UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ

Instituto de Ingeniería y Tecnología

Departamento de Ingeniería Eléctrica y Computación

SISTEMA DE MONITOREO Y CONTROL PARA ESTACIONES METEOROLÓGICAS

Reporte Técnico de Investigación presentado por:

Teo González Calzada 142912

Requisito para la obtención del título de:

INGENIERO EN SISTEMAS COMPUTACIONALES

Asesor:

Mtro. José Fernando Estrada Saldaña

Dr. Felipe Adrián Vazquez Galvez

Ciudad Juárez, Chihuahua 9 de mayo de 2022

Ciudad Juárez, Chihuahua, a 9 de Mayo de 2022

Asunto: Liberación de Asesoría

Mtro. Ismael Canales Valdiviezo
Jefe del Departamento de Ingeniería
Eléctrica y Computación
Presente.-

Por medio de la presente me permito comunicarle que, después de haber realizado las asesorías correspondientes al reporte técnico SISTEMA DE MONITOREO Y CONTROL PARA ESTACIONES METEOROLÓGICAS, del alumno Teo González Calzadade la Licenciatura en Ingeniería en Sistemas Computacionales, considero que lo ha concluido satisfactoriamente, por lo que puede continuar con los trámites de titulación intracurricular.

Sin más por el momento, reciba un cordial saludo.

Atentamente:

José Fernando Estrada Saldaña

Profesor Investigador

Ccp:
Coordinador del Programa de Sistemas Computacionales
Teo González Calzada
Archivo

Ciudad Juárez, Chihuahua, a 9 de Mayo de 2022

Asunto: Autorización de publicación

C. Teo González Calzada

Presente.-

En virtud de que cumple satisfactoriamente los requisitos solicitados, informo a usted que se autoriza la publicación del documento de SISTEMA DE MONITOREO Y CONTROL PARA ESTACIONES METEOROLÓGICAS, para presentar los resultados del proyecto de titulación con el propósito de obtener el título de Licenciado en Ingeniería en Sistemas Computacionales.

Sin otro particular, reciba un cordial saludo.

Dr. Nombre del profesor de la materia

Profesor Titular de Seminario de Titulación II

Declaración de Originalidad

Yo, Teo González Calzadadeclaro que el material contenido en esta publicación fue elaborado con la revisión de los documentos que se mencionan en el capítulo de Bibliografía, y que la solución obtenida es original y no ha sido copiada de ninguna otra fuente, ni ha sido usada para obtener otro título o reconocimiento en otra institución de educación superior.

Teo González Calzada

Agradecimientos

Agradezco a mis Padres por haberme dado la oportunidad.

Agradezco a la Ing. Areli Rubio Rodríguez por su inmensa ayuda en la revisión de este documento, y el apoyo con sus conocimientos.

Agradezco a mi asesor el Dr. José Fernando Estrada Saldaña por su enorme paciencia y ayuda con el desarrollo del proyecto, ofreciéndome su apoyo en los tiempos más difíciles y complicados de mi etapa estudiantil.

[Sustituye este texto escribiendo tus agradecimientos. La sección de agradecimientos reconoce la ayuda de personas e instituciones que aportaron significativamente al desarrollo de la investigación. No te debes exceder en los agradecimientos; agradece sólo las contribuciones realmente importantes, las menos importantes pueden agradecerse personalmente. El nombre de la agencia que financió la investigación y el número de la subvención deben incluirse en esta sección. Generalmente no se agradecen las contribuciones que son parte de una labor rutinaria o que se reciben a cambio de pago.

Las contribuciones siguientes ameritan un agradecimiento pero no justifican la coautoría del artículo: ayuda técnica de laboratorio, préstamo de literatura y equipo, compañía y ayuda durante viajes al campo, asistencia con la preparación de tablas e ilustraciones o figuras, sugerencias para el desarrollo de la investigación, ideas para explicar los resultados, revisión del manuscrito y apoyo económico".]

Dedicatoria

A mi padre, que ya no está conmigo, y a mi madre y hermana que lo están, por su apoyo incondicional durante todos estos años.

Índice general

Índice de figuras

Índice de tablas

Resumen

[Sustituye este texto escribiendo tu sinopsis o resumen. Es un panorama general de todo lo que el lector encontrará en tu documento, en no más de una página. Recuerda que este, junto con el título, son la parte más leída de tu documento cuando alguien más lo busca en las bases de datos, el "punto de venta".]

Introducción

[Redacte de manera coherente en una cuartilla cuál es la nueva contribución, su importancia y por qué es adecuado para sistemas computacionales. Se sugiere para su redacción seguir los cinco pasos siguientes: 1) Establezca el campo de investigación al que pertenece el proyecto, 2) describa los aspectos del problema que ya han sido estudiado por otros investigadores, 3) explique el área de oportunidad que pretende cubrir el proyecto propuesto, 4) describa el producto obtenido y 5) proporcione el valor positivo de proyecto.]

Capítulo 1
Planteamiento del Problema

Las redes de monitoreo de meteorológico y de calidad del aire de alta densidad son una necesidad creciente de las áreas urbanizadas, las cuales necesitan de una infraestructura cada vez más compleja para su monitreo y seguimiento. Estas necesidades crecientes de infraestructura han generado una demanda creciente de recursos humanos, y la falta de homogeneidad entre las estaciones de monitoreo presentan retos a conquistar para proveer servicios cada vez más sofisticados. En este capítulo, se discutirá la importancia de la infraestructura de monitoreo y se explicarán los diferentes métodos de monitoreo que existen actualmente para monitorear las estaciones meteorológicas, así como una propuesta de solución para los problemas que los sistemas tradicionales existentes presentan.

1.1. Antecedentes

El desplegar y mantener una red meteorológica urbana compone bastantes retos: Entre la creciente dificultad de crear sistemas de medición estandarizados que se adapten al siempre cambiante paisaje urbano; como la instalación de los equipos de medición y de guardado de datos en áreas que permitan acceso para mantenimiento y que sean seguros; y la dificultad de encontrar un punto de acceso a internet adecuado para transferir la información generada, el generar una red de monitoreo es una tarea extensa y compleja.

Debido a estos retos, la comunidad de monitoreo climatológico y meteorológico se ha enfocado en la creación de sistemas que sean más eficientes y económicos. Entre estos esfuerzos, se encuentra el amplio uso de RaspberryPi como centro de recolección de datos de estaciones de monitoreo [1], tanto caseras como profesionales, con la ayuda de sistemas abiertos para la recolección de datos como lo es WeeWX. Esto ha hecho factible el desplegar redes de 50 nodos de monitoreo con sensores económicamente viables para actores con un presupuesto limitado [2].

Estas redes densas requieren de un monitoreo contínuo para mantener una alta calidad de los datos recabados, y evitar las pérdidas por falta de mantenimiento. Entre los sistemas de monitoreo que pueden ser adaptados para el monitoreo de estaciones meteorológicas y la red que las soporta, se encuentra la plataforma Nagios, el cual es un sistema de monitoreo contínuo orientado a redes y servidores. Entre la información que recaba Nagios contínuamente para el estado de los servidores, se encuentra el uso de CPU y RAM, así como estado de los discos, puertos, e información variada de servicios de red en los hosts. Debido a que Nagios es un sistema de monitoreo de redes orientado a profesionales de la informática, la interfaz gráfica es poco amigable con los usuarios menos familiarizados con los conceptos técnicos de los sistemas computacionales, como se muestra en la Figura 1.1.

Si bien Nagios ofrece la posibilidad de monitorear parámetros adicionales con un sistema establecido de plugins en python y otros lenguajes, además de poseer una fuerte comunidad que crea contínuamente plugins relacionados con el proyecto, hasta el momento la cantidad de plugins relacionados con estaciones meteorológicas existentes es mínima ya que los esfuerzos de la comunidad se centran principalmente en el monitoreo de centros de datos, redes y routers.

Además de los plugins existentes en la comunidad de Nagios, existe la alternativa abierta conocida como *monitoring plugins* [3], que es una plataforma compatible con diversos sistemas de monitoreo de redes y servicios, en la cual es posible encontrar una mayor cantidad de

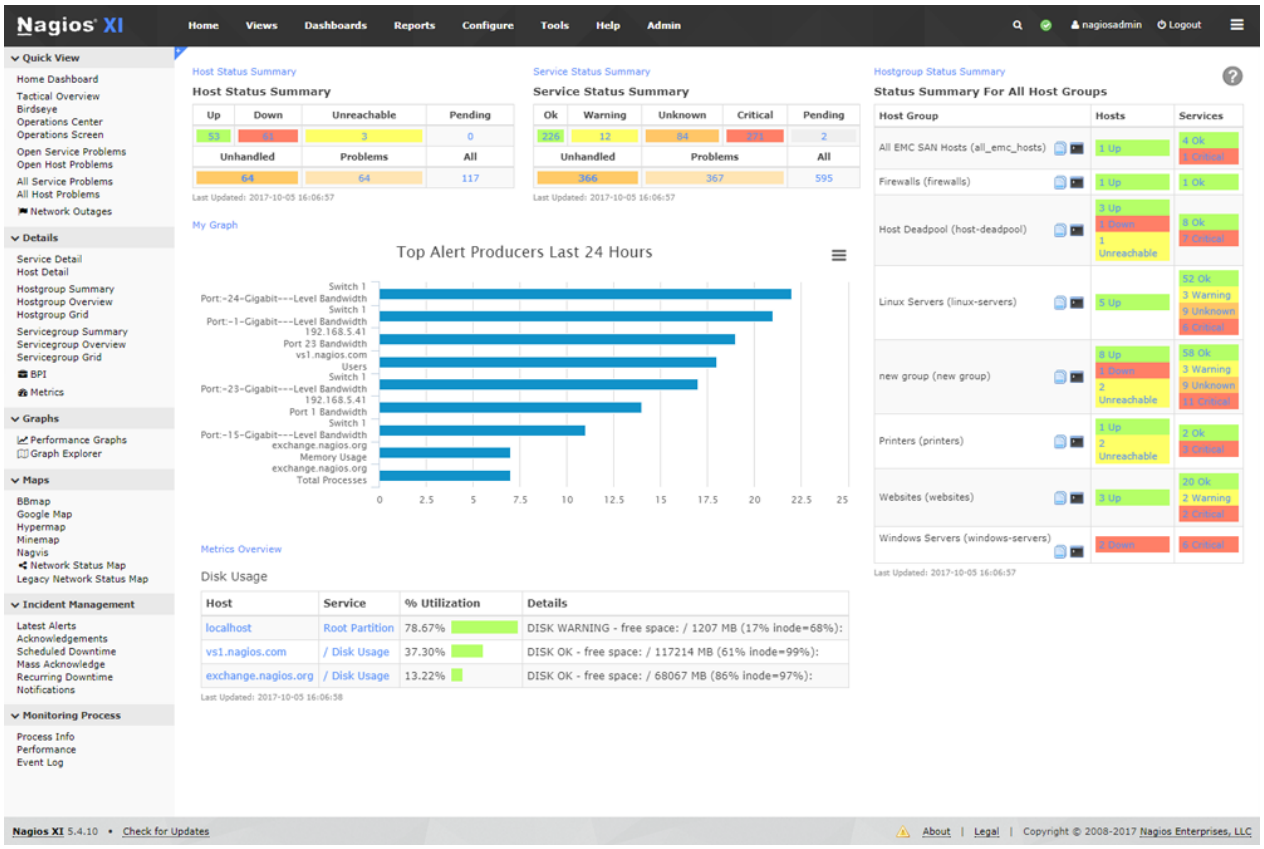

Figura 1.1: Tablero principal de Nagios XI.

scripts de monitoreo relacionados con los sistemas de recolección de datos meteorológicos. La utilización de estos *plugins abiertos* junto con un sistema central como Nagios ya ha sido propuesto anteriormente [2].

