ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA EL MONITOREO DE DISPOSITIVOS EN UN WISP (*WIRELESS INTERNET SERVICE PROVIDER*)

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

NÉSTOR EDUARDO CANCHIGÑA CATAGÑA

DIRECTOR: XAVIER ALEXANDER CALDERÓN HINOJOSA, M. Sc

Quito, diciembre 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Néstor Eduardo Canchigña Catagña, bajo mi supervisión.

XAVIER ALEXANDER CALDERÓN HINOJOSA, M. Sc

DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Néstor Eduardo Canchigña Catagña, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejamos constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

NÉSTOR EDUARDO CANCHIGÑA CATAGÑA

DEDICATORIA

A mis padres y a mi hermana.

Néstor Canchigña

AGRADECIMIENTO

A mis padres Néstor y Carolina, y a mi hermana Estefany, por ser la principal fuente de motivación y apoyo incondicional para lograr este objetivo.

Al M.Sc Xavier Calderón por su tiempo y dedicación al dirigir el presente proyecto.

A la Ph.D Cecilia Paredes por haber sido tutora de carrera.

A los docentes que tuve durante la estancia en la EPN.

A las amistades que de una u otra manera formaron parte de esta etapa de la vida.

Néstor Canchigña

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El objetivo del presente Trabajo de Titulación es desarrollar un prototipo de aplicación web para el monitoreo de dispositivos en un WISP (*Wireless Internet Service Provider*), considerando que algunas microempresas que ofrecen el servicio de Internet realizan el monitoreo de sus equipos de forma descentralizada y manual, por lo cual, el prototipo se convierte en una herramienta de apoyo, ya que, genera alertas según la información obtenida del monitoreo de la red.

El prototipo es implementado en dos partes, la primera consiste en elaborar la aplicación *backend* mediante el *framework* NestJS y la segunda se basa en crear la aplicación *frontend* con el *framework* Angular, además, se utiliza TypeORM para el manejo de la base de datos. Por otra parte, se emplea la metodología ágil Kanban para desarrollar la aplicación.

El primer capítulo presenta los objetivos, el alcance y el marco teórico que expone los conceptos fundamentales para el desarrollo del prototipo.

El segundo capítulo presenta los requerimientos funcionales y no funcionales, también, el diseño del prototipo constituido por la arquitectura, diagrama entidad-relación, diagrama relacional, diagrama de actividades y diagrama de clases, además, se muestra la implementación y despliegue de la aplicación.

El tercer capítulo presenta las pruebas de funcionamiento del prototipo, los resultados de las encuestas de satisfacción y la corrección de errores.

El cuarto capítulo presenta las conclusiones y recomendaciones.

Finalmente, en los anexos se adjuntan los diagramas y el código de la aplicación, además, el *script* para generar los *triggers* en la base de datos.

PALABRAS CLAVE: NestJS, Angular, TypeORM, SNMP

ABSTRACT

The objective of this Degree Project is to develop a prototype of a web application for the monitoring of devices in a WISP (Wireless Internet Service Provider), considering that some micro-companies that offer Internet service carry out the monitoring of their equipment in a decentralized and manual way. Therefore, the prototype becomes a support tool, since it generates alerts based on the information obtained from monitoring the network.

The prototype is implemented in two parts, the first consists of developing the backend application with the NestJS framework and the second is based on creating the frontend application with the Angular framework, in addition, TypeORM is used to manage the database. On the other hand, the agile Kanban methodology is used to develop the application.

The first chapter presents the objectives, the scope and the theoretical framework that exposes the fundamental concepts for the development of the prototype.

The second chapter presents the functional and non-functional requirements, also, the design of the prototype constituted by the architecture, entity-relationship diagram, relational diagram, activity diagram and class diagram, in addition, the implementation and deployment of the application is shown.

The third chapter presents the functional tests of the prototype, the results of the satisfaction surveys and the correction of errors.

The fourth chapter presents the conclusions and recommendations.

Finally, the annexes include the diagrams and the application code, as well as the script to generate the triggers in the database.

KEYWORDS: NestJS, Angular, TypeORM, SNMP

1. INTRODUCIÓN

La pandemia que se vive actualmente ha provocado que los problemas de las microempresas se agudicen, por ejemplo, los clientes que experimentan interrupciones del servicio de Internet durante el teletrabajo o en clases *online*, han optado por cambiarse a otra microempresa o PYME (Pequeña Y Mediana Empresa).

En algunas microempresas que proveen el servicio de Internet se realiza el monitoreo de sus equipos de forma descentralizada y manual, por consiguiente, en el caso de que un cliente experimente intermitencias en la navegación por Internet, al administrador de la red se le complica encontrar estos problemas. De forma similar, al escalar la red de la microempresa el problema de la monitorización se sigue ampliando, por lo que, se obstaculiza hallar inconvenientes en los equipos.

En consecuencia, en el presente Trabajo de Titulación se plantea el desarrollo de un prototipo de aplicación web para el monitoreo de dispositivos en un WISP (*Wireless Internet Service Provider*) con el propósito de que genere alertas o notificaciones, según la información obtenida del monitoreo de la red, de forma similar, será una herramienta de apoyo para el administrador de red.

1.1. OBJETIVOS

El objetivo general de este Proyecto Técnico es desarrollar un prototipo de aplicación web para el monitoreo de dispositivos en un WISP (*Wireless Internet Service Provider*).

Los objetivos específicos de este Proyecto Técnico son:

- Analizar los conceptos fundamentales para el desarrollo del prototipo.
- Diseñar la aplicación web en base a los requerimientos del administrador de red.
- Implementar los módulos y componentes del prototipo.
- Realizar pruebas de funcionalidad al prototipo.

1.2. ALCANCE

Para desarrollar el prototipo de aplicación web se emplean los *frameworks* NestJS [1] y Angular [2], el primero sirve para construir la aplicación en el lado del servidor (*backend*) y el segundo para crear el *dashboard web* (*frontend*). Para el monitoreo de los parámetros y consultas a los dispositivos (*Access Points y Stations*) se usa el módulo net-snmp [3]. La información obtenida de los equipos es almacenada en una base de datos relacional. El desarrollo del proyecto está basado en la metodología ágil Kanban.

Se utiliza como base la información de una microempresa que provee el servicio de Internet, por lo cual, los equipos de esta red trabajan con el sistema operativo AirOS, así que, el documento [4] manifiesta que posee un agente SNMP que ayuda a obtener la información del equipo, además, cuenta con SYSLOG que se aprovecha para las notificaciones de caídas de enlaces.

La notificación de caídas de los enlaces de los APs (*Access Point*) se lo hace mediante un correo electrónico al administrador de la red y una actualización de los datos en el *dashboard web,* de modo similar, para caídas de enlaces de clientes solo se actualiza la información del *dashboard web*. Los equipos no generan *Traps,* en consecuencia, el servidor recepta los *logs* generados por los equipos y realiza un filtrado para solo interpretar los mensajes *logs* que especifican que el dispositivo está conectado o se desconectó. El prototipo genera Pings con la ayuda de [5] para constatar las caídas de APs (*Access Point*). Además, se crean reportes de las caídas de enlaces. Por otra parte, se puede configurar (editar) el tiempo de monitoreo y el límite que deben tener los parámetros a monitorear.

El prototipo cuenta con tres roles de usuario: Administrador, Analista y Usuario. El Administrador viene precargado en la base de datos, el Analista y el Usuario debe ser ingresado por el Administrador. En consecuencia, para que los usuarios interactúen con el *dashboard web* deben ingresar sus credenciales para ser validados.

Los roles de usuario realizan lo siguiente:

- **Administrador:** es el rol con mayor privilegio en la aplicación, tiene la responsabilidad de leer, crear, editar o eliminar usuarios con rol de Analista o Usuario, asimismo, podrá leer, crear, actualizar o eliminar dispositivos (*Access Points* y *Stations*). De igual importancia, editar el tiempo de monitoreo y el límite de los parámetros a monitorear (los parámetros a monitorear se los definirá en las entrevistas), y visualizar las alarmas registradas y los reportes.
- **Analista:** tiene la responsabilidad de leer, crear, actualizar o eliminar dispositivos (*Access Points* y *Stations*) y visualizar las alarmas registradas y los reportes. De la misma forma, editar el tiempo de monitoreo y el límite de los parámetros a monitorear, y visualizar las alarmas registradas y los reportes.
- **Usuario:** es el rol con menor privilegio podrá visualizar las alarmas registradas y los reportes.

Esquema de la red

En la Figura 1.1 se presenta el esquema de la red que está dividido en tres secciones de la siguiente manera:

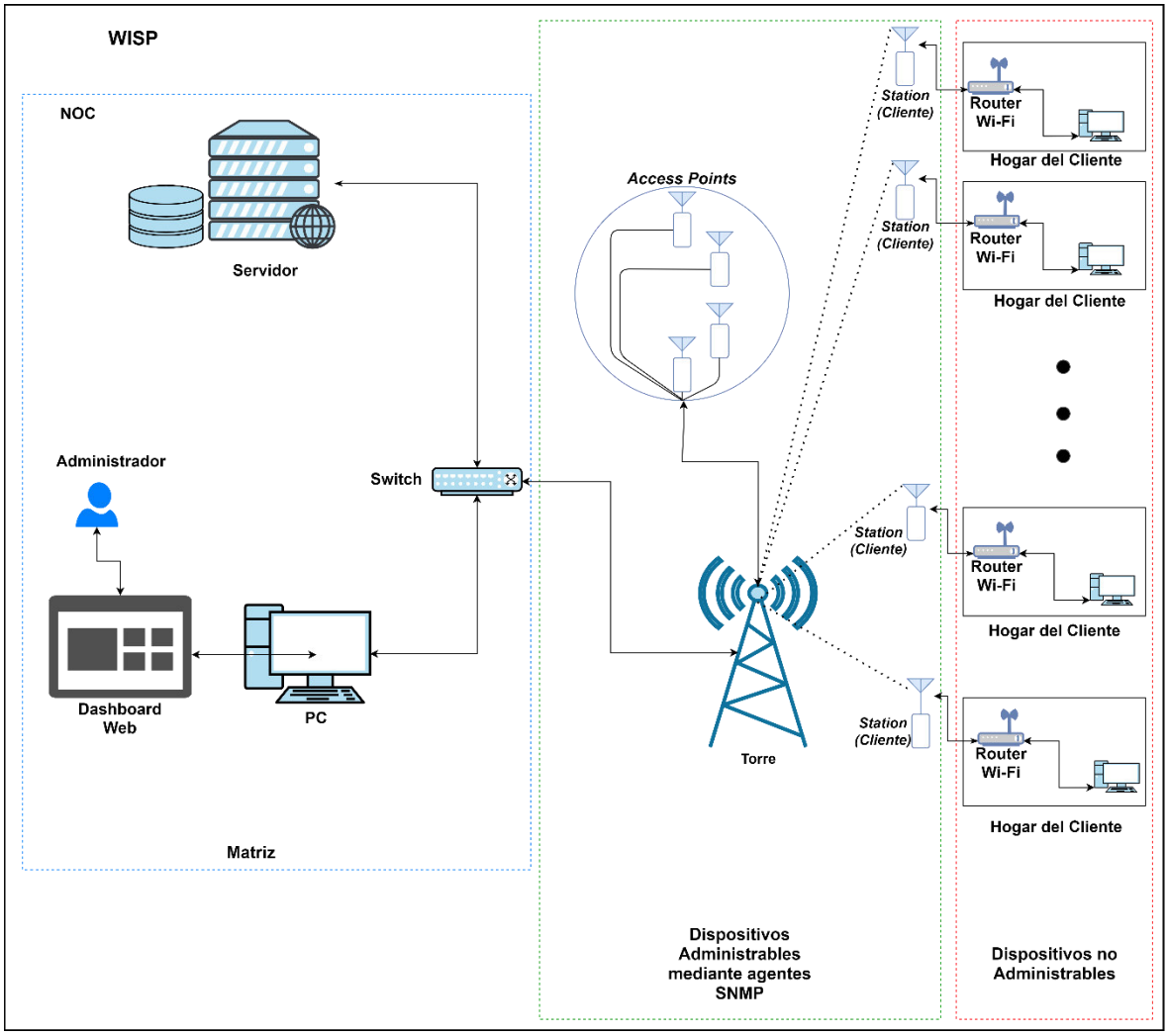

Figura 1.1. Esquema de la red

- **Matriz:** lugar donde se encuentran las oficinas de la microempresa, en el NOC está el servidor web, además, las personas que poseen credenciales pueden ingresar a interactuar con el *dashboard web.*
- **Dispositivos administrables mediante agentes SNMP (*Simple Network Management Protocol*):** lugar donde se encuentran los dispositivos que son monitoreados. Por cuestiones de políticas de seguridad y privacidad las direcciones MACs (*Media Access Control*), IPs (*Internet Protocol*), URLs (*Uniform Resource Locator*) y nombres de clientes son censuradas. Hay que resaltar que los *Access Points* y *Stations* son para ambientes *outdoor,* por tal razón, los *Access Points* se

encuentran instalados en la Torre apuntando a diferentes sitios para que los clientes que tengan línea de vista puedan engancharse mediante un equipo que trabaja como *Station* (Cliente). El equipo *Station* Cliente se lo instala en la parte externa del hogar del cliente.

- **Dispositivos no Administrables:** lugar donde se encuentran los dispositivos (celulares, computadoras, *iPads*, *tables*, etc.) personales de los clientes, estos equipos no son monitoreados, de forma similar, el *router* Wi-Fi que se instala dentro del hogar del cliente tampoco va a ser monitoreado.

1.3. MARCO TEÓRICO

1.3.1. SNMP (*SIMPLE NETWORK MANAGEMENT PROTOCOL*)

En la Figura 1.2 se expone el esquema de la función y características del protocolo SNMP (*Simple Network Management Protocol*) [6].

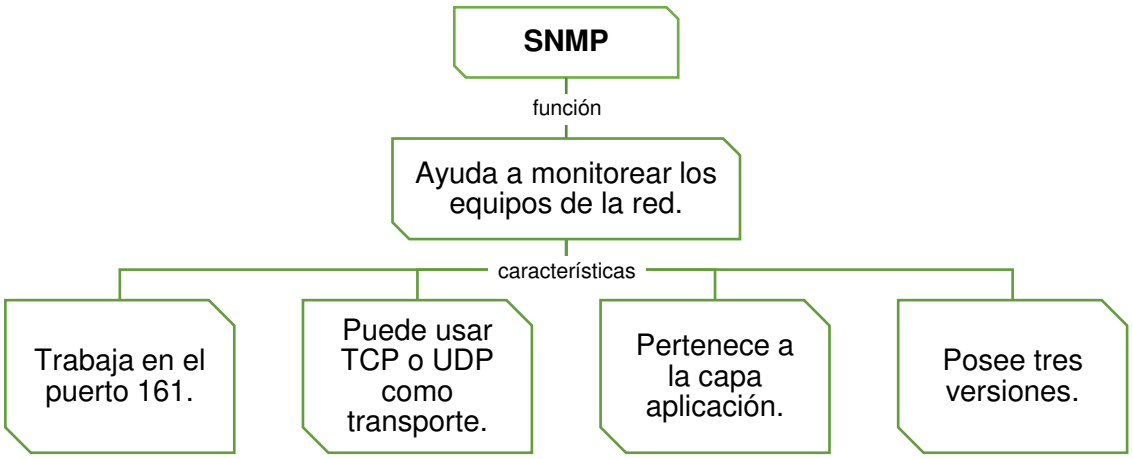

Figura 1.2. SNMP (*Simple Network Management Protocol*)

En la Figura 1.3 se presenta los elementos que conforman el procotolo SNMP (*Simple Network Management Protocol*) [6]:

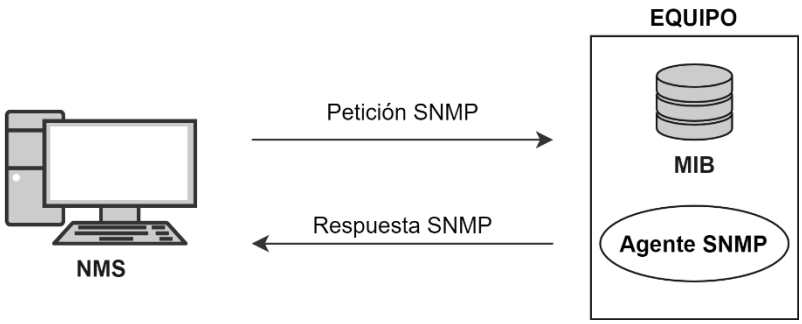

Figura 1.3. Elementos SNMP (*Simple Network Management Protocol*)

- **NMS (*Netwok Management System*):** encargado de gestionar los equipos.
- **Agente SNMP (*Simple Network Management Protocol*):** es un servicio que se ejecuta en el equipo para gestionar las peticiones SNMP (*Simple Network Management Protocol*).
- **MIB (*Management Information Base*):** especifica la estructura de la información del equipo.

En la Tabla 1.1 se detalla las columnas Versión, Autenticación y Cifrado que posee el protocolo SNMP [6].

