos temas desde una perspectiva diferente. Al día de hoy he logrando entender que el marco de trabajo y la metodología al momento de la implementación, junto con la planificación del software, son elementos fundamentales para determinar el éxito de un proyecto. Al momento de elaborar un sistema de software, una planificación de los componentes adecuada, completa y acorde a las necesidades tanto del usuario como del cliente, dan como resultado un sistema que se adecua completamente a lo que se necesita. Esto evita el desperdicio de cualquier tipo de recursos como lo podrían ser el tiempo y el dinero; una buena ingeniería de software nos ayuda también a asegurar que

el sistema sea lo que se busca, y permite a los clientes tener los resultados esperados para la inversión realizada.

- Las bases de datos son sin duda otra experiencia a reflexionar durante el desarrollo de este proyecto. Las bases de datos son parte crucial para todas los sistemas, sin una buena planeación y diseño inicial de una base de datos se pueden llegar a tener grandes problemas en la implementación de cualquier software que sin duda provocarán retrasos y problemas con el desarrollo del sistema.

Durante esta experiencia profesional logré aprovechar de manera rotunda todos los conocimientos que la Facultad de Ingeniería me dio; sin embargo durante este período pude aprender cosas que sólo pueden entenderse por medio de la experiencia de llevar el conocimiento de las aulas a la vida real. El aprender cómo abordar un problema real, así como realizar una planeación adecuada, el uso de metodologías, la importancia del uso de buenas prácticas y la necesidad de una base de datos bien elaborada en todos los sistemas, son cosas del día a día en la vida de un estudiante de ingeniería en computación, pero para poder tener un ciclo de aprendizaje completo la experiencia es fundamental para cualquier ingeniero.

Bibliografía

Date, C. (2012). *Database design and relational theory : normals forms and all that jazz*. Sebastopol, Calif: O'Reilly Media.

Deitel, H., Deitel, P. & rez, J. (2004). *Cómo programar en C/C++ y Java*. México: Pearson Educación.

Sommerville, I. & Galipienso, M. (2005). *Ingeniería del software*. Madrid: Pearson Addison-Wesley.

Sutherland, J. & Sutherland, J. (2014). *Scrum : the art of doing twice the work in half the time*. New York: Crown Business.

Rumbaugh, J., Jacobson, I. & Booch, G. (2004). *The unified modeling language reference manual*. Reading, Mass: Addison-Wesley.

Rumpe, B. (2017). *Agile modeling with UML : code generation, testing, refactoring*. Cham, Switzerland: Springer.

Unhelkar, B. (2018). *Software Engineering with UML*. Boca Raton, FL: CRC Press.