Las limitaciones de Nagios vienen a que debido a su implementación orientada a sistemas de alta disponibilidad, la configuración de los parámetros de tolerancia para la resilencia a fallas son generalmente limitados en la variedad de los mismos y los valores fijos que se pueden establecer. Además, debido a las necesidades de seguridad y accesibilidad de las estaciones meteorológicas en el contexto urbano, estas suelen instalarse en sistemas en los que se posee poco control de la red que les provee comunicación, como son las escuelas, hospitales y estaciones de policía, así como otros espacios públicos [4], dificultando aún más la capacidad

de monitoreo y disponibilidad de la red y los sistemas que soporta. Esta limitante se hace más presente debido a que los sistemas se vuelven completamente dependientes de una VPN para funcionar y para su control, debido al extenso uso de redes NAT IPv4 que predominan en los sitios de instalación.

Otra de las limitantes del sistema Nagios es que al ser un sistema de monitoreo limita la capacidad de acción de los usuarios ante un caso que lo requiera. Si bien es posible el realizar acciones por medio de scripts creados al momento de la configuración de Nagios, estos requieren una configuración extensa y compleja [5]. Además, el sistema basado en eventos impide la interacción directa de un usuario para la respuesta de forma directa desde la consola de administración, requiriendo que el usuario tenga un conocimiento del método de conexión así como la información necesaria para realizar una tarea trivial como es el reiniciar un servicio.

Actualmente, el sistema de monitoreo de calidad del aire y climatológico de la Universidad Autónoma de Ciudad Juárez, engloba varios de los retos antes descritos, ya que a pesar de la baja densidad de estaciones meteorológicas, comprende una variedad importante de las mismas. Actualmente, se compone por prototipos conectados por medio de redes virtuales privadas, monitoreados remotamente [6], estaciones de diferentes proveedores siendo monitoreadas local y externamente, así como estaciones remotas con routers dentro de la red local de la universidad (véase la Figura 1.2). Lo que provoca que sea un reto el monitorearla adecuadamente debido a la variedad de sus componentes.

Una parte vital de los sistemas de monitoreo es el registro detallado de incidentes para su posterior análisis o referencia y actualmente, no se cuenta con un registro de causas y soluciones, una base de conocimientos para los fallos de las estaciones meteorológicas existentes, así como tampoco existe un reporte automático de la calidad de los datos meteorológicos derivado de los diagnósticos y estado de salud de las estaciones meteorológicas, lo cual dificulta el uso de la información recabada por las mismas por los usuarios que accedan a la

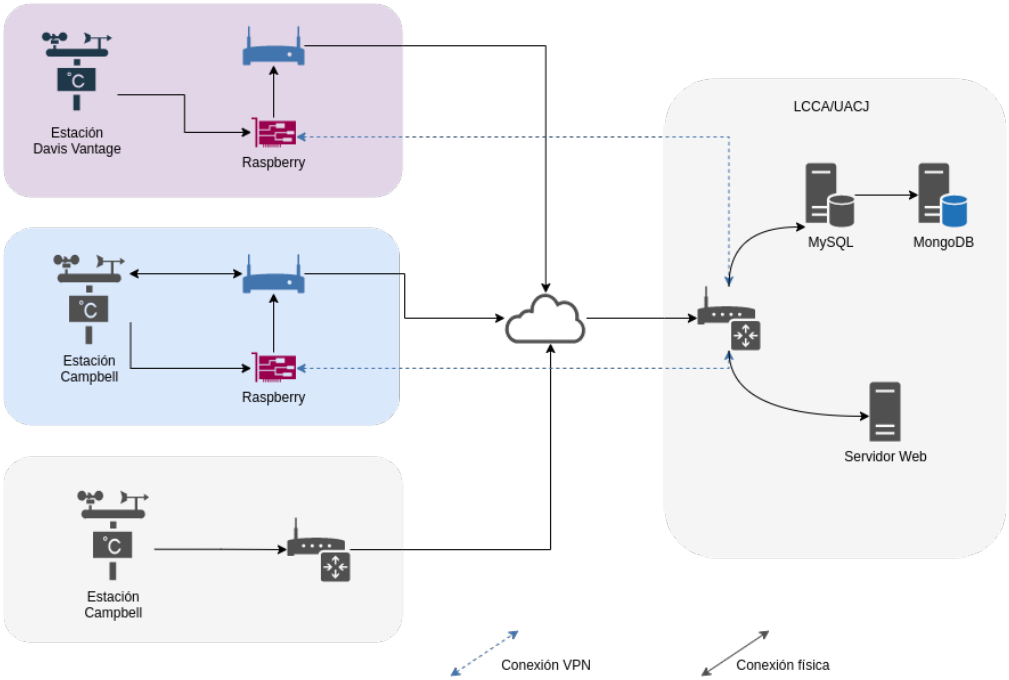

Figura 1.2: Diagrama de red de LCCA UACJ.

información que es recabada.

1.2. Definición del problema

La falta de una plataforma estandarizada para el monitoreo de estaciones meteorológicas que permita un monitoreo contínuo y control de diversos tipos de estaciones, así como la disponibilidad del acceso a los datos, y un registro de incidentes crea una dificultad creciente para los administradores de redes de monitoreo meteorológico. Con redes cada vez más densas que son cada más accesibles económicamente y menos complejas de crear, se hace notar la necesidad de un sistema estandarizado y compatible con soluciones existentes para el monitoreo de las estaciones.

1.3. Objetivo general

Desarrollar un sistema de monitoreo y control para estaciones meteorológicas que permita un monitoreo contínuo de las mismas por parte de personal no especializado, con el objetivo de proveer un mejor mantenimiento preventivo y correctivo de las estaciones meteorológicas y de calidad del aire.

1.3.1. Objetivos específicos

- Desarrollar un sistema central para coordinar y recolectar los datos del estado de las estaciones.
- Diseñar un API REST para consultar el estado de las estaciones meteorológicas.
- Diseñar e implementar una interfaz gráfica para monitorear el estatus de las estaciones.
- Integrar las diferentes estaciones meteorológicas existentes al sistema creando los controladores correspondientes.
- Integrar un sistema existente de notificaciones/alertas de terceros para fallos críticos de las estaciones.

1.4. Justificación

Creando un sistema de monitoreo eficaz para las estaciones meteorológicas se pretende el alcanzar una mayor calidad de los datos obtenidos de las mismas, así como una mejor documentación de los sistemas meteorológicos por consecuencias. Esto pretende dar el tiempo al personal especializado en enfocarse en expandir las redes existentes meteorológicas, mejorando a largo plazo la calidad y la definición de los datos recabados con la misma cantidad

de esfuerzo.

Además, haciendo el sistema de monitoreo un proyecto público y compatible con soluciones existentes, se pretende el ayudar a mejorar la calidad y confiabilidad de las redes de monitoreo meteorológico, de calidad del aire y climatológico al rededor del mundo.

1.5. Alcances y limitaciones

Entre los alcances que pretende tener el proyecto, se tienen contemplados los siguientes:

- Integrar estaciones meteorológicas existentes, que se encuentren conectadas a la red de comunicaciones del CECATEV ya sea directamente o por medio de una VPN.
- Se entregará un sistema instaldo, funcional y listo para monitorear las estaciones seleccionadas, sin necesidad de configuraciones extraordinarias.

Si bien se pretende que el proyecto tenga la flexibilidad suficiente para adaptarse a nuevos casos de uso sin necesidad de reescribir grandes partes del mismo, también se consideran las siguientes limitaciones:

- La integración se limitará a cubrir casos conocidos y recurrentes de estaciones meteorológicas existentes.
- Sólo se considera integrar una estación meteorológica de cada tipo en cada topología, dejando como proyecto futuro integrar la totalidad de la red de la universidad.
- El sistema se limitará a cubrir los casos de fallas comunes y conocidos de las estaciones.
- Sólo se monitorearán los servicios básicos de las estaciones meteorológicas que son vitales para el funcionamiento.

  Servicios tales como:

- Servicios el cliente de VPN para estaciones que así lo requieran.
- Servicio de copias de respaldo (si aplica).
- El servicio de monitoreo climático WeeWX.

Capítulo 2
Marco referencial

En este capítulo se abordarán los conceptos clave para entender las necesidades y funcionalidad del proyecto, así como las tecnologías utilizadas para el desarrollo del mismo. Esto abarca el desarrollo de un sistema de desarrollo.

2.1. Marco teórico

De acuerdo a [4], las estaciones meteorológicas son una parte fundamental de las redes meteorológicas que nos ayudan a monitorear diversas variables. Estas variables pueden ser temperatura, dirección y velocidad del viento, humedad relativa, precipitación, entre otros. Las redes meteorológicas se categorizan por tamaño, los cuales van desde redes a micro-escala que tienen un alcance de $10m^2$ hasta redes regionales, a macro escala, y globales. Cada red tiene diferentes configuraciones, métodos de recolección, y objetivos.

En el caso de la red meteorológica del CECATEV (Centro de Ciencias Atmosféricas y Tecnologías Verdes), la red es de una escala a nivel regional. Esta red se compone de diversas estaciones, las cuales están conectadas por redes privadas virtuales (VPN) para facilitar la comunicación entre las mismas, esto, debido a que las redes de comunicación existentes no suelen contar con una forma de acceder a servicios dentro de la red a la que están conectadas desde un punto externo. La red meteorológica se compone de una varidad heterogénea de

estaciones meteorológicas de diversas marcas, capacidades, sensores y métodos de conexión y recolección de datos [6].

Los sistemas informáticos que se componen de más de un componente, utilizan diversos métodos de comunicación entre ellos. Desde el accesar directamente a localizaciones de memoria física o virtual en un dispositivo para compartir información hasta crear librerías compartidas entre sistemas para accesar a la información en un depósito externo (conocidas como APIs), cada forma de acceso a los datos tiene su propio nivel de abstracción, oportunidades, y desventajas, las cuales deben ser evaluadas antes de elegir una tecnología adecuada para responder a las necesidades de cada proyecto.

Un API REST es un estándar de acceso a la información de sistemas externos por medio de protocolos *Web*, tales como HTTP/HTTPS, los cuáles permiten la consulta de datos en cualquier lenguaje que permita realizar conexiones y consultas a sitios web [7]. Entre las principales características de los API REST se encuentra que no es necesario proveer un estado previo para acceder a la información, esto implica que no es necesario mas que conocer la ruta en la que se encuentran los datos requeridos para acceder a ellos. Las ventajas que ofrece es la amplia disponibilidad de acceso a los datos y la fácil integración con sistemas existentes de manejo y procesamiento de información [8].