Tabla 1.1. Versiones SNMP (*Simple Network Management Protocol*)

Versión	Autenticación	Cifrado
SNMPv1	Comunidad	Ninguno
SNMPv2c	Comunidad	Ninguno
SNMPv3	Usuario	Ninguno
	Usuario con MD5 o SHA	Ninguno
	Usuario con MD5 o SHA	AES o DES

1.3.2. SYSLOG

En la Figura 1.4 se expone el esquema de la función y características del protocolo Syslog.

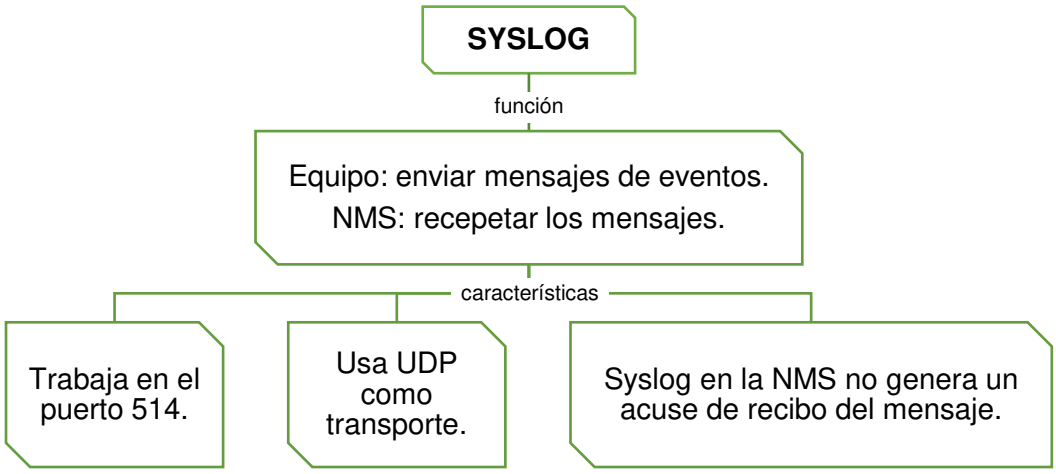

Figura 1.4. Syslog [7]

En la Figura 1.5 se presenta los elementos del protocolo Syslog, de modo que, el EQUIPO envía mensajes sobre eventos que ocurren en su sistema y la NMS (*Netwok Management System*) es la encarga de recolectar los mensajes que genera el EQUIPO.

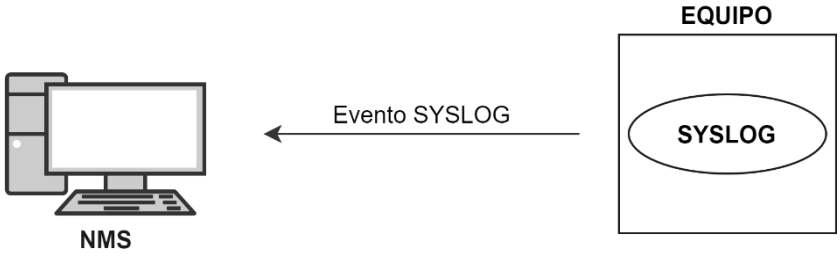

Figura 1.5. Elementos Syslog

1.3.3. REST (*REPRESENTATIONAL STATE TRANSFER*)

REST (*REpresentational State Transfer*) es una arquitectura que permite diseñar aplicaciones de servicios web, además, aprovecha los métodos HTTP (*HiperText Transfer Protocol*) para realizar el CRUD (*Create, Read, Update and Delete*), de modo que, para crear un recurso emplea POST, para leer un recurso usa GET, para actualizar un recurso aplica PUT y para eliminar un recurso utiliza DELETE [8]. En la Tabla 1.2 se presenta los códigos de estado más comunes de respuestas HTTP (*HiperText Transfer Protocol*) [9].

Tabla 1.2. Códigos de estado de respuestas HTTP (*HiperText Transfer Protocol*)

Código	Definición
200 OK	La petición ha tenido éxito.
201 *Created*	La petición ha tenido éxito y se ha creado un nuevo recurso.
400 *Bad Request*	La petición ha fallado debido a una sintaxis inválida.
403 *Forbidden*	El Usuario no tiene los permisos para acceder al recurso.
404 *Not Found*	El servidor no encontró el recurso solicitado.
500 *Internal Server Error*	El servidor ha detectado un error.

En la Figura 1.6 se presenta un ejemplo de un Usuario que requiere obtener un recurso del Servidor, por lo que, la Computadora envía una petición HTTP GET al Servidor y como respuesta obtiene un HTTP 403 que significa que el Usuario no tiene permisos para acceder al recurso solicitado.

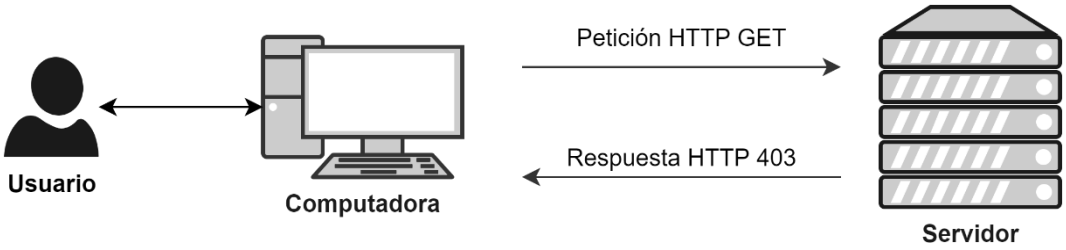

Figura 1.6. Petición HTTP (*HiperText Transfer Protocol*) GET

1.3.4. SSE (*SERVER SENT EVENTS*)

El SSE (*Server Sent Events*) es una tecnología que permite al Servidor enviar información actualizada al Cliente, se debe agregar que la comunicación que se crea entre el Servidor y el Cliente es unidireccional y persistente [10] [11]. En la Figura 1.7 se presenta el proceso de comunicación entre el Cliente y el Servidor SSE (*Server Sent Events*), por lo cual, el Cliente envía una petición HTTP al Servidor SSE para subscribirse a la escucha de eventos.

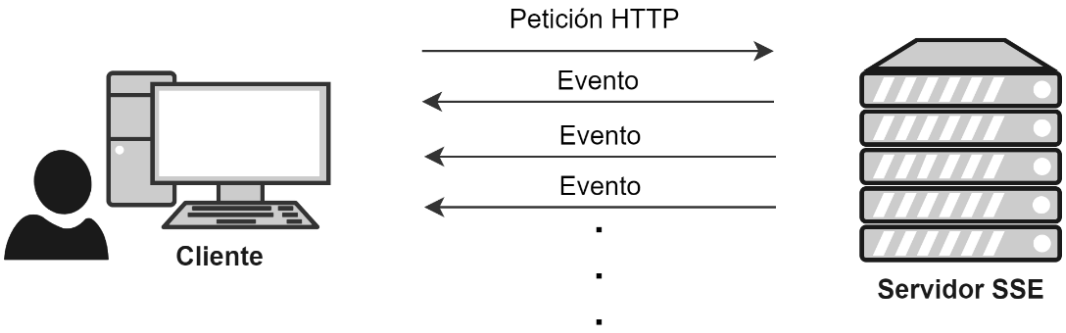

Figura 1.7. SSE (*Server Sent Events*)

1.3.5. HERRAMIENTAS DE DESARROLLO

1.3.5.1. VSC (*Visual Studio Code*)

Visual Studio Code es un editor de código fuente ligero pero potente que está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js [12]. En la Figura 1.8 se presenta una captura de VSC (*Visual Studio Code*), de modo que, se puede observar un fragmento de código escrito en el lenguaje de programación TypeScript.

Figura 1.8. Captura VSC (*Visual Studio Code*)

1.3.5.2. NodeJS

Node.js es un entorno de ejecución de JavaScript de código abierto y multiplataforma, además, ejecuta el motor JavaScript V8 de Google fuera del navegador [13]. Por otro lado, presenta una gran ventaja para los desarrolladores que usan JavaScript, ya qué, pueden crear aplicaciones *backend* y *frontend* empleando un solo lenguaje de programación [14].

1.3.5.3. NestJS

NestJS es un *framework* que permite crear aplicaciones Node.js *backend* escalables, está construido y es compatible con el lenguaje de programación TypeScript [15].

Para usar NestJS se debe instalar el CLI (*Command Line Interface*) de NestJS con el comando: `npm i -g @nestjs/cli`.

En la Figura 1.9 se presenta la descripción general de NestJS. A continuación, se detalla cada elemento:

- ***Frontend***: lado del Cliente.
- ***Backend***: lado del Servidor.
- ***Module***: es una clase que NestJS utiliza para organizar la estructura de la aplicación [16].
- ***Controller***: es una clase que se encarga de manejar las solicitudes y devolver las respuestas al Cliente [17].
- ***Provider***: permite que los objetos puedan crear varias relaciones entre sí [18].
- ***Service***: es una clase que suministra servicios al *Controller*, por ejemplo, almacenar y recuperar datos [19].

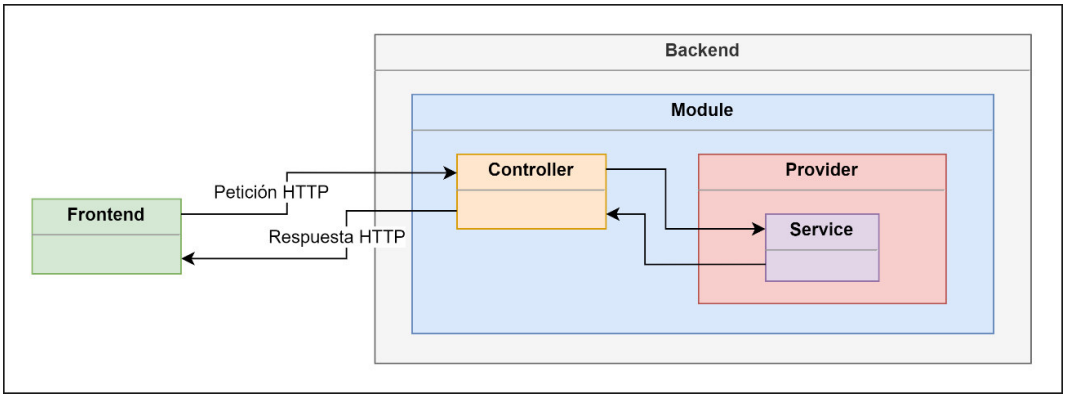

Figura 1.9. Descripción general de NestJS

1.3.5.4. TypeORM

TypeORM es un ORM (*Object Relational Mapping*) que se puede ejecutar en NodeJS, se puede utilizar con TypeScript y JavaScript. Su objetivo es ayudar a desarrollar cualquier tipo de aplicación que utilice bases de datos [20].

En la Figura 1.10 se puede apreciar que TyperORM es un intérprete entre dos mundos diferentes de programación, por ejemplo, la Clase codificada en TypeScript y la Base de datos implementada en SQL (*Structured Query Language*).

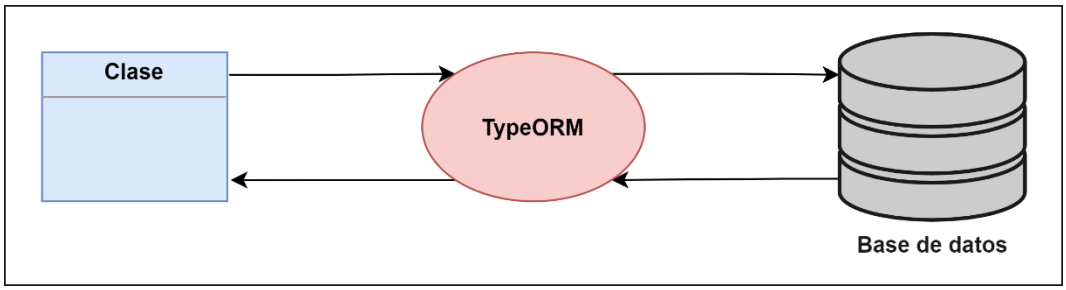

Figura 1.10. TypeORM

Se debe agregar que, TypeORM admite MySQL, MariaDB, Postgres, CockroachDB, SQLite, Microsoft SQL Server, Oracle, SAP Hana, sql.js y MongoDB (NoSQL), de modo que, el desarrollador puede elegir cualquier gestor de base de datos mencionado [20].

1.3.5.5. Angular

Angular es un *framework* que permite crear aplicaciones *frontend* escalables, está construido en TypeScript, consta de un grupo diverso de más de 1.7 millones de desarrolladores, autores de bibliotecas y creadores de contenido [21].

Para usar Angular se debe instalar el CLI (*Command Line Interface*) de Angular con el comando: `npm install -g @angular/cli`.

En la Figura 1.11 se presenta la descripción general de Angular. A continuación, se detalla cada elemento:

- ***Frontend***: lado del Cliente.
- ***Backend***: lado del Servidor.
- ***Component***: es el bloque principal de construcción de la aplicación Angular [22].
- **HTML**: es una plantilla o vista de la página [22].
- **CSS:** estilos aplicados a la plantilla [22].
- ***Service*:** es una clase con un propósito limitado y bien definido [23].

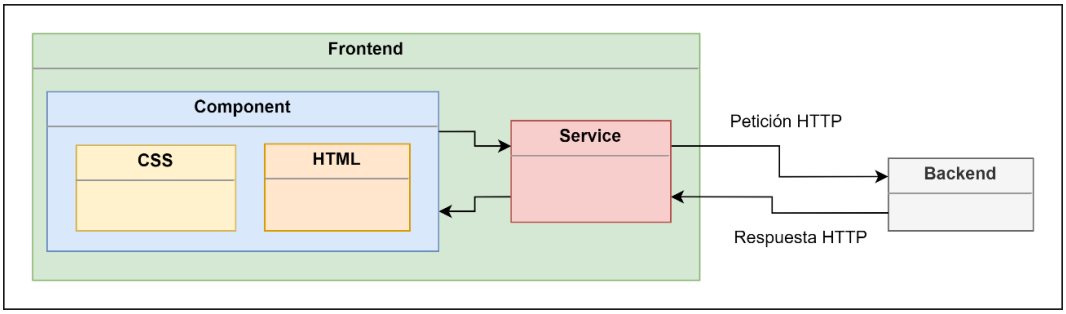

Figura 1.11. Descripción general de Angular

1.3.5.6. Postman

Postman es una aplicación de escritorio que permite simplificar el proceso de prueba de APIs (*Application Programming Interface*), además, elimina las dependencias y reduce el tiempo de producción, por tanto, los equipos de desarrollo *frontend* y *backend* pueden trabajar en paralelo [24].

En la Figura 1.12 se presenta una captura de Postman.

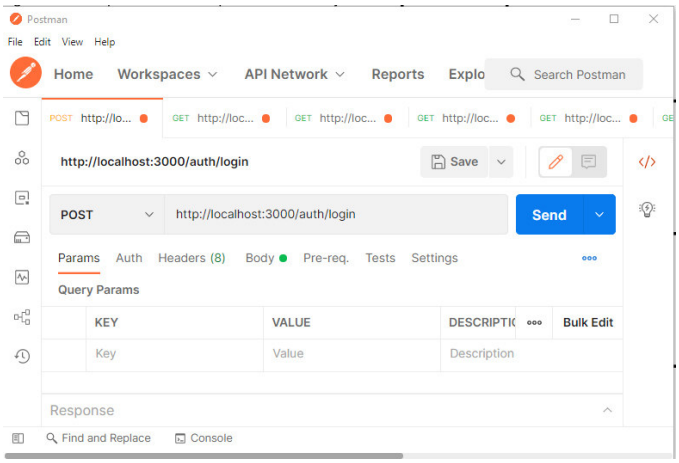

Figura 1.12. Captura Postman

1.3.5.7. Kanban [25]

Con el objetivo de aumentar la eficiencia de fabricación, el ingeniero industrial de Toyota Taiichi Ohno creó Kanban, de manera que, ayudó a respaldar sistemas de producción y promover mejoras adicionales.

En la metodología Kanban las tareas ocupan el primer lugar y solo un equipo trabaja en una tarea de inicio a fin. El equipo de Kanban no estima el tiempo para terminar una tarea, ya que, casi siempre será erróneo.

El tablero Kanban se lee de izquierda a derecha y está constituido por tres columnas con la siguiente información:

- **Tareas:** columna donde se coloca las tareas que están listas para iniciar. La tarea con mayor prioridad se coloca en la parte superior. La tarea continua a la siguiente columna.
- **Tareas en proceso:** columna donde permanecerá una tarea hasta que culmine su desarrollo. Cuando la tarea finalice pasa a la siguiente columna.
- **Tareas realizadas:** columna donde se encuentran todas las tareas que hayan superado las columnas anteriores.

Se debe agregar que, dependiendo el tipo de proyecto se puede agregar más columnas, a fin de esclarecer a los integrantes el flujo de trabajo y el progreso de los elementos del proyecto.

2. METODOLOGÍA

El presente Trabajo Titulación plantea desarrollar un prototipo de aplicación web utilizando la metodología ágil Kanban, por lo cual, se agregan tareas de las cuatro etapas de la elaboración del prototipo en un tablero definido por tres columnas de la siguiente manera: Tareas, Tareas en Proceso y Tareas Realizadas.