Figura 2.1: Diagrama del protocolo REST.

Debido a la naturaleza libre de el desarrollo web, y la poca estandarización de la comunicación entre los clientes web con los servidores, se creó la iniciativa OpenAPI a partir de un estándar existente proveído por la compañía Smartbear, Swagger. Este estándar para la comunicación con sitios por medio de APIs REST rápidamente fué ganando popularidad

gracias a la fundación Linux hasta convertirse en un estándar utilizado ampliamente por diversas organizaciones y empresas [9].

El estándar OpenAPI es un esquema de definición de estructura de datos en JSON. En este esquema se definen las rutas a las cuáles se puede acceder, los parámetros que aceptan y sus respectivos tipos de datos, así como la información que responde y los tipos de datos de los mismos. Y debido a la naturaleza abierta del esquema, este puede ser generado e integrado nativamente con el uso de diversas tecnologías de desarrollo. Permitiendo, por ejemplo, el generar las clases e interfaces correspondientes para el uso por medio de clases de los datos para lenguajes de programación fuertemente tipados [10].

2.2. Marco tecnológico

A continuación se presenta una descripción de las herramientas de tecnología que se utilizarán para el desarrollo del proyecto.

2.2.1. Docker

Docker es un sistema para la creación y distribución de imágenes de software, principalmente orientado a servidores, que permite el crear un ambiente replicable agnóstico al sistema operativo del host. Es un estándar en la industria de desarrollo de software para crear sistemas complejos manteniendo una relativa simpleza al desplegar nuevas instancias [11].

Docker utiliza un sólo Kérnel de linux para la creación de los contenedores y cada uno de los contenedores puede contener hasta n procesos, lo que lo ayuda a reducir el tamaño de sistemas complejos como se muestra en la Figura 2.2. Además de ofrecer una mayor flexibilidad y escalabilidad para tanto para realizar pruebas en máquinas de desarrollo como para

distribuir y empaquetar nuevas instancias en ambientes de producción, se ha demostrado que el costo en eficiencia al sistema que lo ejecuta es mínimo comparado con otros métodos para la administración de sistemas complejos tales como las máquinas virtuales y el empaquetado en KVM [11, 12].

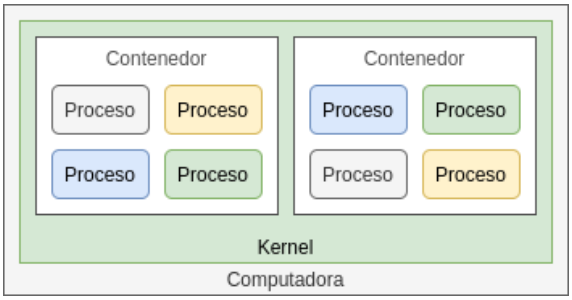

Figura 2.2: Diagrama del contenedor de procesos Docker.

2.2.2. Masonite ORM

Masonite ORM es una solución creada para python que permite la manipulación de sistemas relacionales de bases de datos creando una interfaz de código. Abstrae la complejidad de la manipulación de base de datos para convertirla en un modelo de clases con una interfaz simple para la edición de los datos. Tiene soporte nativo para transacciones, es compatible con MariaDB y está diseñado para ser incluido en proyectos complejos sin necesidad de incluir un framework completo [13].

2.2.3. FastAPI

FastAPI es un framework para desarrollo de APIs REST para Python centrado en el desarrollo rápido con ayuda de las anotaciones estándar de Python. Además de ser uno de los frameworks de desarrollo más rápidos en su ejecución, permite el crear directamente documentación compatible con el estándar OpenAPI sin necesidad de librerías externas [14].

Todo esto lo convierte en un framework ideal para extender proyectos existentes en Python y con su permisiva licencia permite

2.2.4. MariaDB

MariaDB es un motor de base de datos relacional creado por el equipo original que desarrolló MySQL, es un motor que tiene como objetivo mantenerse completamente abierto y tiene una licencia de uso permisiva para su uso en ambientes comerciales y no comerciales [15]. Tiene un rendimiento similar a MySQL en operaciones transaccionales, por lo cual es una excelente alternativa cuando se requiere un modelo de licencias permisivo [16].

2.2.5. VueJS

VueJS es una tecnología de desarrollo de sitios web con un enfoque en la reactividad de los componentes y [17].

Capítulo 3
Desarrollo del Proyecto

En este capítulo se detallará el proceso de diseño e implementación que se realizó para desarrollar el proyecto de monitoreo de estaciones meteorológicas, así como las limitaciones técnicas para el desarrollo de

3.1. Producto propuesto

Se creó un proyecto que permita monitorear el estado de los servicios de las estaciones meteorológicas, así como de la infraestructura en la que dependen, así como ofrecer un control limitado para la solución de problemas de forma remota.

El proyecto consiste de tres partes independientes:

- Un módulo para el monitoreo de el *estatus* de las estaciones, que permita el cargar la información de las estaciones de una base de datos, para luego cargar los controladores específicos de cada estación basado en la infrormación existente de las mismas para después guardar el estado en el que se encuentran en una base de datos.
- Un API REST que tendrá el objetivo de proveer un acceso sencillo a la información de los sistemas de monitoreo, así como el de proveer una interfaz de control para las estaciones que permita la ejecución remota de comandos preestablecidos, desde cualquier

punto con la autorización adecuada que haga una petición a la ruta correspondiente.

- Una interfaz gráfica, que permita el acceso a la información correspondiente de los sistemas de monitoreo, así como acceso a los reportes que se generen y permita capturar informes de solución de problemas de las estaciones para su posterior análisis.

3.2. Metodología de desarrollo

Para el desarrollo de este programa, se utilizó la estrategia de desarrollo ágil centrada en el usuario. En ella se combina la metodología de desarrollo ágil, la cuál tiene como características principales la entrega continua de resultados y la preferencia de sistemas funcionales sobre documentaciones de código robustas [18], con el diseño centrado en el usuario, el cuál tiene a los usuarios como objetivo principal para satisfacer las necesidades de requerimientos.

Debido a que la experiencia de usuario es uno de los factores que pueden separar al sistema en desarrollo de los sistemas actuales de monitoreo para equipos de cómputo, el esquema de entregas, desarrollo y planeación estarán centrado en el mismo [19]. El ciclo de entregas será con un sprint de máximo dos semanas, para una revisión de las metas, planeación y objetivos a alto nivel con el usuario y redifinir los requisitos y como sea necesario. La documentación para el usuario final, así como la documentación del API y la información técnica del sistema será un producto que será entregado al finalizar el mismo, apoyándose de la información generada en los sprints.

En el respecto del lenguaje de programación, tomando en consideración que la red de monitoreo actual utiliza WeeWX para su integración con estaciones meteorológicas [6], así como otros componentes del sistema de monitoreo existente, se pretende utilizar Python como lenguaje principal para el desarrollo del núcleo del sistema, sus módulos, y el API de consulta. Para el desarrollo de la interfaz gráfica del sitio web, se elegirá un framework ligero con Javscript. Todo esto se empaquetará en una imagen de Docker para permitir la

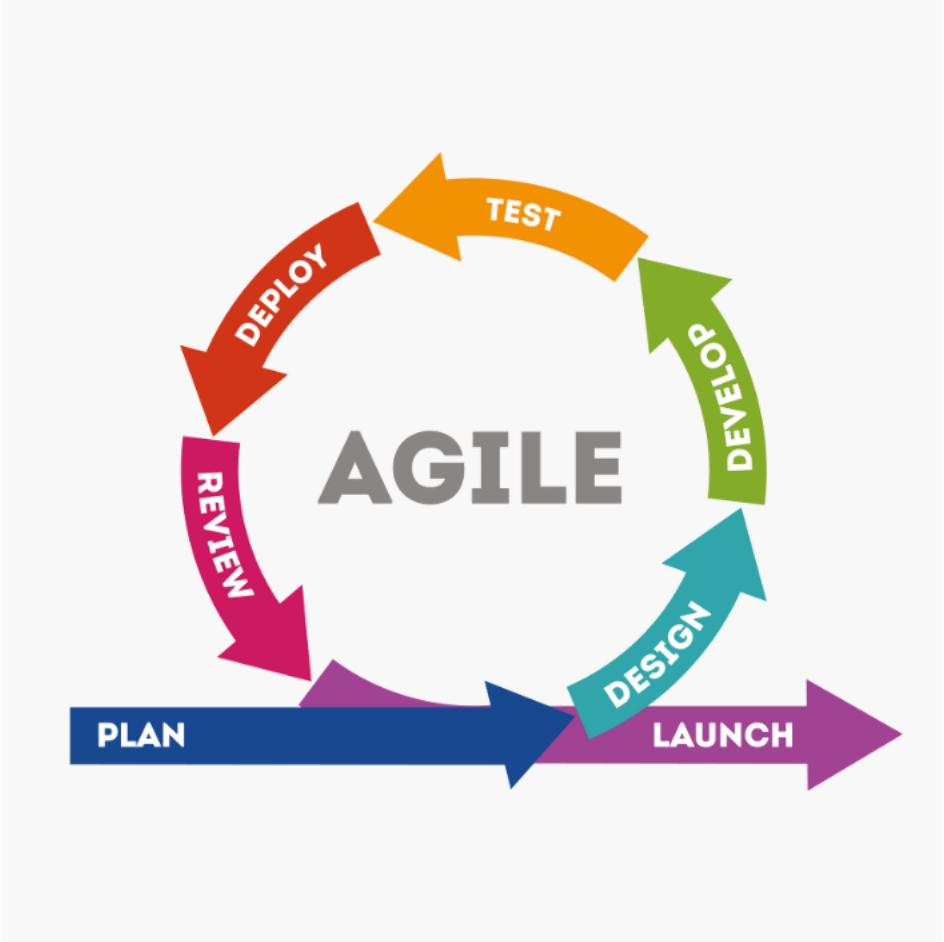

Figura 3.1: Diagrama de metodología ágil.

replicación de la instancia con el mínimo esfuerzo posible.

Este proyecto se desarrolló de acuerdo a las actividades que se muestran en la Tabla 3.1.

Tabla 3.1: Actividades a diez meses.

ACTIVIDAD	Febrero 2021	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre 2021
Revisión de la Literatura	✓	✓	✓							
Protocolo	✓	✓	✓	✓						
Selección de herramientas		✓	✓							
Diseño de la interfaz de usuario		✓		✓	✓					
Documentación de requerimientos		✓	✓	✓	✓					
Diseño de la base de datos				✓	✓					
Desarrollo del núcleo del sistema					✓	✓	✓			
Desarrollo de la interfaz de usuario						✓	✓			
Desarrollo e implementación del API REST						✓	✓	✓		
Integración con sistema de notificaciones							✓	✓		
Compilación y entrega de documentación									✓	✓
Presentación y defensa de trabajo										✓

3.3. Análisis y especificación de requisitos

Debido a la naturaleza autónoma de las estaciones meteorológicas, y a que el hecho que las mismas se encuentran sometidas a [something something] se busca crear un sistema centralizado de recolección de información que tenga gran tolerancia a las diversas condiciones adversas que se enfrentan las estaciones meteorológicas, a la vez que es lo suficientemente confiable para hacer un impacto positivo en la recolección de la información de las mismas.

3.3.1. Conexión a estaciones remotas

La conexión a las estaciones remotas se creó como un sistema modular de conexiones. Teniendo el objetivo de la extensibilidad como objetivo prioritario para el sistema de interacción con las interfaces.

Cada sistema de conexión supone sus propios retos, si bien hay diversos métodos de conexión que podrían ser útiles para la conexión a las estaciones meteorológicas, se decidió enfocarse en la conexión vía SSH a las estaciones meteorológicas que poseen una RaspberryPI como *datalogger* y como medio de interfaz que se encuentran conectadas por medio de puerto serial a las mismas. Y de las estaciones meteorológicas Campbell, que poseen diversos protocolos de comunicación pero se decidió por ukktilizar el protocolo HTTP.

Para la conexión a las estaciones RaspberryPi se considera lo siguiente:

- Actualmente cuentan con una VPN configurada para facilitar el acceso a SSH por medio de una dirección IP en el mismo segmento de red que el segmento al que se pretende el servidor final tenga.
- Ocasionalmente, las estaciones meteorológicas perderán acceso a la VPN, ya sea por fallas técnicas del servidor, del ISP, pérdidas de energía eléctrica o demás.

- Que una estación se encuentre fuera de línea de la VPN temporalmente no implica que esta no pueda operar, o incluso que no pueda contactar al servidor, tal como se observa en la 3.2

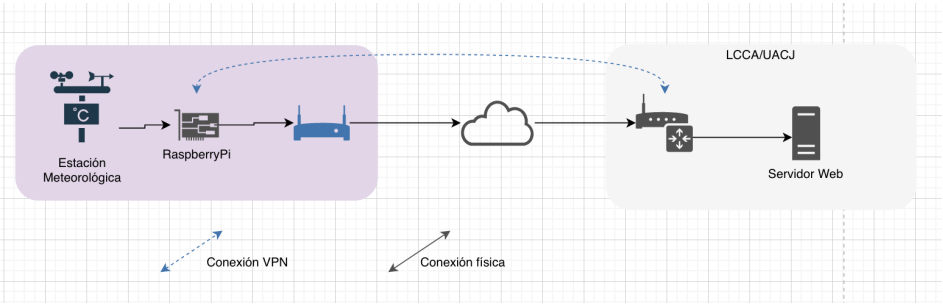

Figura 3.2: Diagrama de la redundancia de las conexiones

Consideraciones de seguridad

Debido a que generalmente no se crea una red virtual privada separada para el manejo exclusivo de estaciones meteorológicas (ya que estas suelen instalarse sobre infraestructura existente) es importante tener consideraciones de seguridad respecto a el acceso a las estaciones, debido a que pueden ser un punto de acceso a una, de otra forma, red segura e isolada.

Para realizar la conexión a las estaciones meteorológicas se requiere de acceso a la raspberrypi que funciona como puente entre ambas. Para realizar cambios, crear un servicio, y establecer la información del sistema con una mínima interacción se requiere de un usuario de alta prioridad a la máquina. En el caso del sistema operativo basado en linux que utilizan las estaciones, es el usuario con la mayor cantidad de procesos *root*.

Al considerarse comprometido el ambiente de la apliación, se consideraría comprometido el sistema completo. Ya que en este ambiente se encontrarán las contraseñas de acceso a la base de datos y la llave privada que se utiliza para hacer autenticación, si bien existen

servicios como AWS-KMS (Key Management Service), el implementar un sistema tan robusto para la administración de secretos sale del alcance de este proyecto.

Por lo tanto, se decidió crear un servicio que tome un usuario y password con acceso root"de forma temporal (o al menos uno que tenga permisos de *sudoer*) y utilizarlo para almacenar la llave pública local del servidor para realizar operaciones sin tener un par de usuario y contraseña almacenados en la base de datos que pudiera ser comprometido. De esta forma, se mitiga el impacto de una posible intrusión a la base de datos, y en caso de comprometerse, sólo es es necesario actualizar la llave privada y públicas.

3.3.2. Selección de herramientas

Para el desarrollo de el proyecto, se decidió por realizar la aplicación de servidor con el lenguaje de programación Python. Esto debido a que otros proyectos de los que depende el funcionamiento de los sistemas de el LCCA, tales como el monitoreo climatológico y meteorológico con Weewx, y el proyecto para obtener la información de las estaciones meteorológicas en diferentes estándares fueron realizados con este lenguaje. Además, el lenguaje cuenta con una librería estándar extensa así como una librería de terceras partes madura que permite el desarrollo de forma sencilla utilizando estas librerías existentes, contando con la certeza de que están listas para un proyecto de producción.

El *ORM* utilizado para el desarrollo de la aplicación, fué *MasoniteORM*, un ORM para python que tiene como objetivos principales la simpleza y extendibilidad de proyectos. Si bien, *MasoniteORM* es parte de *Masonite*, un framework para el desarrollo de aplicaciones web, este es bastante extenso y complejo, y si bien es fácilmente extensible no es simple para usarse. Por lo tanto, se decidió utilizar el framework de desarrollo de aplicaciones web *FastApi*, creado por Sebastián Ramírez (tiangolo), ya que ofrece una forma fácil de crear un *API* web, el cuál será utilizado posteriormente para el desarrollo de una interfaz fácilmente accesible para los usuarios.

La interfaz gráfica para proveer acceso a la información a los usuarios se decidió hacer en una aplicación web. Esto debido a que las interfaces web ofrecen una amplia y madura plataforma desarrollo que se puede acceder desde diferentes tipos de dispositivos, así como una gran variedad de *frameworks*, metodologías, y paradigmas, lo que ofrece una gran flexibilidad al momento de realizar un desarrollo a la medida. También está el hecho de que debido a la naturaleza del proyecto, un sistema que centraliza toda la información que es accesible por medio de un API, no parecía posible que un proyecto de una aplicación de escritorio o una aplicación web ofreciera una ventaja que no ofreciera la interfaz web.

Para el desarrollo de la interfaz web se optó por el desarrollo con los *frameworks* para desarrollo de aplicaciones web *VueJS* y *TailwindCSS*. Se seleccionó *VueJS* por la cualidad reactiva que posee, la cual permite desarrollar sistemas de monitoreo de información que requieren una constante actualización de los datos sin afectar la experiencia de usuario.

Por el motivo de hacer el proyecto lo más estándar, fácilmente acccesible y mantenible, se decidió realizar la programación y documentación del proyecto en inglés, pero manteniendo las interfaces en las que el usuario interactúa con el mismo en español. Además, se eligió el configurar *pylint* con el estándar *pep8* para el formato automático de el código del proyecto en el estándar. De la misma forma, se configuró *ESLint* en el proyecto de VueJS para el fronted, extendiendo los estándares *vue:essential* y *eslint:recommended*, para realizar el formato automático en los archivos.

3.4. Diseño

Conforme a la metodología de desarrollo centrada en el usuario utilizada en este sistema, se decidió empezar por el desarrollo de un prototipo de alta fidelidad para el diseño del sistema, para posteriormente continuar con el diseño de la infraestructura y los sistemas necesarios para entregar el proyecto, con el objetivo de crear un sistema integral que pudiera satisfacer las necesidades del LCCA al mismo tiempo que cumpliera con los requisitos técnicos necesarios para cumplir los objetivos propuestos.

3.4.1. Prototipado de la interfaz gráfica

Para el desarrollo de un prototipo rápido de la interfaz gráfica, se utilizó el software *Figma*, esto debido a que es un software gratuito, sencillo de utilizar orientado al prototipado de interfaces de alta fidelidad. *Figma* ofrece la capacidad de integrar diferentes *frameworks* de diseño de interfaces, para crear un sistema de diseño coherente, consistente a través de todo un proyecto. Utilizando este software se decidió utilizar una plantilla de un tablero de control para una plataforma genérica.

Los colores utilizados para el desarrollo de los prototipos, y posteriormente para el desarrollo de la totalidad del proyecto fueron; el color verde `#09AB5D` como color principal de la interfaz, debido a su relación con el diseño actual del sitio de consulta de la información de las estaciones meteorológicas, y el color azul `#16B2D4` como secundario por su contraste con el color verde y por su actual uso en el sitio existente del LCCA.

Utilizando esta plantilla como base para el lenguaje de diseño de la aplicación, se realizó un prototipado de la interfaz propuesta para evaluar la posible utilidad de la misma, creando un prototipo inicial de alta fidelidad. La primer interfaz en prototiparse fue el tablero de control, después de algunas iteraciones con cambios menores, el prototipo quedó tal como en la Figura 3.3, cabe notar que esta interfaz gráfica tiene elementos en la barra lateral que no

corresponden con el proyecto (tal como la sección de chat y otros similares), esto debido a que se dejaron ciertos elementos predefinidos de la plantilla original, para no romper con la estética del diseño.

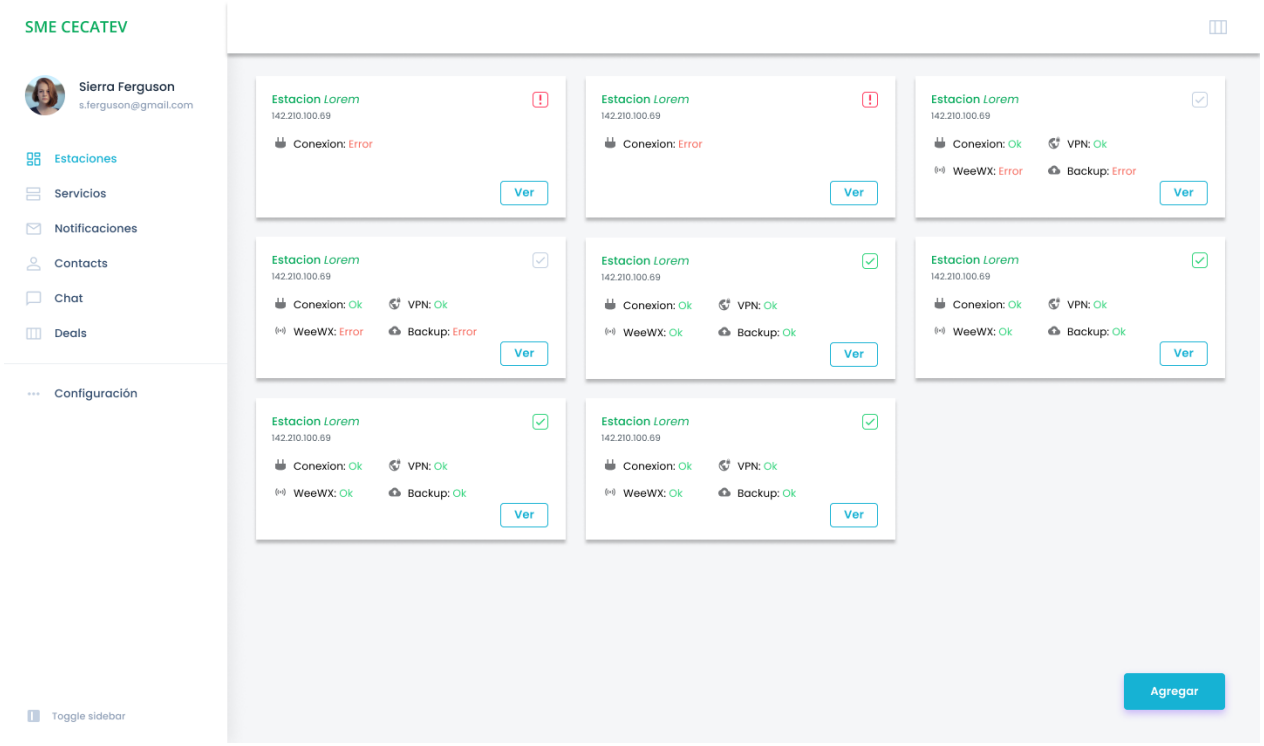

Figura 3.3: Prototipo del tablero de control.