En resumen, la etapa uno consiste en determinar los requerimientos funcionales y no funcionales, luego, en la etapa dos se diseña el prototipo mediante los diagramas de arquitectura, entidad-relación, relacional, actividades, de clases y bosquejos de la interfaz gráfica, después, en la etapa tres se codifica el `backend` y `frontend`, para terminar, en la etapa cuatro se procede a desplegar el prototipo de aplicación web.

2.1. TABLERO KANBAN ETAPA UNO

En la Tabla 2.1 se presenta las actividades de la etapa uno, de modo que, se agregan las tareas que se deben realizarse en la primera columna y se muestran las tareas en proceso en la segunda columna.

Tabla 2.1. Tablero Kanban etapa uno

Tareas	Tareas en proceso	Tareas realizadas
Revisar características generales de los equipos (APs y STAs).	Realizar la primera entrevista al gerente técnico de la microempresa.	
Revisar la MIB de la casa fabricante.	Realizar consultas SNMP de prueba a los equipos (APs y STAs).	
Revisar el lenguaje de programación TypeScript.	Determinar requerimientos funcionales y no funcionales.	
Revisar la documentación de NestJS.		

2.2. LISTA DE REQUERIMIENTOS

Para determinar los requerimientos de la aplicación se realizó dos entrevistas al gerente técnico de la microempresa (en el ANEXO A se encuentran las preguntas realizadas), como resultado se obtuvo los requerimientos funcionales y no funcionales.

2.2.1. REQUERIMIENTOS FUNCIONALES

- La aplicación contará con los siguientes tres roles de usuario: Administrador, Analista y Usuario.
- El Administrador podrá ver, crear, actualizar y eliminar Personas con rol Analista o Usuario.
- El Administrador podrá leer, crear, actualizar o eliminar dispositivos (APs y STAs).
- El Administrador podrá ver las alarmas y los reportes de los dispositivos (APs y STAs).
- El Administrador podrá editar el tiempo de monitoreo y el límite de los parámetros a monitorear.
- El Analista podrá ver, crear, actualizar o eliminar dispositivos (APs y STAs).
- El Analista podrá ver las alarmas y los reportes de los dispositivos (APs y STAs).
- El Analista podrá editar el tiempo de monitoreo y el límite de los parámetros a monitorear.
- El Usuario podrá ver las alarmas y los reportes de los dispositivos (APs y STAs).
- Por cada AP que pierda conexión la aplicación deberá notificar por correo electrónico al administrador de la red o usuarios que hayan sido asignados para recibir correos electrónicos.
- La aplicación obtendrá la siguiente información del AP:
    - IP;
    - Nombre del Equipo;
    - MAC;
    - Modelo del Equipo;
    - SSID;
    - Frecuencia;
    - Ancho de Canal; y,
    - Número de Clientes.
- La aplicación obtendrá la siguiente información del STA:
    - IP;
    - Nombre del Equipo; y,
    - MAC.
- Los parámetros que la aplicación deberá monitorear son: Señal, Capacidad de Trasmisión, Capacidad de Recepción, Ruido y CCQ. Se debe agregar que, los límites de los parámetros a monitorear van a ser ingresados por la persona encargada de la administración de la red de la microempresa.

- La aplicación generará reportes de los enlaces caídos durante la semana, por lo que, la aplicación realizará esta operación cada lunes a la 01:00 AM.
- El usuario que quiera ingresar a la aplicación podrá hacerlo mediante correo electrónico o un nombre de usuario.
- La aplicación en la página principal mostrará la siguiente información:
    - Número total de APs;
    - Número total de STAs (Clientes);
    - El número total de problemas de monitoreo;
    - El número total de reportes generados por la aplicación;
    - El número total de APs *online* y *offline;*
    - El número total de STAs (Clientes) *online* y *offline;* y,
    - El número total de problemas de monitoreo por parámetro.
- La información de los dispositivos (APs y STAs) se visualizará en tablas.
- Deberá existir un filtro para buscar dispositivos (APs o STAs) por IP o Nombre en las tablas.
- La aplicación generará un PDF cuando el usuario solicite ver el detalle de algún reporte.
- Se mostrará los últimos 10 eventos ocurridos en la aplicación.
- El usuario con rol Analista y Usuario podrá cambiar su propia contraseña.
- La aplicación actualizará la página principal cuando cualquier dispositivo (AP o STA) se desconecte.
- La aplicación guardará automáticamente a los clientes (STAs) nuevos.

2.2.2. REQUERIMIENTOS NO FUNCIONALES

- Se usará TypeOrm con MySQL para crear las tablas de la base datos.
- Se usará el *framework* NestJS para desarrollar el *backend*
- Se usará el *framework* Angular junto con Angular Material para desarrollar el *frontend*.
- Se usará Postman como herramienta de apoyo para probar el funcionamiento del *backend.*
- Se usará PM2 y Apache para desplegar la aplicación.

2.3. TABLERO KANBAN ETAPA DOS

En la Tabla 2.2 se presenta las actividades de la etapa dos, por lo que, se agregan las tareas que se deben realizarse en la primera columna, se muestran las tareas en proceso en la segunda columna y se indica las tareas realizadas en la tercera columna.

Tabla 2.2. Tablero Kanban etapa dos

Tareas	Tareas en proceso	Tareas realizadas
Realizar una figura de la arquitectura del prototipo.	Revisar características generales de los equipos (APs y STAs).	Realizar la primera entrevista al gerente técnico de la microempresa.
Realizar el diagrama entidad relación y relacional de la base de datos.	Revisar la MIB de la casa fabricante.	Realizar consultas SNMP de prueba a los equipos (APs y STAs).
Realizar el diagrama de actividades del *backend*.	Revisar el lenguaje de programación TypeScript.	Determinar requerimientos funcionales y no funcionales.
Realizar el diagrama de clases del *backend*.	Revisar la documentación de NestJS.	
Realizar la segunda entrevista al gerente técnico de la microempresa y actualizar requerimientos.		
Realizar los *mockups*.		
Instalar Visual Studio Code, Node.js, MySQL, Git y Postman.		
Instalar el *package* del CLI de NestJS.		

2.4. DISEÑO DEL PROTOTIPO

2.4.1. ARQUITECTURA

En la Figura 2.1 se presenta el esquema del prototipo de aplicación web, por consiguiente, se usa los métodos HTTP (*GET*, *POST*, *PUT* y *DELETE*) para realizar el CRUD (*Create*, *Read*, *Update* and *Delete*) con los módulos que lo permitan, también, se utiliza la tecnología SSE para realizar la actualización de los datos del *dashboard* web. Por otra parte, se emplean los protocolos ICMP, SNMP y SYSLOG para el monitoreo de los dispositivos (APs

y STAs). Además, el manejo de la base de datos se la hace mediante el uso de un ORM para leer, guardar, actualizar o eliminar los registros.

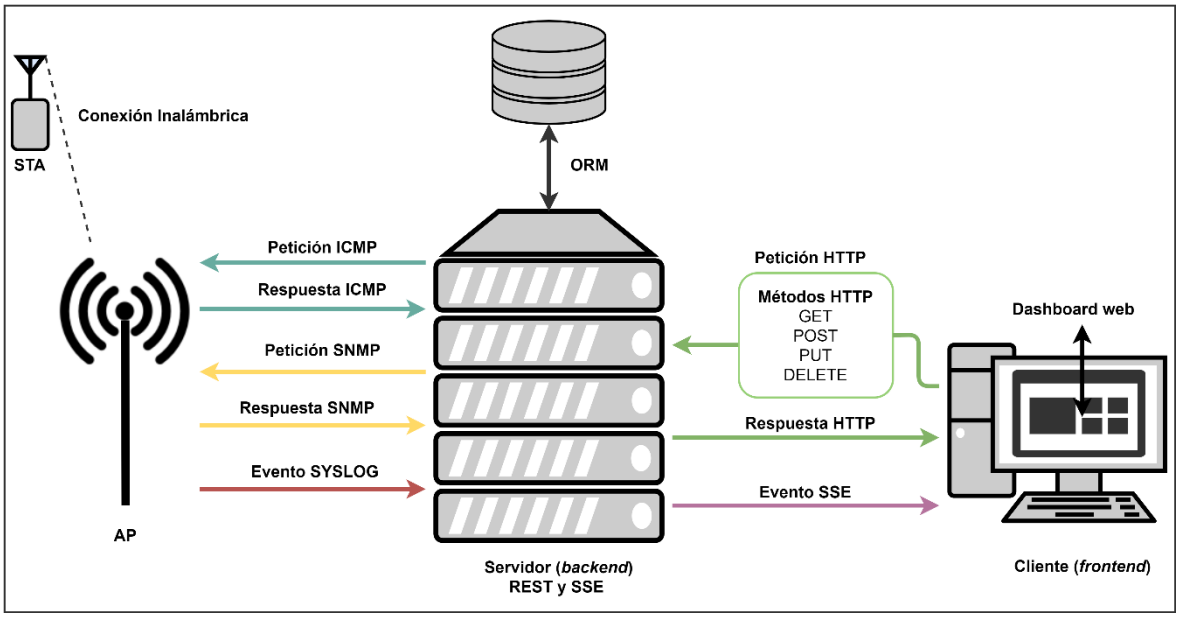

Figura 2.1. Arquitectura del Prototipo

2.4.2. DIAGRAMA ENTIDAD-RELACIÓN

El diagrama entidad-relación representa como está modelada la base de datos mediante el uso de entidades (rectángulos), atributos de una entidad (óvalos) y asociaciones entre entidades (rombos) con su cardinalidad. En la Figura 2.2 se describe el diagrama entidad-relación de la aplicación. A continuación, se detalla cada entidad.

La entidad `ap` se asocia con la entidad `sta` con una cardinalidad de uno a muchos, por consiguiente, un AP puede tener muchas STAs, además, la entidad `ap` se asocia con la entidad `alarma_ap` con una cardinalidad de uno a muchos, así que, un AP puede tener muchas alarmas.

La entidad `sta` se asocia con la entidad `alarma_sta` con una cardinalidad de uno a muchos, de modo que, una STA puede tener muchas alarmas. También, la entidad `monitoreo_sta` se asocia con la entidad `sta` con una cardinalidad de muchos a uno, por tanto, muchos monitoreos de STA pueden tener una STA.

Las entidades `alarma_ap` y `monitoreo_sta` se asocia con la entidad `tipo_alarma` con una cardinalidad de muchos a uno respectivamente, por ende, muchas alarmas de AP y Monitoreo STA pueden tener un tipo de alarma. Además, la entidad `tipo_alarma` se asocia con la entidad `alarma_sta` con una cardinalidad de uno a muchos, por lo que, un tipo de alarma puede tener muchas alarmas de STA.

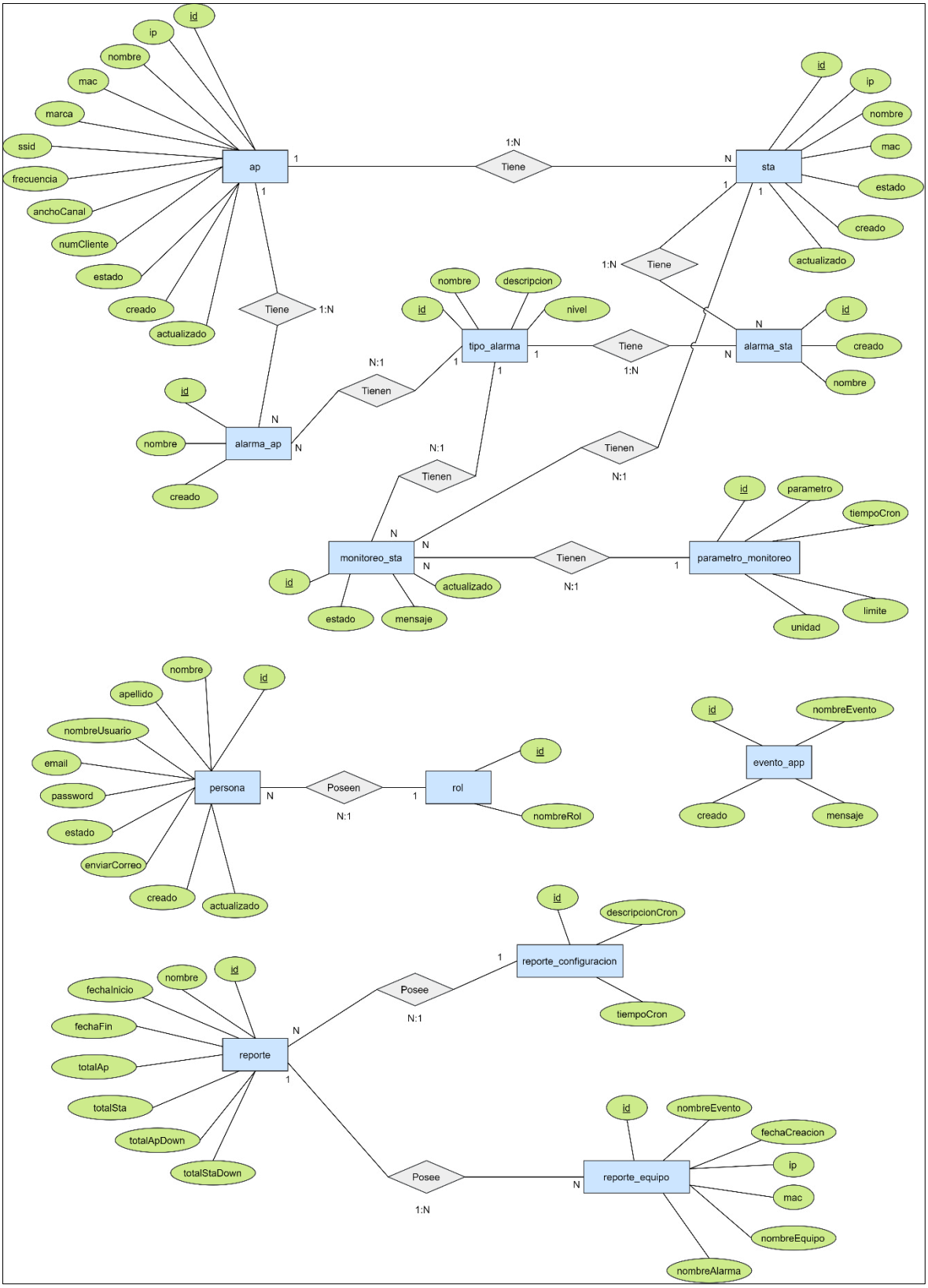

Figura 2.2. Diagrama Entidad-Relación

La entidad monitoreo_sta se asocia con la entidad parametro_monitoreo con una cardinalidad de muchos a uno, por lo cual, muchos monitoreos de STA pueden tener un parámetro de monitoreo.

La entidad persona se asocia con la entidad rol con una cardinalidad de muchos a uno, de manera que, muchas personas pueden poseer un rol.

La entidad reporte se asocia con la entidad reporte_configuracion con una cardinalidad de muchos a uno, en efecto, muchos reportes poseen un reporte de configuración. Además, la entidad reporte se asocia con la entidad reporte_equipo con una cardinalidad de uno a muchos, en tal sentido, un reporte posee muchos reportes de equipos.

La entidad evento_app no se asocia con ninguna entidad, puesto que, esta entidad sirve para realizar auditoría de la base de datos.

En el ANEXO B se encuentra más detallado el diagrama entidad-relación.

2.4.3. DIAGRAMA RELACIONAL

En la Figura 2.3 se presenta el diagrama relacional de la aplicación. A continuación, se describe la función de cada tabla.

Las tablas ap y sta almacenan datos de las consultas SNMP realizadas a los APs. De forma similar, la información de monitoreo de los STAs, que se obtiene mediante poleos SNMP a los APs, se la guarda en la tabla monitoreo_sta.

En la tabla tipo_alarma se guarda la información de los tipos de alarmas que tiene la aplicación. Por otra parte, cuando la aplicación detecta inconvenientes con los equipos (APs y STAs), se guarda esta información en las tablas alarma_ap y alarma_sta según corresponda.

La tabla parametro_monitoreo guarda información respecto a qué parámetros van a ser monitoreados por la aplicación.

La tabla persona guarda información sobre los usuarios que podrán ingresar a interactuar con la aplicación. Por otro lado, la tabla rol guarda datos sobre los roles.

La tabla reporte guarda información sobre los reportes que genera la aplicación, también, la tabla reporte_equipo guarda datos más detallados sobre los equipos que pertenecen a cada reporte. Se debe agregar que, la tabla reporte_configuracion guarda datos sobre cuándo la aplicación debe realizar el reporte.

La tabla `evento_app` se la considera una tabla de auditoría que guarda información sobre eventos (guardar, actualizar o eliminar registros) que ocurren en las tablas `ap`, `sta`, `persona`, `parametro_monitoreo` y `reporte`.

En el ANEXO C se encuentra más detallado el diagrama relacional.