En este prototipo de interfaz se tomaron en cuenta en varios factores, uno de ellos, es que una estación que es monitoreada puede tener un error de conexión que no permita el acceso a la misma, y que al encontrarse con un error de conexión, no es posible obtener el estado de los servicios. De ser así, la información de estos servicios no se muestra. También se tomó en cuenta el identificar las estaciones por un nombre particular, pero también tener presente la dirección IP de la misma (en caso de poseer una) para volver más sencillo el acceso técnico a la información en caso de ser necesario para un usuario técnico. Además, se hizo la consideración de tener una variedad de diferentes servicios que se podrían monitorear, y que el estado de los servicios y de las estaciones fuera independiente.

La interfaz para agregar una nueva estación al sistema contiene la información elemental que se requiere para registrar una nueva estación. En este caso, se compone de una dirección IP, un usuario, el método de conexión a la misma, el tipo de estación y el dispositivo por el que se accede a la misma. El prototipo realizado se puede observar en la Figura 3.4

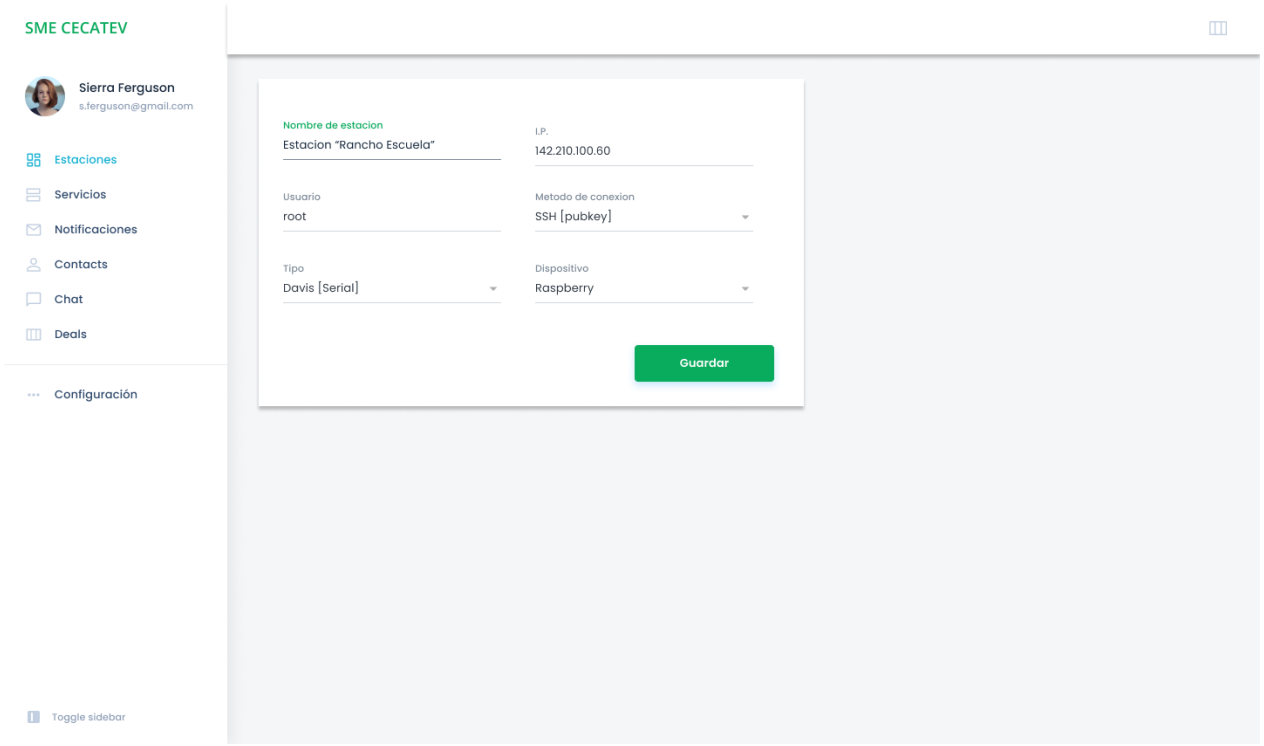

Figura 3.4: Prototipo de la interfaz para agregar una nueva estación.

Posteriormente se creó el componente de la tercera parte importante del monitoreo de las estaciones meteoreológicas, el prototipo de la interfaz de solución de problemas de las estaciones. Al ser un objetivo secundario importante el poder tener y acceder a la información que las estaciones meteorológicas generan, se considera igualmente importante el poder capturar una razón de la solución de los inconvenientes para futuros análisis. Esta información será capturada con la ayuda de una interfaz como la de la Figura 3.5.

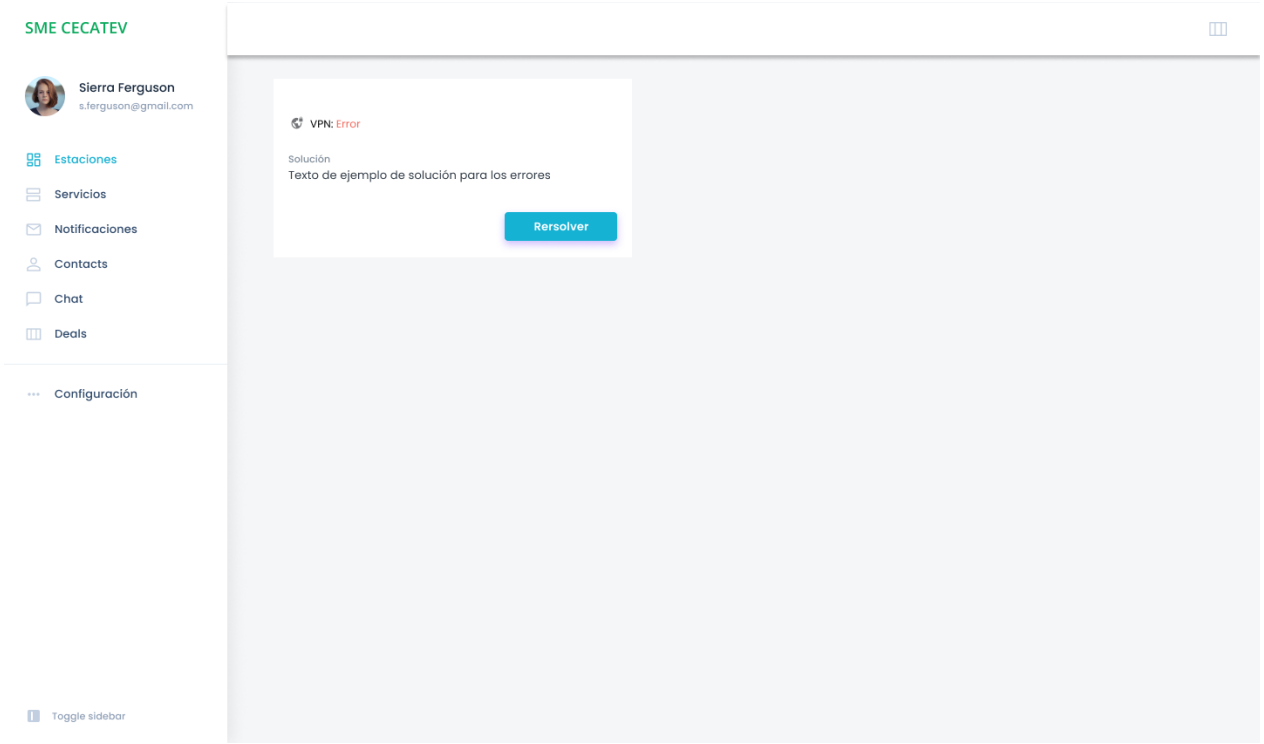

Figura 3.5: Prototipo de la interfaz de solución de errores.

3.4.2. Diseño de base de datos

Para el diseño de base de datos del sistema se consideraron dos elementos como relevantes, los datos de las estaciones meteorológicas, que incluyen datos como la conexión a las estaciones y la forma en la que se accederá a ellas, y los eventos que las mismas generen.

Si bien es posible almacenar la información de el estado de las estaciones en una *time series database*, en la que la información del estado de cada uno de los servicios y elementos que se monitorean es almacenado sin tomar en consideración el estado de los mismos, no se considera necesaria la información de los sistemas en su estado funcional, esto debido a que el volumen de datos generado, *O(n*t*s)*, donde n es el número de estaciones, t la cantidad de veces que la información se consulta por hora y s el número de servicios que se consultan, es demasiado grande en comparación a la utilidad que se podría obtener de los datos generados.

La información más relevante que se podría obtener de acuerdo a los objetivos de este proyecto y las necesidades del laboratorio, es el estado de las estaciones en caso de falla, e información detallada sobre los estados del evento con el objetivo de incrementar la calidad recabada de los datos.

Debido a esto, la base de datos se diseñó con el objetivo de almacenar la información en forma de eventos, y tomando en cuenta la utilidad futura de auditoría se decidieron agregar tiempos de creación de eventos y de la solución de los mismos. Además, para ayudar a preservar una mayor consistencia de los datos, se hizo uso de los *borrados suaves*, una metodología de manejo de la información en la que los registros físicos no son removidos, sino que son desactivados lógicamente.

En cuanto al manejo de la información adicional de las estaciones meteorológicas, es decir, la información de las mismas que no es indispensable para el funcionamiento del sistema pero es necesaria para controles internos, se agregó una tabla de atributos extra. Esta tabla contiene información diversas de las estaciones, y consiste en la forma *llave ->valor* para los

campos, esto, para asegurar la mayor flexibilidad posible de los datos y su almacenamiento. Sacrificando velocidades de indexamiento y procesamiento por una mayor libertad para extender y modificar el sistema.

Si bien habría sido posible el crear un campo extra en formato *JSON* para almacenar la información extra de las bases de datos, estos datos habrían sido enviados por defecto en cada consulta a la base de datos incrementando significativamente el tráfico a la base de datos, sin mencionar la complejidad que agrega en ORM convencionales el hacer uso de este tipo de dato.

Otra consideración que se tomó para el diseñado de la base de datos, es el crear una estructura en la que fuera posible migrar datos de una instancia a otra del proyecto, para que dado el caso que se requiriera el cambio de instancia de unos datos (por ejemplo, en el caso de un cambio de posesión del equipo) pudiera ser mantenida la información recolectada de las estaciones. Para esto, se utilizó un *uuid* como llave principal de la tabla de estaciones, lo que permite migrar la información de un sistema sin tener que realizar cambios al identificador único de las estaciones, que suele ser un dato de tipo entero sin signo autoincremental.

Con todas estas consideraciones en mente, se creó una base de datos que corresponde con los contenidos que se muestran en el modelo entidad-relación que se puede obeservar en la Figura 3.6. Para el diseño de esta base de datos, se optó por utilizar una normalización orientada al costo beneficio, sin perder la calidad de los datos obtenidos. Por ello se realizó un análisis informal con base en desiciones lógicas para la organización de la información en formas normales [20].

Las relaciones de la información después de un proceso de normalización quedaron de la siguiente forma:

- De uno a muchos, de la tabla de datos de estaciones a los campos adicionales que pudieran ser requeridos.

- De uno a muchos, de la tabla de estaciones a una tabla de eventos que pudieran presentar estas estaciones.
- De uno a muchos, de la tabla de soluciones o comentarios a la tabla de eventos.
- De uno a uno, de la tabla de eventos a la tabla de soluciones.

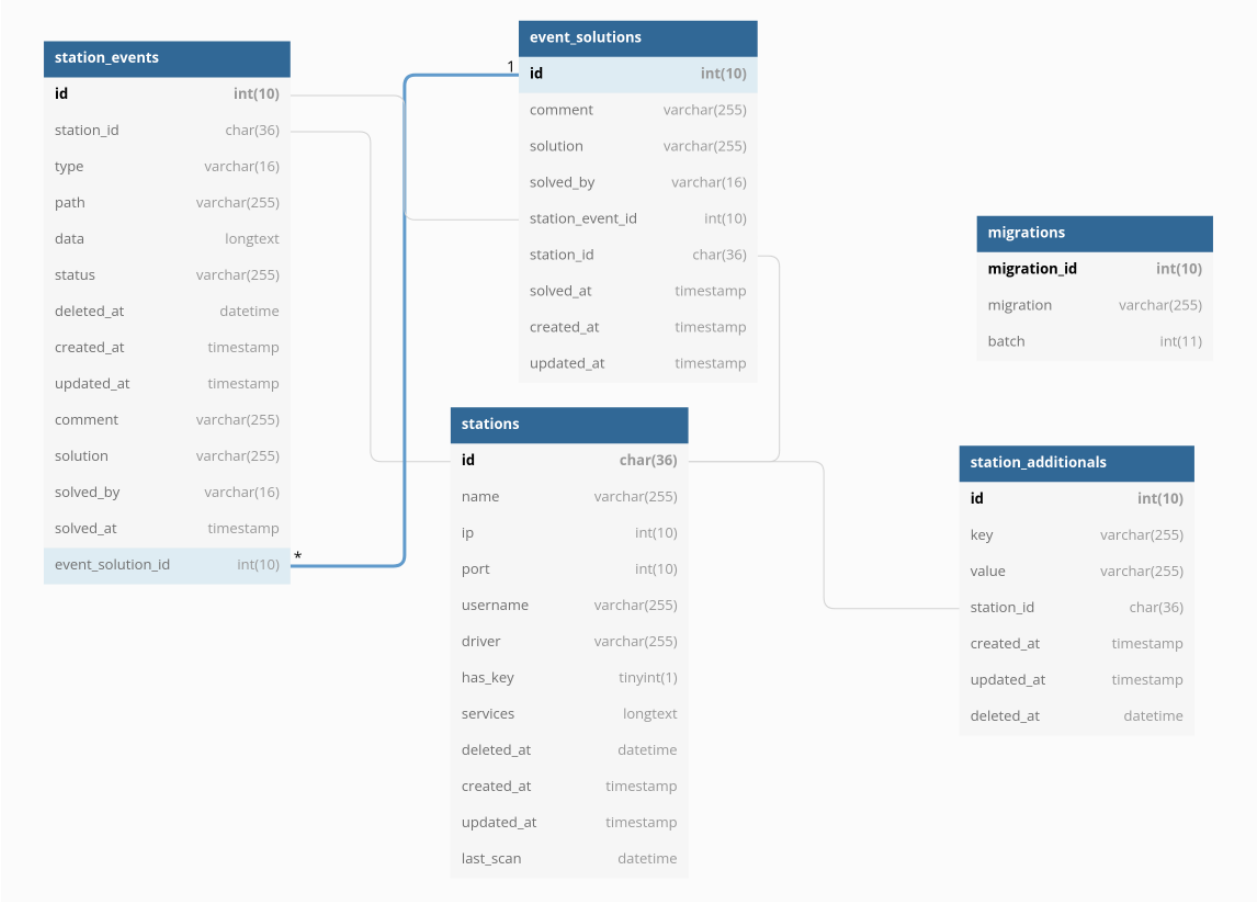

Figura 3.6: Modelo entidad-relación del proyecto meteoreo

3.4.3. Configuración de ambiente de desarrollo

El ambiente de desarrollo que se seleccionó para la sección de python del proyecto fué seleccionado con el objetivo de proveer la mayor flexibilidad y portabilidad a corto y largo plazo, haciendo fácil la modificación posterior del proyecto por los que no estuvieron involucrados inicialmente, y sencillo de replicar para futuras investigaciones. Para lograr estos objetivos, se optó por realizar el proyecto con ayuda de las tecnologías de *Docker*, debido a que permite realizar imágenes de proyectos de forma sencilla, y permite hacer contenedores multiplataforma que con una mínima configuración se vuelven útiles para el desarrollo.

Debido a la sencillez que el sistema de cnfiguración de contenedores *docker compose* ofrece, se eligió para almacenar los parámetros de configuración de los contenedores en vez de crear comandos compatibles con docker para ello. Esto permite una fácil edición de los servicios y la aplicación de los mismos de una forma estandarizada que permite una comprensión más eficaz de los parámetros y de las dependencias.

El archivo de configuración fué almacenado en la raíz del proyecto, con el nombre de `docker-compose.yml` tal como el estándar de la utilería `docker-compose` sugiere, y este archivo tiene el contenido siguiente que se muestra en el Listado 1. En este archivo, se especifica que se requiere de un servicio de *MySQL*, el cuál fué utilizado para corroborar que el desarrollo era de utilidad con el motor seleccionado para producción, así como una referencia al archivo *Docker* en el que se tiene el contenedor que se utilizará para el desarrollo. Además, se hace referencia a algunas variables, como `MYSQL_DATABASE`, que son obtenidas de un archivo `ėnv` estandarizado en la raíz del proyecto.

El archivo de *docker-compose* tiene una dependencia con un archivo de Docker, que se pretende que facilite la adición de librerías adicionales al proyecto en la posteridad. Actualmente, extiende la imagen existente de *tiangolo*, el proyecto *Uvicorn-Gunicorn-Fastapi*. Esta imagen fue utilizada como base debido a su increíble flexibilidad para el desarrollo de proyectos en

```yaml
version: '3.3'
services:
  api:
    container_name: meteoreo-api
    build:
      context: ./
      dockerfile: Dockerfile
    # ...
    # Configuración del ambiente
    # ...
    networks:
      - meteoreo-backend
  mysql:
    image: mysql
    container_name: meteoreo-mysql
    environment:
      MYSQL_DATABASE: '${MYSQL_DATABASE}'
      MYSQL_ROOT_PASSWORD: '${MYSQL_ROOT_PASSWORD}'
      MYSQL_PASSWORD: '${MYSQL_PASSWORD}'
      MYSQL_USER: '${MYSQL_USER}'
      SERVICE_TAGS: dev
      SERVICE_NAME: mysql
    ports:
      - '3306:3306'
    networks:
      - meteoreo-backend
networks:
  meteoreo-backend:
    driver: bridge
```

Listing 1: Archivo docker-compose

FastApi, sus optimizaciones automáticas para el balanceo de cargas entre diversos procesos creados automáticamente (ya que python es monoproceso) y además, por ser una imagen altamente mantenida por la comunidad, debido a su popularidad. En este archivo también se especifíca el instalar la librería `inteutils-ping` debido a que el proyecto dependerá de realizar pruebas por `ping` para revisar la conectividad con las estaciones antes de intentar realizar una conexión y la imagen base no tenía esta librería, el archivo final quedó como se muestra en el Listado 2.

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7

# Installs lib to do pings from the server
RUN apt-get update && apt-get install -y \
   inetutils-ping \
   && rm -rf /var/lib/apt/lists/*

CMD [ "/start-reload.sh" ]
```

Listing 2: Archivo Dockerfile

El editor de código seleccionado para el desarrollo del proyecto es Visual Studio Code, el cual posee una gran extensibilidad y predeterminados sensibles que permiten configurar el ambiente de trabajo de la forma que más sea conveniente para el desarrollo del proyecto, además provee la funcionalidad de *devcontainers*, los cuales son parte de una extensión que permiten el crear ambientes de desarrollo dentro de ambientes virtuales en docker, utilizando las herramientas instaladas en la imagen de docker y que no requieren de configuración adicional por parte del desarrollador para comenzar a trabajar en un proyecto. Al detectar un archivo `devcontainer.json`, esta extensión automáticamente informa al desarrollador de su existencia y le invita a iniciar su ambiente de desarrollo utilizando los parámetros definidos en el arhivo.

En este archivo se especifica un nombre para identificar el ambiente de desarrollo que sea

reconocible por el desarrollador, la localización del archivo que describe el contenedor, y una lista de extensiones para el editor de código. Entre las más importantes se encuentra *pylance* que permite realizar el formato automático de códgo con pep8 y *magicpython* una adición al editor de código que provee un motor de autocompletación para python, las demás siendo preferencias personales útiles para agilizar el desarrollo del proyecto.

```
{
  "name": "Meteoreo API",
  "service": "api",
  "remoteUser": "root",
  "shutdownAction": "stopCompose",
  "workspaceFolder": "/app",
  "dockerComposeFile": "../docker-compose.yml",
  "extensions": [
    "editorconfig.editorconfig",
    "mikestead.dotenv",
    "njpwerner.autodocstring",
    "aaron-bond.better-comments",
    "mhutchie.git-graph",
    "hookyqr.beautify",
    "magicstack.magicpython",
    "gruntfuggly.todo-tree",
    "ms-python.vscode-pylance",
    "sleistner.vscode-fileutils"
  ]
}
```

Para el desarrollo de la sección de la interfaz de web del proyecto, se instaló en la máquina de desarrollo NPM versión 14.8.1, debido a que era la última versión *LTS* (Soporte a largo plazo, por sus siglas en inglés) disponible, y el sistema de manejo de dependencias *yarn* debido a las ventajas que ofrece sobre *npm*, tales como mayor velocidad de instalación de paquetes y caché multiproyecto. No se vió como un elemento necesario el intergrar Docker o algún otro

tipo de tecnología de contenedores para el proyecto de frontend, debido a la ubicuidad de las herramientas y la simpleza de instalación y de mantenimiento de las mismas.