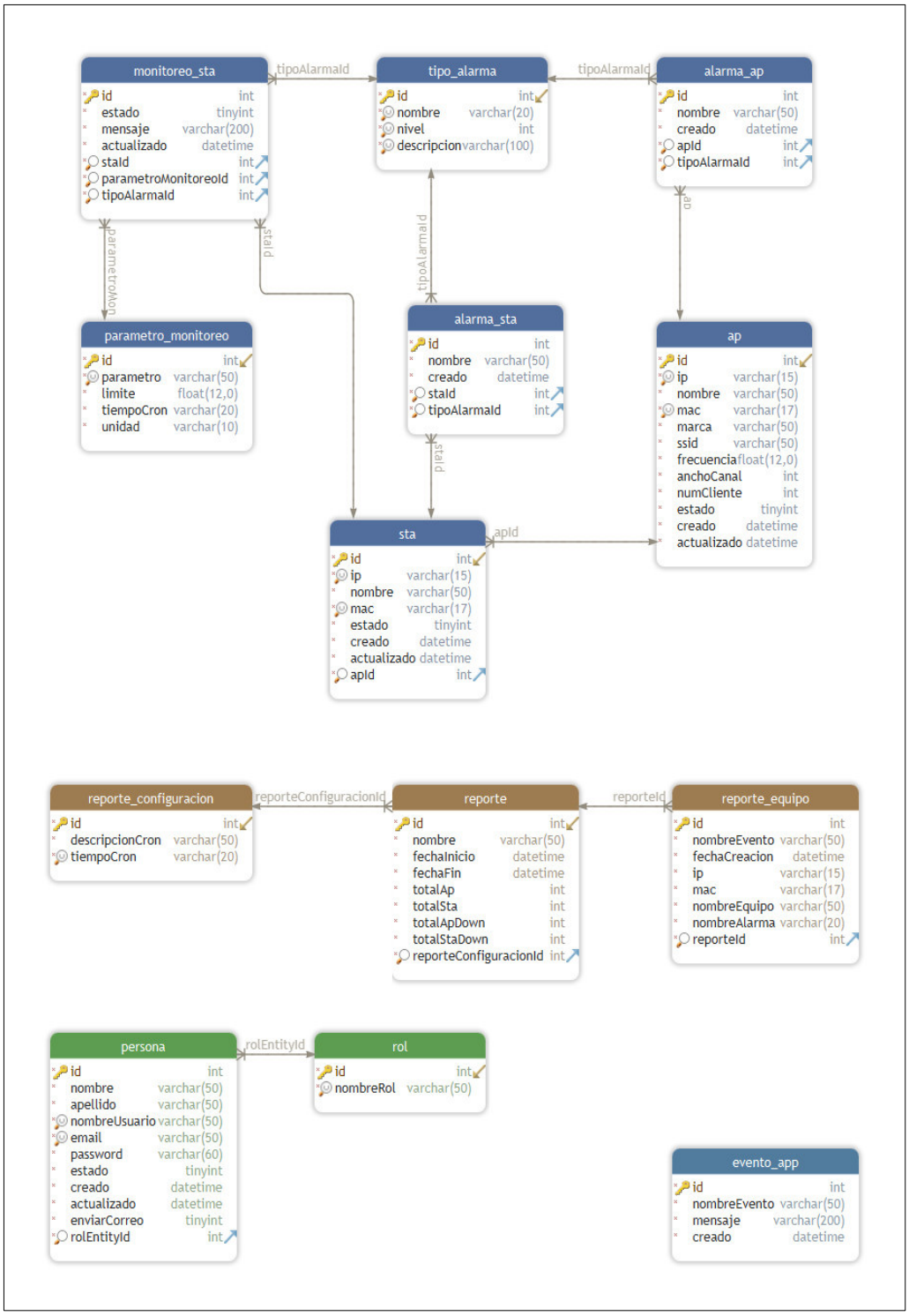

Figura 2.3. Diagrama Relacional

2.4.4. DIAGRAMA DE ACTIVIDADES

Los diagramas de actividades describen el comportamiento de un proceso desde que inicia hasta que termina. A continuación, se muestra los diagramas de actividades del procedimiento de receptar *logs*, monitorear con ICMP y monitorear con SNMP.

En la Figura 2.4 se presenta el diagrama de actividades para receptar *logs*, por lo que, el proceso inicia cuando el AP envía un *log* al Servidor, luego, el Servidor recepta el *log* y procede a filtrarlo, de modo que, si el mensaje del *log* no coincide con ninguna palabra clave el proceso termina, caso contrario, el Servidor guarda la alarma del mensaje *log* y envía un evento al *Dashboard* web, después, el *Dashboard* web recepta el evento y lo filtra, si el evento no coincide con la regla actualizar datos el proceso finaliza, caso contrario, se muestra los nuevos datos en pantalla y el proceso culmina.

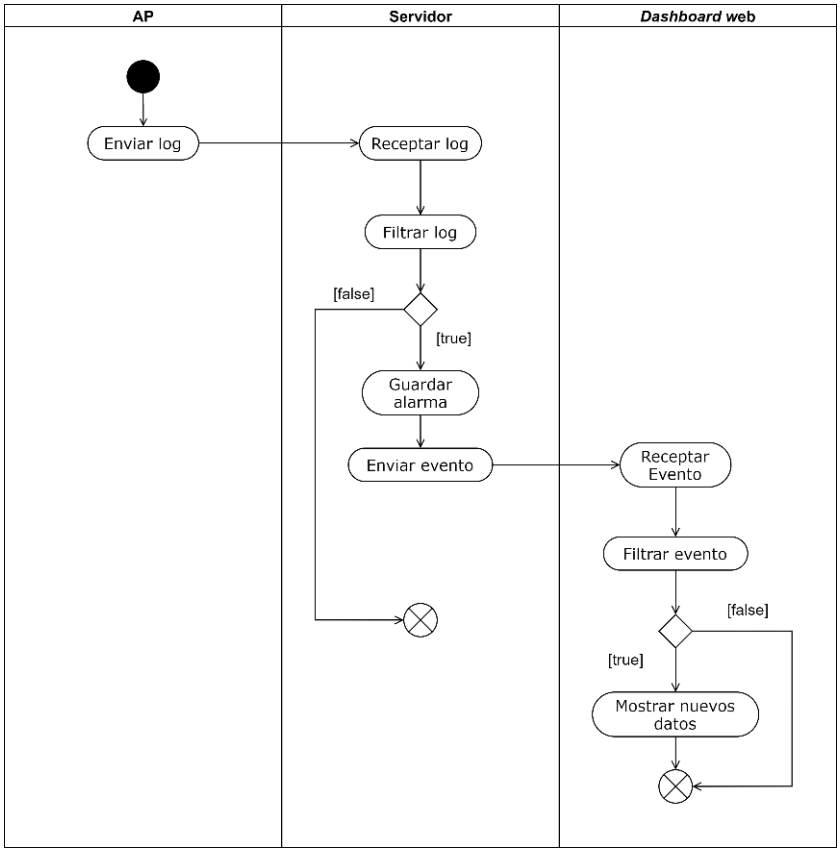

Figura 2.4. Diagrama de actividades de receptar *logs*

En la Figura 2.5 se presenta el diagrama de actividades para monitorear APs con ICMP, por tanto, el proceso inicia cuando el Servidor envía “X” paquetes Ping, luego, si el AP está *online* recepta la petición y envía “X” respuestas, caso contrario, se envía X = 0 que significa que no hay respuesta del AP, después, el Servidor recibe las “X” respuestas y verifica el valor de “X”, si el valor de “X” es diferente de cero entonces termina el proceso, caso

contrario, guarda la alarma, envía un correo electrónico y envía un evento al *Dashboard*, luego, el *Dashboard* web recepta el evento y lo filtra, si el evento no coincide con la regla actualizar datos el proceso finaliza, caso contrario, se muestra los nuevos datos en pantalla y el proceso culmina.

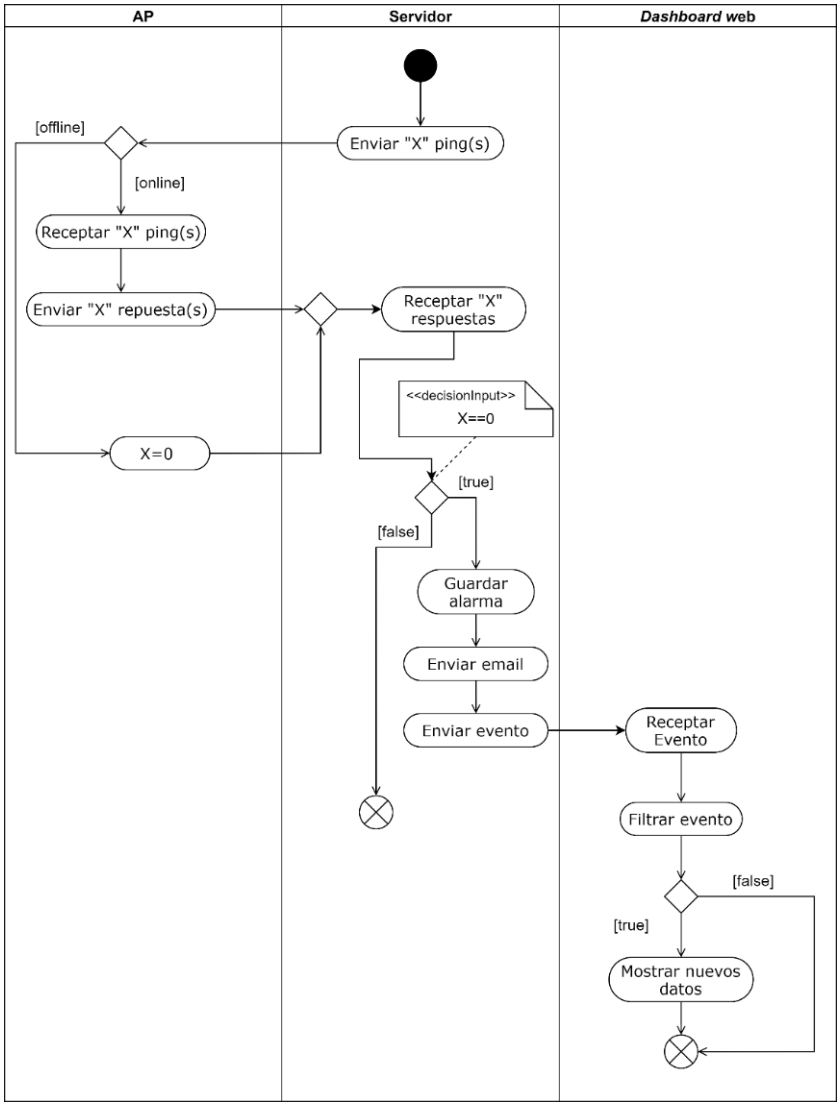

Figura 2.5. Diagrama de actividades del monitoreo con ICMP

En la Figura 2.6 se presenta el diagrama de actividades para monitorear STAs mediante el AP, por lo que, el proceso inicia cuando el Servidor envía una consulta SNMP al AP, luego, el AP recepta la consulta SNMP, procesa la consulta y envía los datos solicitados, después, el Servidor recepta los datos, guarda o actualiza los datos receptados y el Servidor envía un evento al *Dashboard* web. Por otro lado, el *Dashboard* web recepta el evento y lo filtra, si el evento no coincide con la regla refrescar se termina el proceso, caso contrario, el *Dashboard* web hace una petición HTTP *GET* al Servidor, por otra parte, el Servidor recepta la petición HTTP *GET*, procesa la petición y envía los datos solicitados al *Dashboard* web,

finalmente, el *Dashboard* web recepta los datos, muestra los nuevos datos en pantalla y se termina el proceso.

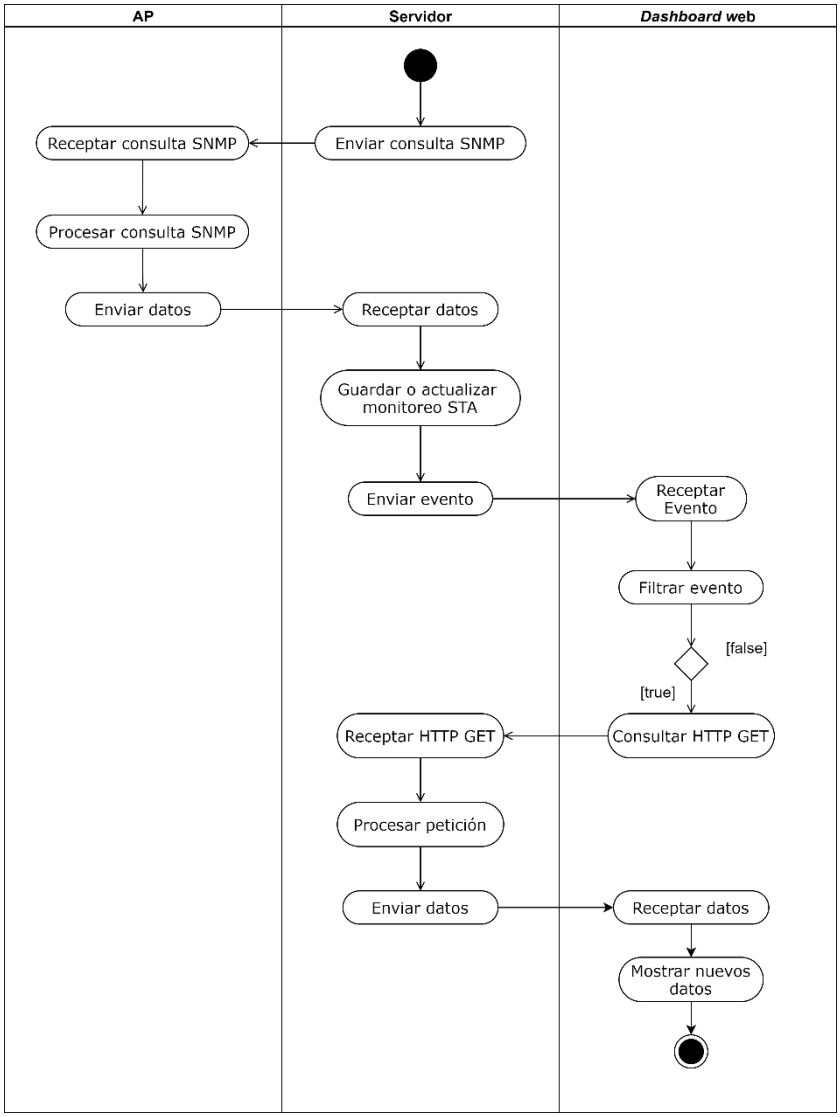

Figura 2.6. Diagrama de actividades del monitoreo con SNMP

En el ANEXO D se encuentra más detallado los diagramas de actividades.

2.4.5. DIAGRAMA DE CLASES

El diagrama de clases representa la estructura de la aplicación mediante el uso de entidades (clases), interfaces, herencias y dependencias. Por consiguiente, se realiza el diagrama de clases por módulo, ya que, una característica del *framework* NestJS es que permite crear aplicaciones modulares. A continuación, se presenta dos diagramas de clases de la aplicación, el resto diagramas se encuentran en el ANEXO E.

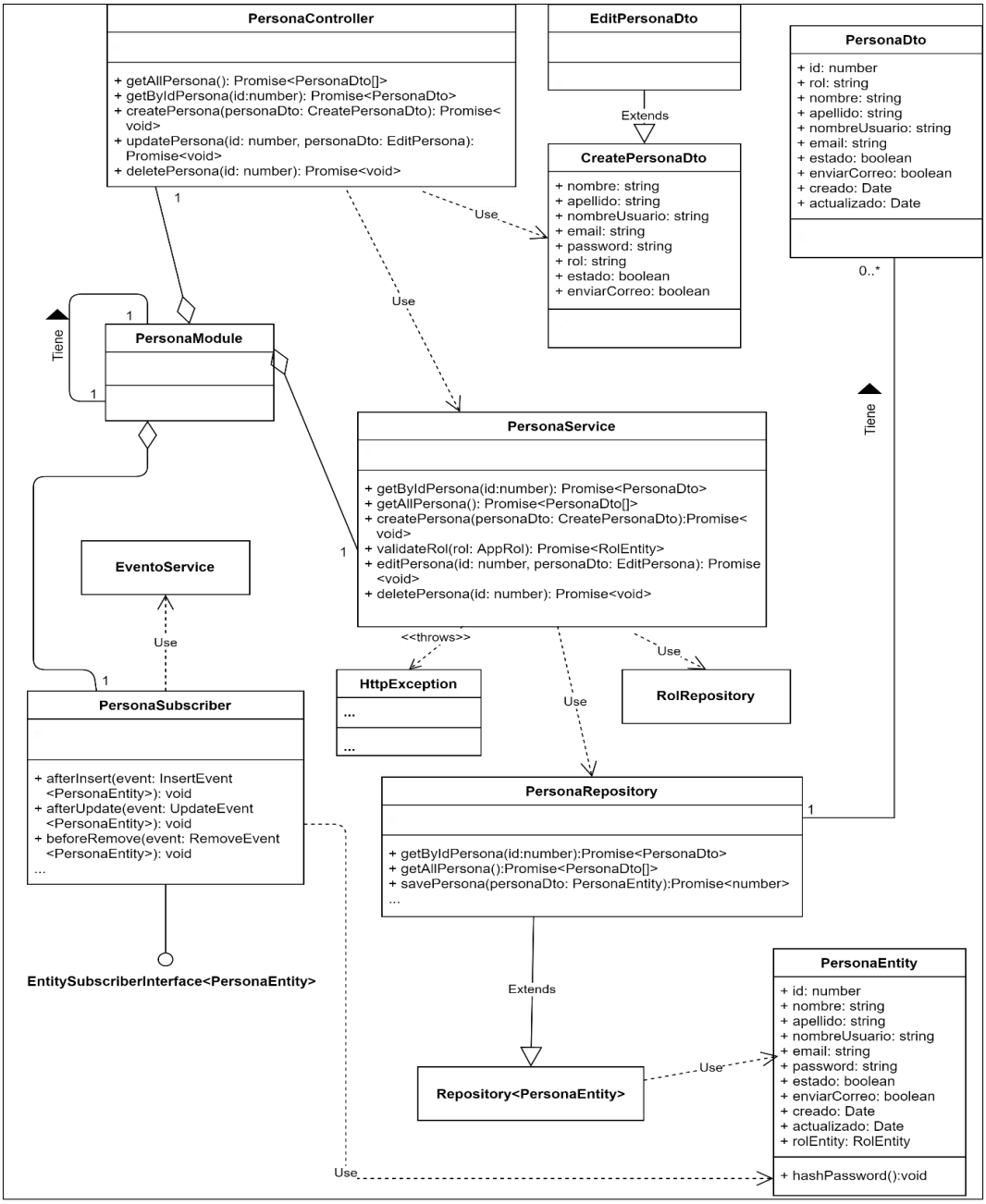

Figura 2.7. Diagrama de clases del módulo `PersonaModule`

En la Figura 2.7 se representa el diagrama de clases del módulo `PersonaModule`. A continuación, se listan todas las clases:

- **`PersonaModule:`** clase principal que instancia a las clases `PersonaController`, `PersonaService y PersonaSubscriber`.
- **`PersonaController:`** clase que recepta las peticiones HTTP.
- **`PersonaService:`** clase que gestiona las peticiones HTTP.