```
2021_08_03_052340_stations.py
2021_10_14_162126_station_additional.py
2021_11_13_020934_event_solutions.py            2022_04_03_003436_add_timestamps.py
2021_08_16_000028_station_events.py
2021_10_29_035514_stations.py
2022_03_25_013638_normalize_events_comments.py
2022_04_20_222630_events_fix.py
```

Listing 3: Archivos de migración en el proyecto.

3.5. Desarrollo

Después de haber realizado el análisis inicial de el alcance del proyecto y las necesidades de los usuarios, se comenzó con el desarrollo del proyecto. Este desarrollo se hizo en tres partes. Primero, el desarrollo de un módulo de monitoreo de las estaciones meteorológicas por medio de drivers, después un API como intermediaria entre la información almacenada en la base de datos y una interfaz gráfica para el monitoreo eficaz.

3.5.1. De la base de datos

Debido a que se seleccionó un sistema basado en un ORM para el manejo de la base de datos, esta tiene que modelarse en el sistema en forma de código para ser reconocida, de la misma forma, la creación de la estructura de la base de datos en el motor se hará por medio de migraciones creadas con el sistema de creación de base de datos, para facilitar su mantenimiento e interoperabilidad en diferentes sistemas y facilitar las integraciones con otros sistemas existentes de información.

Los archivos resultantes de estas migraciones fueron tal como se muestran en el Listado 3, las cuales se pueden encontrar en la ruta `/app/database/migrations` del proyecto, tal como es especificado en las guías de desarrollo de MasoniteORM.

```
import ipaddress
[...]
class Station(Model, UUIDPrimaryKeyMixin, SoftDeletesMixin):
[...]
def get_ip_address_attribute(self):
   return str(ipaddress.ip_address(self.ip))

def set_ip_attribute(self, attribute):
   try:
      ip = ipaddress.ip_address(attribute)
   except ValueError:
      raise ValueError("Invalid IP Address %s" % attribute)

   return int(ipaddress.ip_address(attribute))
```

Listing 4: Definición de accesor y mutador para modelo de estaciones

Si bien los modelos creados para la ejecución de este proyecto siguen los estándares de modelado de base de datos especificados en MasoniteORM, tal que al modelar la base de datos es posible obtener el mismo código que el implementado, una modificación importante al modelado de los datos es el uso de un mutador para traducir las direcciones IP a enteros y viceversa, como se muestra en el Listado 4. Este mutador es accesado con un nombre alternativo al nombre del campo en el modelo, debido a que por limitaciones del sistema no es posible utilizar mutadores y accesores con el mismo nombre del campo objetivo.

3.5.2. Del módulo de monitoreo de las estaciones

Para realizar la conexión a las estaciones meteorológicas, se decidió dividir el proyecto en dos componentes principales, un módulo de generación de reportes y un sistema de controladores que contuvieran el código de conexión y restauración de reportes de las estaciones.

Tomando como referencia el proyecto de *Monitoring Plugins* [3], el cual es compatible con diversos proyectos especializados en monitoreo de sistemas de alta resilencia, tales como *Nagios* y *Icinga*, se decidió crear un proyecto basado en drivers que pudieran ser extendibles. Cada uno de estos drivers, puede cargar una cantidad n de módulos o servicios, que contienen la información necesaria para obtener el estado de la estación meteorológica y saber si están funcionando de forma correcta o no.

Con el objetivo de crear un sistema que fuera posible integrar en diferentes ambientes y no requiriera de previa instalación de componentes extra en el dispositivo objetivo, se decidió que la información con la que se revisaría el estado de las estaciones es por medio de un comando que se ejecutaría en una estación remota, o de forma local dado el caso, y se comparara la respuesta obtenida con el resultado de la operación. En caso de que la respuesta de esta operación sea diferente a la respuesta esperada, se buscará en un arreglo de casos conocidos, que pueden ser solucionados con la ejecución de un comando.

La estructura del servicio es un objeto con la forma que se puede apreciar en el Listado 5, donde el arreglo de casos conocidos tiene el nombre de *actions*, y es anidable, lo que permite listar una serie de comandos que pueden ayudar a la solución de problemas con una produndidad n.

Para el desarrollo de la estructura del *driver* de monitoreo de las estaciones meteorológicas, se optó por crear un sistema orientado a la extensión de un componente base que fungiera como sistema principal de ejecución, verificación de credenciales y ejecución de comandos en el sistema objetivo. La estructura de los drivers para el acceso a la información quedó tal como es posible observar en la Figura 3.7.

Para los ejecutores, tal como se muestra en el diagrama de clases de la Figura 3.8

```
service = {
   "command": "", # Comando a ejecutar
   "stdout": "", # Salida esperada
   "stderr": "", # Error esperado
   "actions": {
      "read_write_enabled": {
         "response_stdout": "", # Si la respuesta del comando previo es esta
         "response_stderr": "", # Si el error del comando previo es este

         "description": "", # Descripción para el usuario del error
         "solution": "", # Solución propuesta, si existe, para el usuario

         "command": "", # Comando a ejecutar
         "stdout": "", # Salida esperada
         "stderr": "", # Error esperado
         "actions": {
            # ...
         }
      }
   }
}
```

Listing 5: Ejemplo de estructura de un servicio

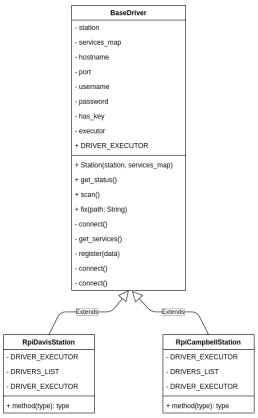

Figura 3.7: Diagrama de clase de drivers

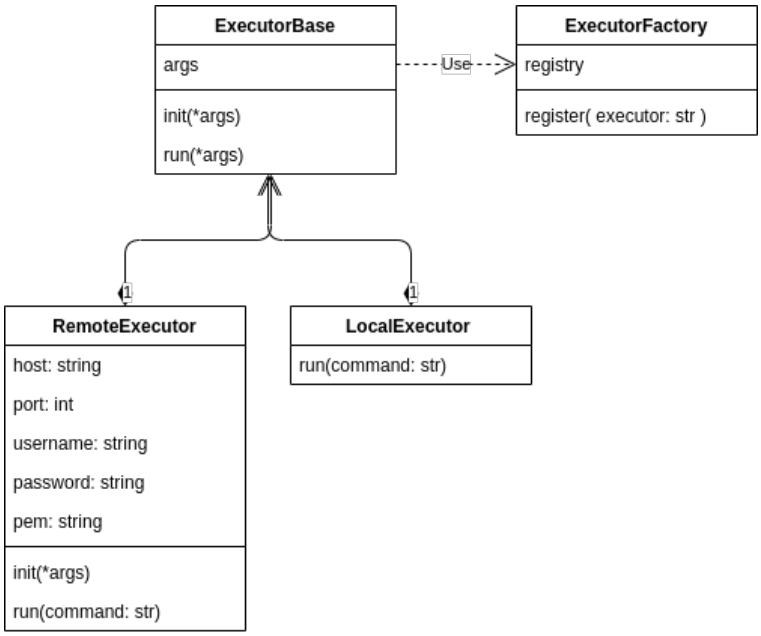

Figura 3.8: Diagrama de clase de ejecutores

3.5.3. Del módulo de monitoreo de estaciones

El módulo de monitoreo de estaciones meteorológicas tiene como objetivo el observar la información obtenida por los diversos drivers de conexión a las estaciones meteorológicas y generar reportes conforme sea necesario, la lógica de reporte es tal como se muestra en la

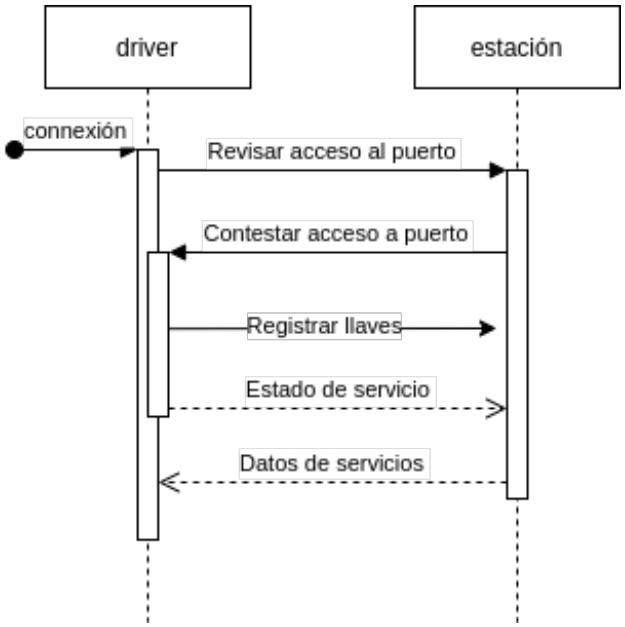

Figura 3.9: Diagrama de clase de drivers

figura 3.10.

3.5.4. Del API para el acceso a la información

3.5.5. De la interfaz gráfica del proyecto

PAra el desarrollo del proyecto, se utilizó tailwind.

3.5.6. De la documentación

Un sistema es tan bueno como su documentación.

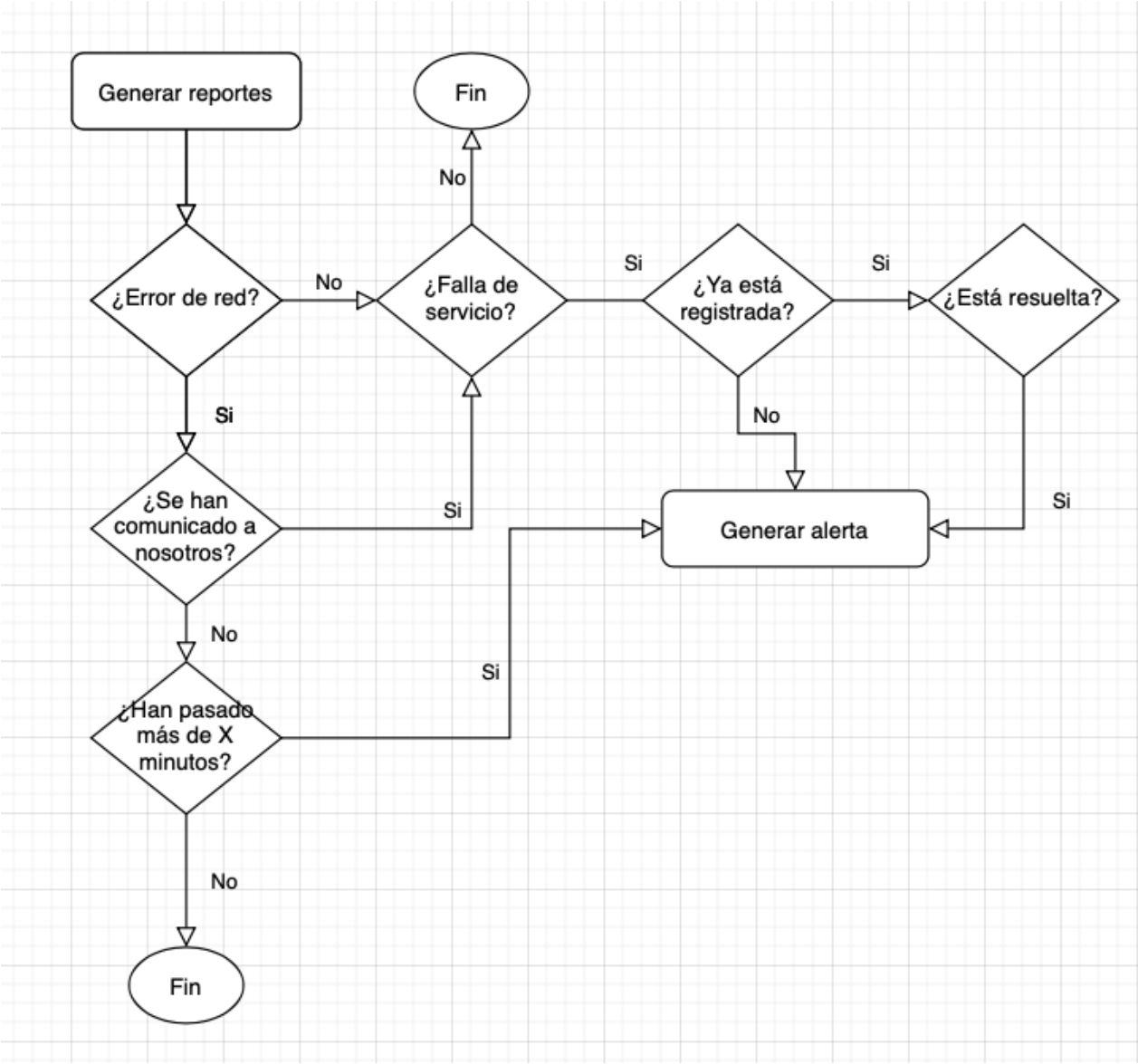

Figura 3.10: Lógica de reporte del estado de las estaciones meteorológicas

3.6. Avances

3.7. Módulo de monitoreo

Requisitos

Seguridad

Método de conexion

3.8. Selección de base de datos

Con un tiempo de respuesta de $[N]ms$, el sistema puede soprotar hasta N estaciones concurrentes.

Debido a que la recolección de los datos es por métodología pull y no push, es posible tener las estaciones en una cola que se ejecute hasta por un periodo de 5 minutos (que es un estándar en la recolección de datos de estaciones meteorológicas). Esto implica que la base de datos [X] puede soportar hasta [N x 60 x 5] datos de forma concurrente.

Tomando en cuenta las necesidades actuales del LCCA, y el estimado del tamaño de las redes de alta densidad (que pueden llegar hasta los N nodos como X artículo lo demuestra), no vale la pena el introducir la complejidad extra de un motor de base de datos desconocido y para el que no existen ORM's con soporte completo en el lenguaje de desarrollo. Porque no es un sistema de alta densidad de datos.

Si bien es posible escalar horizontalmente la infraestructura, se busca evitarlo ya que los *diminishing returns* del costo de tener que mantener un sistema de monitoreo no es costeable. Para los casos de sistemas de extremadamente alta densidad, se recomienda el crear varias

instancias seccionadas en bases de datos, o escalar la base de con un redis en vez de escalar.

3.8.1. Selección del motor de base de datos

Para el caso de uso del centro de monitoreo de estaciones meteorológicas de la UACJ, en el que la red actual cuenta con 13 estaciones, no es necesario considerar como cuello de botella el motor de base de datos que se utilizará para el sistema. Esto debido a que, con un tiempo mínimo para la consulta del estado de las estaciones de hasta 5 minutos entre consultas, el sistema podría funcionar incluso con un tiempo promedio de 23 segundos desde la consulta hasta el almacenamiento de la información. Esto, sin tomar en cuenta que es posible paralelizar el proceso de consulta y generación de eventos de las estaciones meteorológicas, por lo que no se considera como algo relevante la selección de un motor de base de datos que cuente con alto rendimiento de lectura y/o escritura de la información.

Debido a que la infraestructura del sistema de las estaciones meteorológicas ya utiliza un motor relacional de base de datos adecuado para el proyecto, MySQL, se pretende utilizarlo para este proyecto, reduciendo la carga de mantenimiento para el equipo de la universidad, además de un sistema familiar que permitirá a los involucrados realizar consultas a la información sin necesidad de aprender nuevas tecnologías.

Para esto, se utilizó la flexibilidad que ofrecen los sistemas modelado de objetos y roles (ORM, por sus siglas en inglés) [21], en la que se permite el crear sistemas agnósticos de un motor de base de datos en específico, y la creación de modelos, esquemas y relaciones de base de datos se dejan al *framework* de modelado de datos. Esto además ofrece soporte para migraciones para realizar actualizaciones de base de datos controladas en caso de requerir extender un sistema existente.

El motor de base de datos seleccionado para el desarrollo local del proyecto fué el conocido como *SQLite*, debido a la flexibilidad que ofrece al ser una base de datos que sólo depende

de un archivo para funcionar y que no requiere de instalar paquetes de software extra en la estación que se utiliza para desarrollar y probar el proyecto.

Capítulo 4
Resultados y Discusiones

[En este capítulo se presentan los resultados obtenidos correspondientes al proyecto descrito en el capítulo anterior. Los resultados se pueden presentar en tablas o gráfica y deben ser redactados y organizados de tal manera que sea fácil de comprender por los lectores.

La los resultados no se explican por si mismo, por lo que es necesario una discusión que los explique y muestre cómo ayudan a resolver el problema definido en el capítulo 1. La discusión puede mencionar someramente los resultados antes de discutirlos, pero no debe repetirlos en detalle. No prolongues la discusión citando trabajos "relacionados" o planteando explicaciones poco probables. Ambas acciones distraen al lector y lo alejan de la discusión realmente importante. La discusión puede incluir recomendaciones y sugerencias para investigaciones futuras, tales como métodos alternos que podrían dar mejores resultados, tareas que no se hicieron y que en retrospectiva debieron hacerse, y aspectos que merecen explorarse en las próximas investigaciones.]

Pruebas

Pruebas de carga

Con la ayuda de locust: https://locust.io/

Apache benchmark: https://httpd.apache.org/docs/2.4/programs/ab.html

Capítulo 5
Conclusiones

En este documento se reporta el desarrollo de un sistema de monitoreo y control para estaciones meteorológicas. A través de este apartado se presentan las conclusiones a las que se llegó durante el desarrollo del proyecto, así como las recomendaciones para trabajos futuros.

5.1. Con respecto al objetivo de la investigación

Con respecto al objetivo general del proyecto, que es el desarrollo de un sistema que permita un monitoreo contínuo de estaciones meteorológicas, para proveer un mejor mantenimiento preventivo y correctivo, se logró de manera exitosa y funcional en resolver el problema descrito en la sección 1.2, conforme a los resultados presentados en el capítulo 4.

Con la realización del proyecto se lograron los siguientes resultados:

- Se desarrolló un sistema para recolectar los datos del estado de las estaciones, que permitió un mejor control de la información y estado de las mismas.
- Se diseñó un API REST para facilitar la consulta de los datos del estado de las estaciones, documentada con OpenAPI de forma automática y,
- Se generó una interfaz gráfica para la consulta del estado de las estaciones, que permitió

una mejor visualización de los datos, integrado con un sistema de notificación de alertas.

5.2. Con respecto al futuro del proyecto

Por cuestiones de que las siguientes funcionalidades se encuentran fuera del alcance de este proyecto, se recomienda que sean implementadas posteriormente:

- El utilizar un sistema automatizado para la solución de incidentes de fallos en las estaciones meteorológicas.
- Generar reportes automatizados de calidad de datos y de líneas de tiempo de las estaciones con la información recolectada.
- El desarrollo de un sistema de predicción de tendencias de errores, que ayude a la toma de decisiones de mantenimiento preventivo y correctivo.

Se recomienda el automatizar

Tomar los componentes que fueron creados con tailwind y vue y realmente hacer uso de lasa clases de componentes e información.

5.3. Recomendaciones para futuras investigaciones

Debido a que las siguientes

Bibliografía

[1] R. Hayler, "Build your own weather station with our new guide!." `https://www.raspberrypi.org/blog/build-your-own-weather-station/`, 2018.

[2] O. Aspiazu Ituarte, *Diseño de un sistema de fertirrigación automático con control telemático y sensores*. PhD thesis, Universitat Oberta de Catalunya, Jun 2019.

[3] (2022, Marzo 17), "The monitoring plugins project." [Internet] Disponible en `https://www.monitoring-plugins.org/index.html`, 2008.

[4] C. L. Muller, L. Chapman, C. S. B. Grimmond, D. T. Young, and X. Cai, "Sensors and the city: a review of urban meteorological networks," *International Journal of Climatology*, vol. 33, no. 7, pp. 1585–1600, 2013.

[5] Nagios, "Guide to configure the restarting of a service in nagios." `https://assets.nagios.com/downloads/nagiosxi/docs/Restarting-Linux-Services-With-NRPE.pdf`, 2018.

[6] F. A. Vázquez-Galvez, F. Estrada-Saldaña, and J. I. Hernández-Hernández, "Red climatológica y de calidad del aire uacj," tech. rep., Universidad Autónoma de Ciudad Juárez, 2019.

[7] M. Massé, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'reilly, 1 ed., 2012.

[8] H. Ed-douibi, J. L. Cánovas Izquierdo, and J. Cabot, "Example-driven web api specification discovery," in *Modelling Foundations and Applications* (A. Anjorin and H. Espinoza, eds.), (Cham), pp. 267–284, Springer International Publishing, 2017.

[9] O. Organization, "Openapi faq." `https://openapis.org/faq`.

[10] W. Cheng, "Openapi generator · generate clients, servers, and documentation from openapi 2.0/3.x documents," Apr 2021.

[11] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[12] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pp. 171–172, IEEE, 2015.

[13] Masonite, "Introduction." `https://orm.masoniteproject.com/`, 2021. [Online; accessed 19-July-2008].

[14] S. Ramírez, "Fastapi." `https://fastapi.tiangolo.com/`, Dec 2020.

[15] M. foundation. `https://mariadb.org/`, 2019.

[16] S. Tongkaw and A. Tongkaw, "A comparison of database performance of mariadb and mysql with oltp workload," in *2016 IEEE Conference on Open Systems (ICOS)*, pp. 117–119, 2016.

[17] C. Macrae, *Vue JS Up and Running*. O'Reilly Media, Incorporated, 2018.

[18] M. Fowler, J. Highsmith, *et al.*, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[19] Z. Hussain, M. Lechner, H. Milchrahm, S. Shahzad, W. Slany, M. Umgeher, and P. Wolkerstorfer, "Agile user-centered design applied to a mobile multimedia streaming application," *Lecture Notes in Computer Science*, p. 313–330, 2008.

[20] H. Lee, "Justifying database normalization: a cost/benefit model," *Information processing & management*, vol. 31, no. 1, pp. 59–67, 1995.

[21] T. "Halpin, *Object-Role Modeling (ORM/NIAM)*, pp. 81–103. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

Apéndice A
Nombre del Apéndice

[Sustituye este texto. En esta sección opcional se deberá incluir información secundaria o material importante que es muy extenso. El apéndice se coloca después de la literatura citada. Ejemplos de información que puede colocarse en el apéndice: una lista de universidades visitadas; los datos obtenidos de todas las repeticiones del experimento; derivaciones matemáticas extensas; todos los resultados del análisis estadístico (incluyendo quizás los no significativos) y mapas de distribución para cada fenómeno estudiado; listados completos de código fuente; etc.]

El hash del commit en Github que se entregó como producto es [HASH]