- **PersonaSubscriber:** clase que escucha los eventos (*Insert, Update or Remove*) entre la clase `PersonaEntity` y la base de datos.
- **EventoService:** esta clase se la detalla en el módulo `EventoModule`.
- **PersonaRepository:** clase que implementa métodos personalizados para interactuar con la base de datos.
- **HttpException:** clase de NestJS para el manejo de excepciones HTTP.
- **RolRepository:** esta clase se la detalla en el módulo `RolModule`.
- **Repository<PersonaEntity>:** clase de TypeORM que posee métodos para trabajar con la base de datos, por ejemplo: buscar, guardar, eliminar, entre otros.
- **PersonaEntity:** clase que se le asigna a la tabla `persona` para manipular sus registros mediante la ayuda de clase `PersonaRepository`.
- **PersonaDto:** clase instanciada por la clase `PersonaRepository` para filtrar la información obtenida de la base de datos.
- **CreatePersonaDto:** clase que ayuda a crear personas.
- **EditPersonaDto:** clase que ayuda editar personas.

En la Figura 2.8 se representa el diagrama de clases del módulo `ApModule`. A continuación, se listan todas las clases:

- **ApModule:** clase principal que instancia a las clases `ApController`, `ApService y ApSubscriber`.
- **ApController:** clase que recepta las peticiones HTTP.
- **ValidationIpPipe:** esta clase se la detalla en el módulo `EquipoModule`.
- **ApService:** clase que gestiona las peticiones HTTP.
- **ApSubscriber:** clase que escucha los eventos (*Insert, Update or Remove*) entre la clase `ApEntity` y la base de datos.
- **EventoService:** esta clase se la detalla en el módulo `EventoModule`.
- **HttpException:** clase de NestJS para el manejo de excepciones HTTP.
- **StaRepository:** esta clase se la detalla en el módulo `StaModule`.
- **ApRepository:** clase que implementa métodos personalizados para interactuar con la base de datos.
- **Repository<ApEntity>:** clase de TypeORM que posee métodos para trabajar con la base de datos, por ejemplo: buscar, guardar, eliminar, entre otros.
- **ApEntity:** clase que se le asigna a la tabla `ap` para manipular sus registros mediante la ayuda de clase `ApRepository`.

- **`ApDto:`** clase instanciada por `ApService` para filtrar la información obtenida de la base de datos.
- **`CreateApDto:`** clase que ayuda a crear o actualizar APs.
- **`PaquetePingDto:`** esta clase se detalla en el módulo `EquipoModule`.

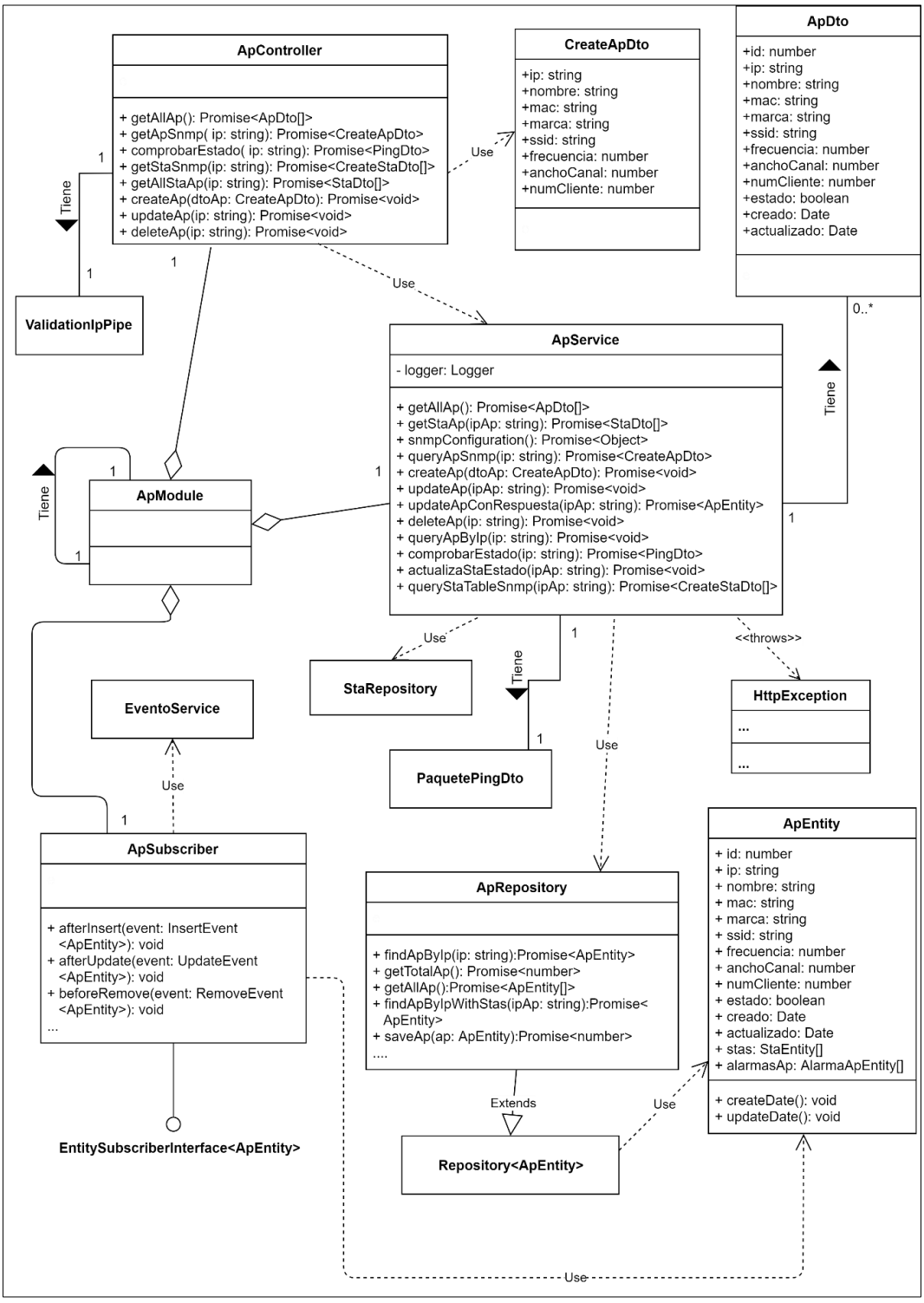

Figura 2.8. Diagrama de clases del módulo `ApModule`

2.4.6. MOCKUPS

A continuación, se representa los bosquejos del *dashboard* web, en el ANEXO F se encuentra detallado los *mockups*.

En la Figura 2.9 se muestra el bosquejo de la página web que permitirá realizar la autenticación de usuarios de la aplicación web.

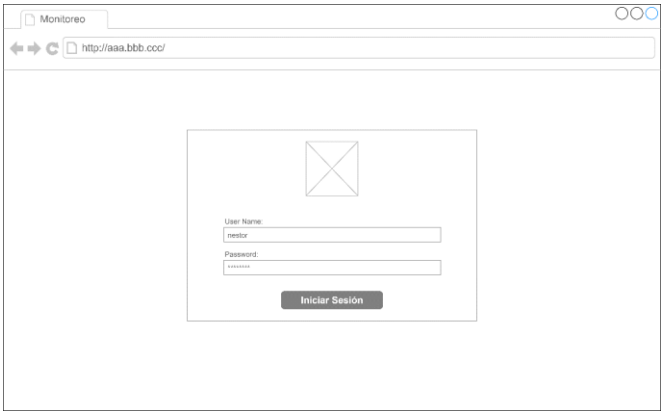

Figura 2.9. Bosquejo de la página web `login`

En la Figura 2.10 se muestra el boceto de una *card* que permitirá mostrar información de totales, por ejemplo: Total APs.

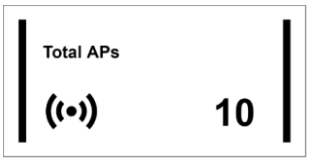

Figura 2.10. Boceto de la *Card* de totales

En la Figura 2.11 se muestra el bosquejo de la página web que permitirá mostrar la información de totales (*dashboard*).

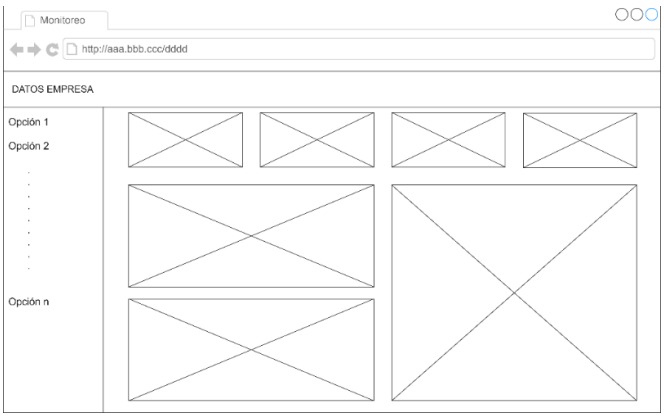

Figura 2.11. Bosquejo de la página web `dashboard`

En la Figura 2.12 se muestra el bosquejo de la página web que permitirá mostrar la información de equipos (APs o STAs), personas o reportes.

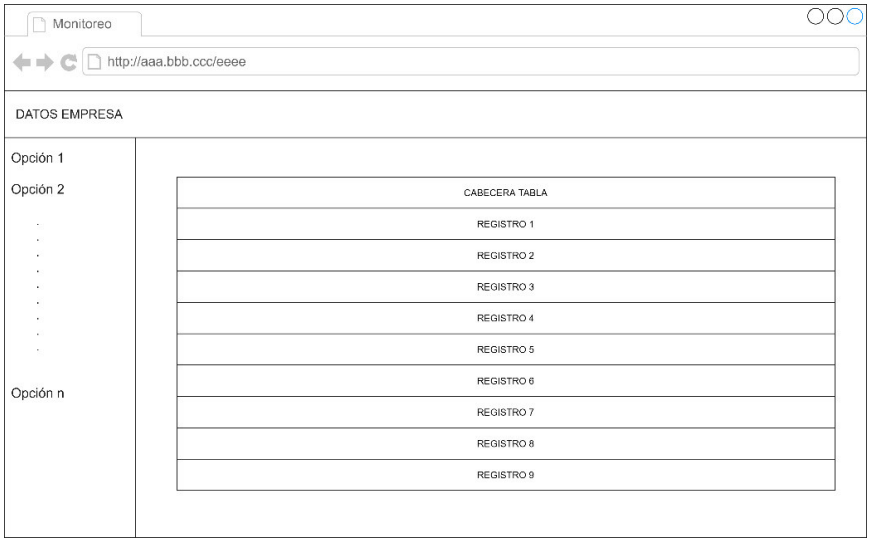

Figura 2.12. Bosquejo de la página web `equipo`

En la Figura 2.13 se muestra el bosquejo de la página web para formularios, por ejemplo, ingreso de personas.

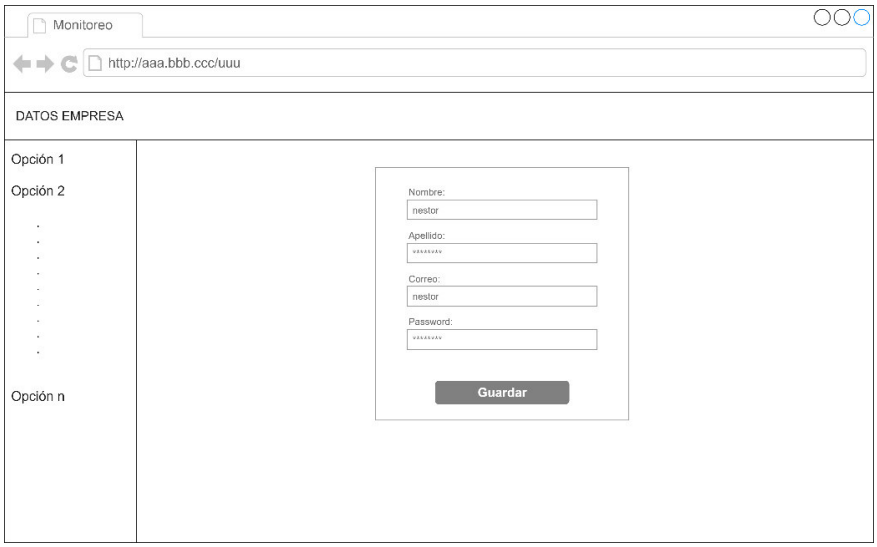

Figura 2.13. Bosquejo de la página web para formularios

2.5. TABLERO KANBAN ETAPA TRES

En la Tabla 2.3 se presenta las actividades de la etapa tres, de manera que, se agregan las tareas que se deben realizarse en la primera columna, se muestran las tareas en proceso en la segunda columna y se indica las tareas realizadas en la tercera columna.

Tabla 2.3. Tablero Kanban etapa tres

Tareas	Tareas en proceso	Tareas realizadas
Codificar la aplicación `backend`	Realizar la segunda entrevista al gerente técnico de la microempresa y actualizar requerimientos.	Realizar la primera entrevista al gerente técnico de la microempresa.
Probar el funcionamiento del `backend`.	Realizar los *mockups.*	Realizar consultas SNMP de prueba a los equipos (APs y STAs).
Codificar la aplicación `frontend`	Instalar Visual Studio Code, Node.js, MySQL, Git y Postman.	Determinar requerimientos funcionales y no funcionales.
Probar el funcionamiento del `frontend`.		Revisar características generales de los equipos (APs y STAs).
		Revisar la MIB de la casa fabricante.
		Revisar el lenguaje de programación TypeScript.
		Revisar la documentación de NestJS.
		Realizar una figura de la arquitectura del prototipo.
		Realizar el diagrama entidad relación y relacional de la base de datos.
		Realizar el diagrama de actividades del *backend.*
		Realizar el diagrama de clases del *backend*.

2.6. IMPLEMENTACIÓN DEL PROTOTIPO

La implementación del prototipo de aplicación web del presente Trabajo de Titulación se la desarrolló utilizando las siguientes herramientas:

- Laptop con sistema operativo Windows 10.
- Visual Studio Code para la edición de código [26].
- Node.js 12.09 para crear la aplicación web [27].
- NestJS v7.5.1 para la creación de la aplicación *backend* [28].

- TypeORM con MySQL para la creación de la base de datos [29].
- Angular v10 para la creación de la aplicación *frontend* [30].
- Angular Material v10.2.7 para la creación de la interfaz del *dashboard* web [31].
- GitHub para crear el repositorio de la aplicación web [32].
- Postman para probar los módulos de la aplicación *backend* [33].
- La librería net-snmp para las consultas SNMP [3].
- La librería ping para las consultas ICMP [5].
- La librería syslog-server para la recepción de *logs* [34].
- La librería nodemailer para enviar correos electrónicos [35].
- La librería pdfmake-wrapper para generar el pdf [36].
- La librería angular-sse-client para crear la conexión con el Servidor SSE [37].

2.6.1. APLICACIÓN *BACKEND*

Para la implementación del *backend* se utilizó el *framework* NestJS [1], por consiguiente, se usó el comando `$ nest new backend` del CLI de NestJS para generar el proyecto. (Ver Figura 2.14)

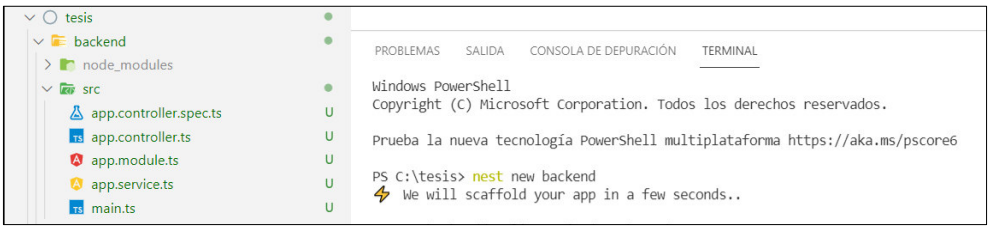

Figura 2.14. Creación del proyecto `backend`

2.6.1.1. Repositorio

Para la generación del repositorio `backend` se procedió a crear un repositorio en GitHub. (Ver Figura 2.15)

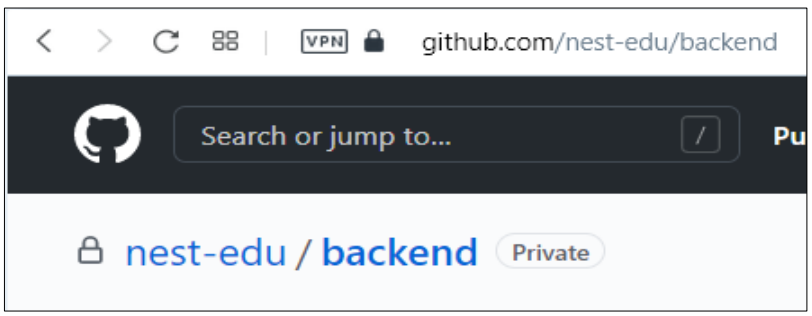

Figura 2.15. Repositorio `backend`

Los comandos para unir la aplicación `backend` local con el repositorio remoto `backend` son `$ git init` y `$ git remote set-url origin https://nest-edu:TOKEN@github.com/nest-edu/backend.git` . La palabra TOKEN representa el *token* generado por la cuenta personal para la administración de los repositorios [38].

2.6.1.2. Base de Datos

Para la implementación de la base de datos se utilizó MySQL y TypeORM. Por lo cual, se procedió a crear una base de datos en MySQL e instalar el módulo de TypeORM [29] en la aplicación `backend` con el siguiente comando: `$ npm install --save @nestjs/typeorm typeorm mysql2`.

En el Código 2.1 se muestra cómo realizar la conexión entre la base de datos y la aplicación `backend`, así que, la línea 25 indica que el archivo `.env` posee la configuración inicial de la aplicación, además, especifica que el módulo `ConfigModule` puede ser utilizado por cualquier submódulo. En la línea 27 se inyecta el servicio `ConfigService` del módulo `ConfigModule`, se realiza la inyección para que el módulo `TypeOrmModule` pueda acceder a la información de configuración del archivo `.env`. Desde la línea 28 a la 34 se obtiene los valores especificados en el archivo `.env` para realizar la conexión. La línea 35 denota que se va a buscar todas las entidades en la carpeta de la aplicación `backend`. La línea 36 y 37 se utiliza para el desarrollo de la aplicación, por ende, cuando se despliegue la aplicación se debe colocar en `false` estas propiedades.

```
25    ConfigModule.forRoot({ isGlobal: true, envFilePath: '.env' }),
26    TypeOrmModule.forRootAsync({
27      inject: [ConfigService],
28      useFactory: (config: ConfigService) => ({
29        type: 'mysql',
30        host: config.get<string>(DATABASE_HOST),
31        port: parseInt(config.get<string>(DATABASE_PORT), 10),
32        username: config.get<string>(DATABASE_USERNAME),
33        password: config.get<string>(DATABASE_PASSWORD),
34        database: config.get<string>(DATABASE_NAME),
35        entities: [__dirname + '/**/*.entity{.ts,.js}'],
36        synchronize: true,
37        logging: true,
38      }),
39    }),
```

Código 2.1. Conexión con la base de datos

A continuación, se muestra cómo generar las tablas `persona` y `rol` con su respectiva relación.

En el Código 2.2 se presenta la primera parte de la clase `PersonaEntity` que mapea la tabla `persona`, por lo cual, se inicia especificando el nombre de la tabla (línea 12), se crea el atributo `id` con propiedad incrementable (líneas 14-15); se crean los atributos `nombre`, `apellido`, `nombreUsuario` y `email` que poseen el tipo `varchar`, longitud de 50 caracteres, y propiedad no nula (líneas 17-27); se crea el atributo `password` de tipo `varchar`, longitud de 60 caracteres y propiedad no nula (líneas 29-30); se crean los atributos `estado` y `enviarCorreo` que poseen el tipo `bool`, y propiedad no nula (líneas 32-36); se crean los atributos `creado` y `actualizado` que poseen el tipo `datetime` y propiedad no nula (líneas 38-42).

```
12  @Entity({ name: 'persona' })
13  export class PersonaEntity {
14    @PrimaryGeneratedColumn('increment')
15    id: number;
16
17    @Column({ type: 'varchar', length: 50, nullable: false })
18    nombre: string;
19
20    @Column({ type: 'varchar', length: 50, nullable: false })
21    apellido: string;
22
23    @Column({ type: 'varchar', length: 50, nullable: false, unique: true })
24    nombreUsuario: string;
25
26    @Column({ type: 'varchar', length: 50, nullable: false, unique: true })
27    email: string;
28
29    @Column({ type: 'varchar', length: 60, nullable: false, select: false })
30    password: string;
31
32    @Column({ type: 'bool', nullable: false })
33    estado: boolean;
34
35    @Column({ type: 'bool', nullable: false })
36    enviarCorreo: boolean;
37
38    @Column({ type: 'datetime', nullable: false })
39    creado: Date;
40
41    @Column({ type: 'datetime', nullable: false })
42    actualizado: Date;
```

Código 2.2. Entidad `PersonaEntity` (1/2)

En el Código 2.3 se presenta la segunda parte de la clase `PersonaEntity` que indica los métodos de la clase y la relación entre las tablas `persona` y `rol`, de modo que, el método `createDate` asigna la fecha automáticamente cuando se crea la entidad (líneas 44-47), el método `update` asigna la fecha automáticamente cuando se actualiza la entidad (líneas 49-53), la función `@ManyToOne` realiza la relación de uno a muchos entre las tablas `rol` y `persona`, además, se detalla las propiedades de borrado en cascada (`onDelete`), columna no nula (`nullable`) y cuando se realice una consulta con el método `find` se extraiga de la base de datos los registros de personas con su respectivo rol o viceversa

(`eager`) (líneas 55-64); el método `hashPassword` aplica *hash* a la contraseña antes de guardar o actualizar la entidad en la base de datos (líneas 66-75).

```
43
44  @BeforeInsert()
45  createDate() {
46    this.creado = new Date();
47  }
48
49  @BeforeInsert()
50  @BeforeUpdate()
51  updateDate() {
52    this.actualizado = new Date();
53  }
54
55  @ManyToOne(
56    type => RolEntity,
57    rolEntity => rolEntity.personasEntity,
58    {
59      onDelete: 'CASCADE',
60      nullable: false,
61      eager: true,
62    },
63  )
64  rolEntity: RolEntity;
65
66  /**Aplica Hash al password */
67  @BeforeInsert()
68  @BeforeUpdate()
69  async hashPassword() {
70    if (!this.password) {
71      return;
72    }
73
74    this.password = await hash(this.password, 10);
75  }
76 }
```

Código 2.3. Entidad `PersonaEntity` (2/2)

En el Código 2.4 se presenta la clase `RolEntity` que mapea la tabla `rol`, por tanto, se inicia especificando el nombre de la tabla (línea 4), se crea el atributo `id` con propiedad incrementable (líneas 6-7); se crea el atributo `nombreRol` de tipo `varchar`, longitud de 50 caracteres, propiedad no nulo y valor único (líneas 9-10), así mismo, se muestra la función `@OneToMany` que realiza la relación de uno a muchos entre las tablas `rol` y `persona`, además, se detalla las propiedad de borrado en cascada (`onDelete`) (líneas 12-19).

```
4   @Entity({ name: 'rol' })
5   export class RolEntity {
6     @PrimaryGeneratedColumn('increment')
7     id: number;
8
9     @Column({ type: 'varchar', length: 50, nullable: false, unique: true })
10    nombreRol: AppRol;
11
12    @OneToMany(
13      type => PersonaEntity,
14      personaEntity => personaEntity.rolEntity,
15      {
16        onDelete: 'CASCADE',
17      },
18    )
19    personasEntity: PersonaEntity[];
20
21  }
```

Código 2.4. Entidad `RolEntity`

De forma similar, el resto de las tablas fueron creadas mediante el proceso detallado anteriormente. Los archivos que contienen las entidades se encuentran en la carpeta `backend/src/` del ANEXO G y tienen la sintaxis `<nombre>.entity.ts`.

2.6.1.3. Módulo Rol

El módulo `RolModule` posee elementos que ayudan a establecer el proceso de obtener Roles. A continuación, se menciona los elementos `RolService` y `RolController` para obtener roles.

En el Código 2.5 se presenta el método `getAllRol` (línea 9) de la clase `RolService`, por lo que, se declara un *array* de tipo `RolDto` (línea 10), se busca los roles en la base de datos (línea 14), se valida si existen roles (líneas 15-17), luego, se filtra los datos obtenidos de la base de datos (líneas 18-23) y se retorna los roles encontrados (línea 24).

```
12  async getAllRol(): Promise<RolDto[]> {
13    let arrayRol: Array<RolDto> = [];
14    const roles = await this.rolRepository.find();
15    if (!roles) {
16      return undefined;
17    }
18    for (let iter in roles) {
19      let rol = new RolDto();
20      rol.id = roles[iter].id;
21      rol.nombreRol = roles[iter].nombreRol;
22      arrayRol.push(rol);
23    }
24    return arrayRol;
25  }
```

Código 2.5. Método para obtener roles

En el Código 2.6 se presenta la función `@Get` que recepta las peticiones HTTP *GET*, además, se muestra el método `getAllRol` de la clase `RolController` (línea 13), por lo que, se llama al método `getAllRol` de la clase `RolService` para obtener los roles (línea 14).

```
12  @Get()
13  async getAllRol(): Promise<RolDto[]> {
14    return await this.rolService.getAllRol();
15  }
```

Código 2.6. Método para receptar peticiones HTTP *GET*

En la carpeta `backend/src/rol` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Rol.

2.6.1.4. Módulo Persona

El módulo `PersonaModule` posee varios elementos que ayudan a establecer los procesos de leer, crear, editar o eliminar Personas. A continuación, se menciona los elementos `PersonaService` y `PersonaController` para crear personas.

En el Código 2.7 se presenta el método `createPersona` (línea 85) de la clase `PersonaService`, por ende, se procede a extraer los atributos `nombreUsuario`, `email` y `rol` de `personaDto` (línea 86); se realiza una búsqueda por `nombreUsuario` o `email` en la tabla `persona` (líneas 87-89), se valida si existe la Persona y se utiliza la clase `HttpException` para el manejo de excepciones (líneas 90-99); se valida si existe el Rol (línea 100) y se crea la entidad `PersonaEntity` (líneas 101-102), luego, se guarda la entidad `PersonaEntity` en la base de datos (línea 103) y para el manejo de errores en el proceso de guardado se usa la clase `HttpException` (líneas 104-122).

```
85  async createPersona(personaDto: CreatePersonaDto): Promise<void> {
86    const { nombreUsuario, email, rol } = personaDto;
87    const existPersona = await this.personaRespository.findOne({
88      where: [{ nombreUsuario: nombreUsuario }, { email: email }],
89    });
90    if (existPersona) {
91      throw new HttpException(
92        {
93          status: HttpStatus.CONFLICT,
94          message: 'Conflict',
95          error: 'Conflict',
96        },
97        HttpStatus.CONFLICT,
98      );
99    }
100   const rolPersona = await this.validateRol(rol as AppRol);
101   const persona = this.personaRespository.create(personaDto);
102   persona.rolEntity = rolPersona;
103   const resultado = await this.personaRespository.savePersona(persona);
104   if (resultado == 0) {
105     throw new HttpException(
106       {
107         status: HttpStatus.CREATED,
108         message: 'Creado',
109         error: 'Created',
110       },
111       HttpStatus.CREATED,
112     );
113   } else {
114     throw new HttpException(
115       {
116         status: HttpStatus.INTERNAL_SERVER_ERROR,
117         message: 'Internal Server Error',
118         error: 'Internal Server Error',
119       },
120       HttpStatus.INTERNAL_SERVER_ERROR,
121     );
122   }
123 }
```

Código 2.7. Método para crear una Persona

En el Código 2.8 se presenta la función `@Post` que recepta las peticiones HTTP *POST* (línea 48), luego, se muestra el método `createPersona` (línea 49) de la clase `PersonaController`, de modo que, la función `@Body` extrae el objeto de tipo `CreatePersonaDto` de cada petición (línea 49), y se llama al método `createPersona` de la clase `PersonaService` para crear la Persona (línea 50).

```
48  @Post()
49  async createPersona(@Body() personaDto: CreatePersonaDto): Promise<void> {
50    await this.personaService.createPersona(personaDto);
51  }
```

Código 2.8. Método para receptar peticiones HTTP *POST* de creación

En la carpeta `backend/src/persona` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Persona.

2.6.1.5. Módulo Autenticación

El módulo `AuthenticationModule` posee varios elementos que ayudan a establecer el proceso de autenticación basado en roles. A continuación, se menciona los elementos `AuthenticationService` y `AuthenticationController` para iniciar sesión en la aplicación.

En los Código 2.9 y Código 2.10 se presenta el método `login` (línea 16) de la clase `AuthenticationService`, por tanto, se extrae el `nombreUsuario` del `loginDto` (línea 17), se busca en la base de datos por nombre de usuario o correo electrónico y que la Persona esté en estado activo (true) (líneas 18-30); se valida que la persona exista, por lo que, se usa la clase `HttpException` para el manejo de errores (líneas 31-40). Después, con el método `compare` se comprueba que la contraseña ingresada sea igual a la contraseña guardada en la base de datos (línea 41), se valida la respuesta del método `compare` (líneas 42-51), luego, se crea un objeto plano con la información obtenida de la base de datos (línea 52-59) y se genera el *token* con el método `sing` (línea 60), para terminar, se retorna el *token* (línea 61).

```
16  async login(loginDto: LoginDto): Promise<any> {
17    const { nombreUsuario } = loginDto;
18    const persona = await this.personaRepository
19      .createQueryBuilder('persona')
20      .leftJoinAndSelect('persona.rolEntity', 'rolEntity')
21      .where('persona.email = :email', {
22        email: nombreUsuario,
23      })
24      .andWhere('persona.estado = :estado', { estado: true })
25      .orWhere('persona.nombreUsuario  = :nombreUsuario', {
26        nombreUsuario: nombreUsuario,
27      })
28      .andWhere('persona.estado = :estado', { estado: true })
29      .addSelect('persona.password')
30      .getOne();
31    if (!persona || persona == undefined) {
32      throw new HttpException(
33        {
34          status: HttpStatus.UNAUTHORIZED,
35          message: 'Usuario/Mail o contraseña Incorrecta',
36          error: 'Unauthorized',
37        },
38        HttpStatus.UNAUTHORIZED,
39      );
40    }
41    const validatePassword = await compare(loginDto.password, persona.password);
42    if (!validatePassword) {
43      throw new HttpException(
44        {
45          status: HttpStatus.UNAUTHORIZED,
46          message: 'Usuario/Mail o contraseña Incorrecta',
47          error: 'Unauthorized',
48        },
49        HttpStatus.UNAUTHORIZED,
50      );
51    }
```

Código 2.9. Método para iniciar sesión (1/2)

```
52      const payload: IPayload = {
53        id: persona.id,
54        nombre: persona.nombre,
55        apellido: persona.apellido,
56        nombreUsuario: persona.nombreUsuario,
57        email: persona.email,
58        rol: persona.rolEntity.nombreRol,
59      };
60      const token: string = this.jwtService.sign(payload);
61      return { token };
62    }
```

Código 2.10. Método para iniciar sesión (2/2)

En el Código 2.11 se muestra la función `@Post` que acepta las peticiones HTTP *POST* (línea 21), luego, se muestra el método `login` (línea 22) de la clase `AuthenticationController`, de modo que, la función `@Body` extrae el objeto de tipo `LoginDto` de cada petición (línea 22), y se llama al método `login` de la clase `AuthenticationService` para realizar la autenticación (línea 23).

```
21    @Post('login')
22    async login(@Body() loginDto: LoginDto): Promise<any> {
23      return this.authService.login(loginDto);
24    }
```

Código 2.11. Método para recibir peticiones HTTP *POST* de autenticación

En la carpeta `backend/src/authentication` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Autenticación.

2.6.1.6. Módulo Equipo

`EquipoModule` es el módulo raíz de `ApModule` y `StaModule`, posee varios elementos que ayudan a gestionar equipos (APs y STAs). A continuación, se menciona un elemento para validar la dirección IP.

```
11  export class ValidationIpPipe implements PipeTransform {
12    transform(value: string, metadata: ArgumentMetadata): string {
13      let validarIp = isIP(value);
14      if (validarIp == false) {
15        throw new HttpException(
16          {
17            status: HttpStatus.BAD_REQUEST,
18            message: 'Error',
19            error: 'Bad Request',
20          },
21          HttpStatus.BAD_REQUEST,
22        );
23      } else {
24        return String(value);
25      }
26    }
27  }
```

Código 2.12. Validación de una dirección de IP

En el Código 2.12 se presenta la clase llamada `ValidationIpPipe` la cual fue generada mediante el CLI de NestJS con la siguiente sintaxis `nest generate pipe validation-pipe`. Después, se procedió a implementar la funcionalidad en el método `transform`, de manera que, se pasa `value` a la función `isIP` (línea 13) proveída por class-validator [39]

(línea 13), por último, si el valor retornado por la función `isIP` es falso se lanza una excepción con la ayuda de la clase `HttpException` (líneas 14-22), caso contrario, se retorna la dirección de IP (línea 24) .

En la carpeta `backend/src/equipos` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Equipo.

2.6.1.7. Módulos AP y STA

`ApModule` y `StaModule` son submódulos del módulo `EquipoModule`, por lo que, los procesos de leer, crear, editar, eliminar o consultar se asigna a cada submódulo según corresponda. A continuación, se presenta los códigos para realizar una consulta SNMP al AP y el proceso de eliminación de una STA.

Se debe agregar que, para implementar las consultas SNMP se utilizó net-snmp [3].

```
113  async queryApSnmp(ip: string): Promise<CreateApDto> {
114    return await new Promise(async (resolve, reject) => {
115      let options = await this.snmpConfiguration();
116      let session = snmp.createSession(ip, ConfigSnmpEnum.COMUNIDAD, options);
117      let dataApSnmp = [];
118      dataApSnmp.push(ip);
119      let oids = [
120        ApOidEnum.NOMBRE,
121        ApOidEnum.MAC,
122        ApOidEnum.MARCA,
123        ApOidEnum.SSID,
124        ApOidEnum.FRECUENCIA,
125        ApOidEnum.ANCHO_CANAL,
126        ApOidEnum.NUM_CLIENTE,
127      ];
128      session.getNext(oids, function(error, varbinds) {
129        if (error) {
130          reject(
131            new HttpException(
132              {
133                status: HttpStatus.NOT_FOUND,
134                message: 'Not Found',
135                error: 'Not Found',
136              },
137              HttpStatus.NOT_FOUND,
138            ),
139          );
140        } else {
```

Código 2.13. Consulta SNMP (1/2)

En los códigos Código 2.13 y Código 2.14 se presenta el método `queryApSnmp` (línea 113), en primer lugar, se instancia y se retorna una promesa (línea 114), en segundo lugar, se utiliza las funciones de la librería [3] para crear la consulta SNMP (líneas 115-116), se crea un array auxiliar (línea 117), se agrega la dirección de IP al array auxiliar (línea 118), se crea un array con las OIDs a consultar (líneas 119-127), después, se llama al método `getNext` proveído por la librería [3] para que realice la consulta SNMP (línea 128), luego, se procede a validar si hubo errores en el proceso de la consulta SNMP con la ayuda de la clase `HttpException` (líneas 129-139), caso contrario, se realiza un filtrado de los datos

obtenidos en la consulta SNMP (líneas 140-164), finalmente, se cierra la sesión con el método `close` proporcionado por la librería [3] (línea 165).

```
140         } else {
141           for (var i = 0; i < varbinds.length; i++) {
142             if (snmp.isVarbindError(varbinds[i])) {
143               console.error(snmp.varbindError(varbinds[i]));
144             } else {
145               if (i == 1) {
146                 let stringMac = Buffer.from(varbinds[i].value).toString('hex');
147                 let formatMac = stringMac.match(/.{1,2}/g).join(':');
148                 dataApSnmp.push(formatMac);
149               } else {
150                 dataApSnmp.push(String(varbinds[i].value));
151               }
152             }
153           }
154           let dtoAp = new CreateApDto();
155           dtoAp.ip = dataApSnmp[0];
156           dtoAp.nombre = dataApSnmp[1];
157           dtoAp.mac = dataApSnmp[2];
158           dtoAp.marca = dataApSnmp[3];
159           dtoAp.ssid = dataApSnmp[4];
160           dtoAp.frecuencia = Number(dataApSnmp[5]);
161           dtoAp.anchoCanal = Number(dataApSnmp[6]);
162           dtoAp.numCliente = Number(dataApSnmp[7]);
163           resolve(dtoAp);
164         }
165         session.close();
166       });
167     });
168   }
```

Código 2.14. Consulta SNMP (2/2)

En el Código 2.15 se muestra la función `@Delete` que acepta las peticiones HTTP *DELETE* (línea 48), luego, se muestra el método `deleteSta` (línea 49) de la clase `StaController`, de manera que, la función `@Param` extrae la dirección IP de cada petición y llama a la clase `ValidationIpPipe` para validar la dirección IP (línea 50), para culminar, se elimina la STA llamando al método `deleteSta` de la clase `StaService` (línea 52).

```
48  @Delete(':ip')
49  async deleteSta(
50    @Param('ip', new ValidationIpPipe()) ip: string,
51  ): Promise<void> {
52    await this.staService.deleteSta(ip);
53  }
```

Código 2.15. Método para recibir las peticiones HTTP *DELETE*

En el Código 2.16 se presenta el método `deleteSta` (línea 212) de la clase `StaService`, así que, se procede a buscar la STA por dirección de IP en la base de datos (línea 213), luego, se valida si existe la STA (líneas 214-223), después, se elimina la STA (línea 224), si en el proceso de eliminado no hubo errores se emite un HTTP *OK*, caso contrario, se emite un HTTP *Internal Server Error* (líneas 225-244). Para el manejo de errores se usó la clase `HttpException` (líneas 215, 226 y 235).

```
212   async deleteSta(ip: string): Promise<void> {
213     let sta = await this.staRepository.getStaByIp(ip);
214     if (sta == undefined) {
215       throw new HttpException(
216         {
217           status: HttpStatus.BAD_REQUEST,
218           message: 'Bad Request',
219           error: 'Bad Request',
220         },
221         HttpStatus.BAD_REQUEST,
222       );
223     }
224     let eliminado = await this.staRepository.remove(sta);
225     if (eliminado && eliminado != undefined) {
226       throw new HttpException(
227         {
228           status: HttpStatus.OK,
229           message: 'Ok',
230           error: 'Ok',
231         },
232         HttpStatus.OK,
233       );
234     } else {
235       throw new HttpException(
236         {
237           status: HttpStatus.INTERNAL_SERVER_ERROR,
238           message: 'Internal Server Error',
239           error: 'Internal Server Error',
240         },
241         HttpStatus.INTERNAL_SERVER_ERROR,
242       );
243     }
244   }
```

Código 2.16. Método para eliminar una STA

En las carpetas `backend/src/equipos/ap` y `backend/src/equipos/sta` del ANEXO G se encuentran el resto de los archivos que conforman los módulos AP y STA.

2.6.1.8. Módulo Monitoreo

El módulo `MonitoreoModule` posee varios elementos que ayudan a establecer los procesos de leer, crear, editar o monitorear equipos (APs y STAs). A continuación, se presenta un fragmento de código para obtener información de la señal mediante poleos SNMP.

En los códigos Código 2.17 y Código 2.18 se presenta el método `signal` (línea 65) de la clase `SignalService`, para empezar, se busca en la base de datos los APs en estado *online* (`true`) (línea 66), de forma similar, se busca en la base de datos el parámetro señal (líneas 67-69), enseguida, se valida que el parámetro señal exista, por lo que, se usa la clase `Logger` para el manejo de errores (líneas 70-73). Después, se declara un array especificando que se requiere la dirección de IP de la STA y el parámetro señal (líneas 74-77), luego, se declara un `for` para recorrer el array de APs (línea 78) y se llama al método `monitoreoEquipoSnmpTable` de la clase `MonitoreoService` para obtener la dirección de IP de las STAs y sus valores de señal (líneas 79-82). En segundo lugar, si existen datos sobre STAs se procede iterarlos (línea 84), de modo que, se crean declaran variables auxiliares `estado`, `findTipoAlarma` y `message` (líneas 85-87), también, se busca en la base de datos la STA monitoreada (líneas 88-90), además, se verifica si el

valor de señal de la STA monitoreada supera el límite permitido, así que, si supera el valor se le asigna el estado `false` y una alarma de tipo crítica, caso contrario, se le asigna un estado de `true` y una alarma de `notificación` (líneas 91-104). Finalmente, se actualiza o crea el valor de señal de la STA monitoreada en la base de datos (líneas 105-113) y se emite un evento llamado refrescar para que se actualice la información en el *dashboard* web (líneas 117-120).

```
65  async signal(): Promise<void> {
66    const apIp = await this.monitoreoService.getAllApUp();
67    const findParametroSignal = await this.paramMonitoreoRepository.findOne({
68      where: { parametro: MonitoreoComponononenteEnum.SIGNAL },
69    });
70    if (!findParametroSignal || findParametroSignal == undefined) {
71      this.logger.log('Error BD no existe Registro');
72      return;
73    }
74
75    const columnaSignal = [
76      Number(StaOidTableEnum.COLUMN_IP_STA),
77      Number(StaOidTableEnum.COLUMN_SIGNAL),
78    ];
79    for (const iter in apIp) {
80      const arrayStaMonitoreo = await this.monitoreoService.monitoreoEquipoSnmpTable(
81        apIp[iter].ip,
82        columnaSignal,
83      );
84      if (arrayStaMonitoreo != undefined) {
85        for (const iter2 in arrayStaMonitoreo) {
86          let estado: boolean;
87          let findTipoAlarma: TipoAlarmaEntity;
88          const message = `Valor: ${arrayStaMonitoreo[iter2].datoMonitoreado} dbm, Límite:
89          const findSta = await this.staRepository.getStaByIp(
90            arrayStaMonitoreo[iter2].ip,
91          );
92          if (
93            findParametroSignal.limite >
94            arrayStaMonitoreo[iter2].datoMonitoreado
95          ) {
96            findTipoAlarma = await this.tipoAlarmaRepository.findAlarma(
97              TipoAlarmaNivelEnum.CRITICO,
98            );
99            estado = false;
```

Código 2.17. Método para monitorear la señal (1/2)

```
100           } else {
101             findTipoAlarma = await this.tipoAlarmaRepository.findAlarma(
102               TipoAlarmaNivelEnum.NOTIFICACION,
103             );
104             estado = true;
105           }
106           if (findSta != undefined && findTipoAlarma != undefined) {
107             await this.monitoreoService.createOrUpdateMonitoreoSta(
108               estado,
109               findSta,
110               message,
111               findTipoAlarma,
112               findParametroSignal,
113             );
114           }
115         }
116       }
117     }
118     this.eventoService.disparador({
119       id: EventoAppNumberEnum.REFRESCO,
120       message: EventoAppEnum.REFRESCAR,
121     });
122   }
```

Código 2.18. Método para monitorear la señal (2/2)

En la carpeta `backend/src/monitoreo` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Equipo.

2.6.1.9. Módulo Log

El módulo `LogModule` posee elementos que ayudan a establecer el proceso de recepción y filtrado de *logs*. A continuación, se menciona el elemento `LogService` para filtrar el mensaje *log*.

Se debe añadir que, para implementar la recepción de *logs* se utilizó syslog-server [34].

En el Código 2.19 se presenta el método `downSta` (línea 107) de la clase `LogService`, por consiguiente, se extrae la dirección MAC del mensaje *log* (línea 108) y se busca la STA por dirección MAC (línea 109), luego, si los datos obtenidos son válidos se procede a asignar el nuevo estado (líneas 111-112), se actualiza el nuevo valor de estado de la STA y se válida el proceso de guardado (líneas 113-116), después, se busca un tipo de alarma para el evento STA *offline* y se crea la alarma de STA (líneas 117-124), finalmente, se consulta y se envía los nuevos valores de total de equipos al *Dashboard* web (líneas 125-129).

```
107   async downSta(message: string): Promise<void> {
108     const macSta = await this.extraerMac(message);
109     const sta = await this.findStaByMac(macSta);
110
111     if (macSta && macSta !== undefined && sta && sta !== undefined) {
112       sta.estado = false;
113       const resultado = await this.staRepository.saveSta(sta);
114       if (resultado == -1) {
115         this.logger.warn('Error Save DataBase');
116       }
117       const advertencia = await this.tipoAlarmaRepository.findOne({
118         where: { nivel: TipoAlarmaNivelEnum.ADVERTENCIA },
119       });
120       await this.createAlarmaSta(
121         advertencia,
122         sta,
123         MonitoreoCompononenteEnum.EQUIPO_APAGADO,
124       );
125       const data = await this.equipoService.queryTotalEquipo();
126       await this.eventoService.disparador({
127         id: EventoAppNumberEnum.ACTUALIZACION,
128         message: data,
129       });
130     }
131   }
```

Código 2.19. Fragmento para filtrar el mensaje *log*

En la carpeta `backend/src/log` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Log.

2.6.1.10. Módulo Alarma

El módulo `AlarmaModule` posee varios elementos que ayudan leer, crear y notificar alarmas de equipos (APs y STAs). A continuación, se menciona el elemento

`AlarmaApSubscriber` para escuchar eventos de inserción de alarmas de APs entre la aplicación `backend` y la base de datos.

En el Código 2.20 se presenta el método `afterInsert` (línea 27) de la clase `AlarmaApSubscriber`, por lo cual, se procede a validar si existe la entidad (línea 27), si existe la entidad se emite un evento al *dashboard* web de notificación (Alarma Creada).

```
27  afterInsert(event: InsertEvent<AlarmaApEntity>) {
28    if (event.entity && event.entity != undefined) {
29      this.eventoService.disparador({
30        id: EventoAppNumberEnum.NOTIFICACION,
31        message: EventoAppEnum.ALARMA_SAVE,
32      });
33    }
34  }
```

Código 2.20. Método para escuchar eventos de *insert*

En la carpeta `backend/src/alarma` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Alarma.

2.6.1.11. Módulo Evento

El módulo `EventoModule` posee varios elementos que ayudan a establecer los procesos de emitir, leer o crear eventos de la aplicación `backend`. A continuación, se menciona los elementos `EventoService` y `EventoController` para emitir eventos.

En el Código 2.21 se presenta los métodos `listener` (línea 67) y `disparador` (línea 74) de la clase `EventoService`, de manera que, el método `listener` es el encargado de escuchar (líneas 67-73). El método disparador tiene la responsabilidad de emitir los eventos con la ayuda de la clase `EventEmitter2` [40] (líneas 74-76).

```
67  listener(): Observable<string> {
68    const events: Observable<string> = fromEvent(
69      this.eventEmitter,
70      'eventName',
71    );
72    return events;
73  }
74  async disparador(data: any): Promise<void> {
75    this.eventEmitter.emit('eventName', { data });
76  }
```

Código 2.21. Métodos para emitir y escuchar eventos

En el Código 2.22 se presenta la función `@sse` que envía eventos por la ruta especificada (línea 13), luego, se muestra el método `sse` de la clase `EventoController`, por lo cual, llama al método `listener` de la clase `EventoService` para enviar los eventos del servidor al *frontend* (líneas 14-16).

```
13  @Sse('sse/event')
14  sse(): Observable<string> {
15    return this.eventoService.listener();
16  }
```

Código 2.22. Método para enviar los eventos

En la carpeta `backend/src/evento` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Evento.

2.6.1.12. Módulo Reporte

El módulo `ReporteModule` posee varios elementos que ayudan a establecer el proceso de generar reportes semanales de equipos (APs y STAs). A continuación, se menciona el elemento `ReporteService` para realizar la tarea de generar reportes.

En el Código 2.23 se presenta el método `cronReporteSemanal` de la clase `ReporteService` (línea 40), así que, se busca en la base de datos el registro que indique cuando se debe realizar la tarea de reporte semanal (líneas 41-45), después, se valida si existe el registro (línea 46), si existe el registro se instancia a la clase `CronJob` (línea 47) y se le pasa como argumentos de entrada el tiempo en formato CRON (línea 48) y el método que debe ejecutarse (línea 50), luego, se procede a registrar la tarea con la ayuda de la clase `SchedulerRegistry` (líneas 53-56) y por último se inicia la tarea programada (línea 57).

```
40  async cronReporteSemanal(): Promise<void> {
41    let timeReporteSemanal = await this.configReportRepository.findOne({
42      where: {
43        tiempoCron: CronReporteConfiguracionEnum.CRON_SEMANAL,
44      },
45    });
46    if (timeReporteSemanal && timeReporteSemanal != undefined) {
47      const jobReporteSemanal = new CronJob(
48        timeReporteSemanal.tiempoCron,
49        () => {
50          this.reporteSemanal();
51        },
52      );
53      this.schedulerRegistry.addCronJob(
54        ReporteEnum.REPORTE_SEMANAL,
55        jobReporteSemanal,
56      );
57      jobReporteSemanal.start();
58    }
59  }
```

Código 2.23. Método para crear una tarea programada

En la carpeta `backend/src/reporte` del ANEXO G se encuentran el resto de los archivos que conforman el módulo Reporte.

2.6.1.13. Creación de *Triggers*

A continuación, se presenta dos *triggers* implementados en MySQL para escuchar el evento de insertar un nuevo registro en la tabla `ap` y el evento de eliminar un registro de la tabla `sta`.

En el Código 2.24 se presenta la sintaxis para crear el *trigger* `after_insert_ap` (línea 8), por lo cual, se especifica que después de insertar en la tabla `ap` (línea 9), por cada registro ingresado (línea 10), se guarda en la tabla `evento_app` un evento llamado Equipo Creado con el mensaje "AP; IP: <IP ingresada>; modelo: <modelo ingresado>" y la fecha (`now()`) (líneas 12-13).

```
8  CREATE TRIGGER after_insert_ap
9  AFTER INSERT ON ap
10 FOR EACH ROW
11 BEGIN
12   INSERT INTO evento_app (nombreEvento,mensaje,creado)
13   VALUES ('Equipo Creado',CONCAT('AP;',' IP: ',NEW.ip,'; modelo: ',NEW.modelo),now());
14 END$$
15 DELIMITER ;
```

Código 2.24. *Trigger* `after_insert_ap`

En el Código 2.25 se presenta la sintaxis para crear el *trigger* `after_delete_sta` (línea 88), por tanto, se indica que después de eliminar en la tabla `sta` (línea 89), por cada registro eliminado (línea 90), se crea una variable auxiliar con el número total de clientes (STAs) que están guardados en la tabla `sta` (línea 92), se actualiza el número total de clientes en la tabla `ap` (línea 93), se guarda en la tabla `evento_app` un evento llamado Equipo Eliminado con el mensaje "STA; IP: <IP eliminada>; nombre: <nombre eliminado>" y la fecha (`now()`) (líneas 94-95).

```
88 CREATE TRIGGER after_delete_sta
89 AFTER DELETE ON sta
90 FOR EACH ROW
91 BEGIN
92   SET @numCliente := (SELECT count(*) FROM sta WHERE apId=OLD.apId);
93   UPDATE ap SET numCliente=@numCliente WHERE id=OLD.apId;
94   INSERT INTO evento_app (nombreEvento,mensaje,creado)
95   VALUES ('Equipo Eliminado',CONCAT('STA;',' IP: ',OLD.ip,'; nombre: ',OLD.nombre),now());
96 END$$
97 DELIMITER ;
```

Código 2.25. *Trigger* `after_delete_ap`

En el ANEXO H se encuentra el *script* completo que genera los *triggers* en la base de datos.

2.6.2. APLICACIÓN *FRONTEND*

Para la implementación del *frontend* se utilizó el *framework* Angular [2], por consiguiente, se usó el comando `$ ng new frontend` del CLI de Angular para generar el proyecto. (Ver Figura 2.16)

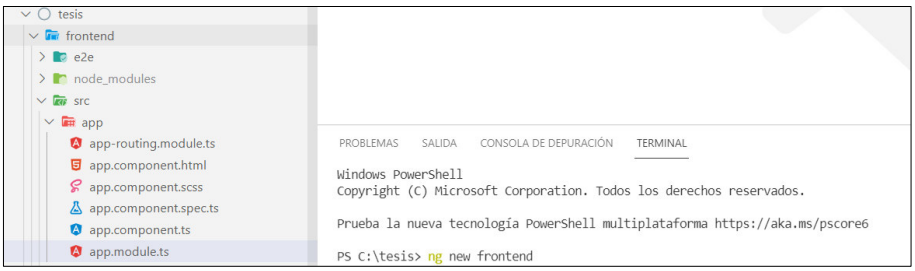

Figura 2.16. Creación del proyecto `frontend`

Por otra parte, para obtener los componentes de Angular Material [31] se debe ejecutar el siguiente comando `$ ng add @angular/material`.

2.6.2.1. Repositorio

Para la generación del repositorio `frontend` se procedió a crear un repositorio en GitHub. (Ver Figura 2.17)

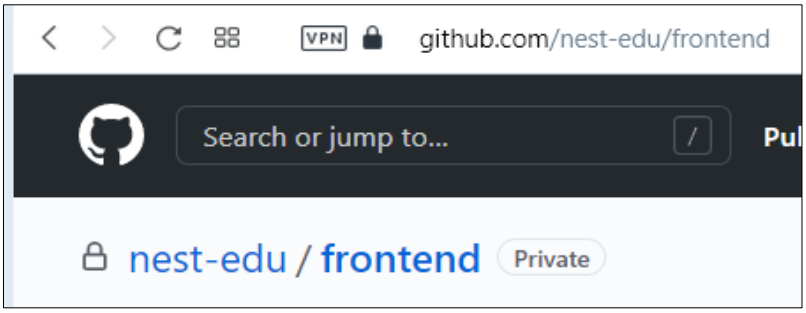

Figura 2.17. Repositorio `frontend`

Los comandos para unir la aplicación `frontend` local con el repositorio remoto `frontend` son `$ git init` y `$ git remote set-url origin https://nest-edu:TOKEN@github.com/nest-edu/frontend.git` . La palabra TOKEN representa el *token* generado por la cuenta personal para la administración de los repositorios [38].

2.6.2.2. Componente Inicio de Sesión

Para que la Persona pueda interactuar con la aplicación web se crea la interfaz inicio de sesión. A continuación, se muestra fragmentos de códigos para realizar el proceso de inicio de sesión.

En el Código 2.26 se presentan dos elementos `input` [41] que permite al usuario ingresar el nombre de usuario o correo electrónico (líneas 22-32) y la contraseña (líneas 34-45).

```
22          <mat-form-field appearance="outline" class="login-form">
23            <mat-label>Usuario/Email</mat-label>
24            <input
25              placeholder="Usuario"
26              formControlName="nombreUsuario"
27              matInput
28              required
29              type="text"
30            />
31            <mat-icon matSuffix>sentiment_very_satisfied</mat-icon>
32          </mat-form-field>
33
34          <mat-form-field appearance="outline" class="login-form">
35            <mat-label>Password</mat-label>
36
37            <input
38              placeholder="Password"
39              formControlName="password"
40              matInput
41              required
42              type="password"
43            />
44            <mat-icon matSuffix>vpn_key</mat-icon>
45          </mat-form-field>
```

Código 2.26. Ingreso de datos

En el Código 2.27 se presenta el método `login` (línea 37) que permite obtener los datos ingresados por la Persona, de manera que, se evita que haya un *refresh* de la página (línea 38), después, se valida el formulario (línea 40), si el formulario es válido se procede llamar al método `auth` para que realice la petición al `backend` (línea 41), para terminar, si los datos ingresados por la Persona fueron correctos (línea 42) se muestra una animación de bienvenida (línea 43), se guarda el token (línea 44) y se muestra la interfaz del *dashboard* web (línea 46), caso contrario, se muestra una animación indicando el error (línea 49).

Para las animaciones de bienvenida y error se usa la librería ngx-toastr [42].

```
37  login(event: Event) {
38    event.preventDefault();
39
40    if (this.frmLogin.valid) {
41      this.authService.auth(this.frmLogin.value).subscribe(
42        (res) => {
43          this.toastr.success(MensajeVariosEnum.BIENVENIDO, res.error);
44          this.tokenService.setToken(res.token);
45
46          this.router.navigate(['/admin']);
47        },
48        (err) => {
49          this.toastr.error(err.error.message, err.error.error, {});
50        }
51      );
52    }
53  }
```

Código 2.27. Método para obtener los valores del formulario

En el Código 2.28 se presenta el método `auth` (línea 17) que permite crear una petición HTTP *POST* con la ayuda de la clase `HttpClient` para enviar el nombre de usuario o email y la contraseña al `backend` (líneas 18-21).

```
17  public auth(persona: LoginModel): Observable<any> {
18    const authPersona = this.httpClient.post<any>(
19      `${environment.url_api}/auth/login`,
20      persona
21    );
22    return authPersona;
23  }
```

Código 2.28. Creación petición HTTP *POST*

Finalmente, en la Figura 2.18 se presenta la interfaz de inicio de sesión.

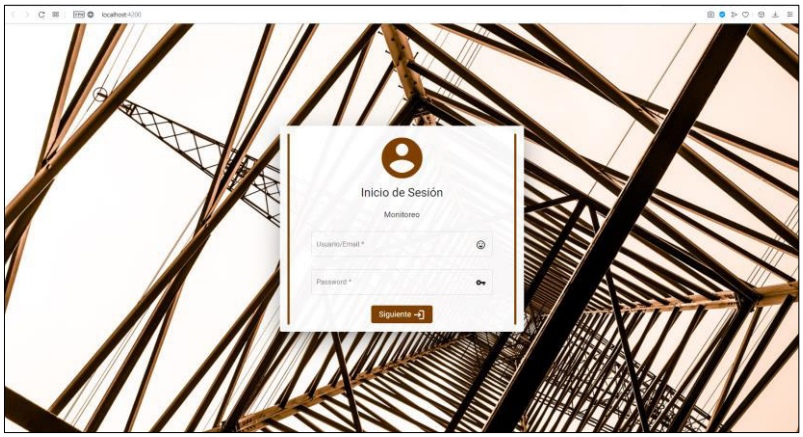

Figura 2.18. Interfaz de inicio de sesión

En la carpeta `frotend/src/app/auth` del ANEXO I se encuentran el resto de los archivos que conforman el componente Inicio de Sesión.

2.6.2.3. Componente *Dashboard*

Una vez que la Persona inicio sesión de forma correcta se le indica la interfaz *dashboard*. A continuación, se muestra fragmentos de códigos para ver los totales de Equipos.

En el Código 2.29 presenta la `card` [43] que permite mostrar el total de APs, por lo que, se especifican características (líneas 5-7), si se hace *click* en la `card` se muestra la interfaz de APs (línea 8), luego, se tiene el diseño para mostrar el título, icono y valor (líneas 10-24).

```
4   <mat-card
5     matRipple
6     [matRippleColor]="color"
7     class="mat-elevation-z8 contorno-card"
8     (click)="navigateEquipoAp()"
9   >
10    <div class="vertical-linea-verde">
11      <div class="contenido-card">
12        <div class="superior-card">
13          <div>
14            <b> Total APs </b>
15          </div>
16        </div>
17        <div class="inferior-card">
18          <div>
19            <mat-icon class="icons-card-verde"> sensors </mat-icon>
20          </div>
21          <div *ngIf="dataDashboard">{{ dataDashboard.totalAp }}</div>
22        </div>
23      </div>
24    </div>
25  </mat-card>
```

Código 2.29. *Card* total de APs

En el Código 2.30 se presenta el método `getDataDashboard` (línea 124) que permite obtener los totales de Equipos, así que, se llama al método `getDataDashboard` de la clase `DashboardService` para que realice la petición al `backend` (línea 125), si la petición fue correcta pasa los valores para que sean mostrados y dibujados (líneas 126-136), caso contrario, se muestra una animación indicando el error (línea 137-140).

Para realizar las gráficas se usa la librería ng2-charts [44].

```
124  getDataDashboard() {
125    this.dashboardService.getDataDashboard().subscribe(
126      (auxData) => {
127        this.dataDashboard = auxData;
128        this.graficaAp(
129          this.dataDashboard.totalApUp,
130          this.dataDashboard.totalApDown
131        );
132        this.graficaSta(
133          this.dataDashboard.totalStaUp,
134          this.dataDashboard.totalStaDown
135        );
136      },
137      (err) => {
138        this.toastr.error(err.error.error, MensajeEquipoEnum.ERROR, {});
139      }
140    );
```

Código 2.30. Método para obtener los totales de Equipos

En el Código 2.31 se presenta el método `getDataDashboard` (línea 38) que permite crear una petición HTTP *GET* con la ayuda de la clase `HttpClient` para obtener los totales de equipos del `backend` (líneas 39-41).

```
38  public getDataDashboard(): Observable<ResumenEquipoDto> {
39    return this.httpClient.get<ResumenEquipoDto>(
40      `${environment.url_api}/equipo/totalEquipo`
41    );
42  }
43
```

Código 2.31. Creación petición HTTP *GET*

Finalmente, en la Figura 2.19 se presenta la interfaz *dashboard*.

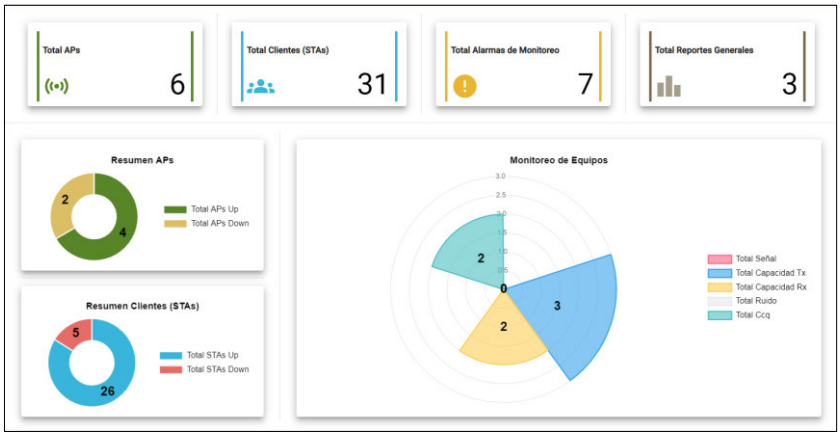

Figura 2.19. Interfaz *dashboard*

En la carpeta `frotend/src/admin/dashboard` del ANEXO I se encuentran el resto de los archivos que conforman el componente Dashboard.

2.6.2.4. Componente Equipo

El Componente Equipo está constituido por varios elementos que permiten generar las interfaces de APs y STAs. A continuación, se proporciona fragmentos de código para mostrar los APs y de cómo eliminar una STA.

En el Código 2.32 se presenta la `table` [45] que permite mostrar la información obtenida de los APs, de modo que, se especifica el array que contiene los datos de APs (línea 25), luego, se indica que la primera columna de la tabla corresponde a `id` (línea 26), después, se ingresa un nombre para la columna (línea 27) y se itera los datos de la columna `id` (línea 28). Para las siguientes columnas se realiza el mismo procedimiento que se detalló anteriormente.

```
25  <table mat-table [dataSource]="dataSourceAp" class="aps-table">
26    <ng-container matColumnDef="id">
27      <th mat-header-cell *matHeaderCellDef class="cabecera-tabla">ID</th>
28      <td mat-cell *matCellDef="let ap; let i = index">{{ i + 1 }}</td>
29    </ng-container>
30    <ng-container matColumnDef="ip">
31      <th mat-header-cell *matHeaderCellDef class="cabecera-tabla">IP</th>
32      <td mat-cell *matCellDef="let ap">{{ ap.ip }}</td>
33    </ng-container>
```

Código 2.32. Tabla con información de APs

Por otro parte, para obtener los datos de APs y crear la petición HTTP *GET*, se lo realiza de forma similar a lo expuesto en los códigos Código 2.30 y Código 2.31.

En la Figura 2.20 se presenta una parte de la tabla de APs.

Figura 2.20. Tabla de APs

En el Código 2.33 se presenta el método `deleteSta` (línea 121) que permite eliminar una STA, de modo que, se extrae la dirección de IP de la STA (línea 122), luego, se personaliza la animación de confirmación de eliminación (líneas 123-130), se valida la respuesta de confirmación (línea 131), si la respuesta es afirmativa se busca el índice del *array* donde se encuentra la STA (línea 132), después, se valida el índice (línea 133), si el índice es válido se llama al método `deleteSta` de la clase `EquipoService` para que realice la petición al `backend` (línea 134); si la petición fue correcta se muestra una animación satisfactoria, se elimina la STA del *array* y se refresca la tabla (líneas 135-142), caso contrario, se muestra una animación indicando el error (línea 143-145).

Para la animación de eliminar se usa la librería sweetalert2 [46].

```
121  deleteSta(sta: StaDto) {
122    const { ip } = sta;
123    Swal.fire({
124      title: SwalDeleteEnum.TITLE,
125      text: SwalDeleteEnum.TEXT,
126      icon: SwalDeleteEnum.ICON,
127      showCancelButton: true,
128      confirmButtonText: SwalDeleteEnum.CONFIRM,
129      cancelButtonText: SwalDeleteEnum.CANCEL,
130    }).then((result) => {
131      if (result.value) {
132        const findSta = this.listSta.findIndex((aux) => aux.ip == sta.ip);
133        if (findSta !== -1) {
134          this.equipoService.deleteSta(ip).subscribe(
135            (res) => {
136              this.toastr.success(
137                MensajeEquipoEnum.EQUIPO_DELETE,
138                MensajeEquipoEnum.OK
139              );
140              this.listSta.splice(findSta, 1);
141              this.dataSource = new MatTableDataSource(this.listSta);
142            },
143            (err) => {
144              this.toastr.error(err.error.error, MensajeEquipoEnum.ERROR, {});
145            }
146          );
147        }
148      }
149    });
150  }
```

Código 2.33. Método para eliminar una STA

En el Código 2.34 se presenta el método `deleteSta` (línea 84) que permite crear una petición HTTP *DELETE* con la ayuda de la clase `HttpClient` para eliminar la STA del `backend` (línea 85).

```
84  public deleteSta(ip: string) {
85    return this.httpClient.delete(`${environment.url_api}/equipo/sta/${ip}`);
86  }
```

Código 2.34. Creación petición HTTP *DELETE*

Finalmente, en la Figura 2.21 se presenta la animación para eliminar una STA.

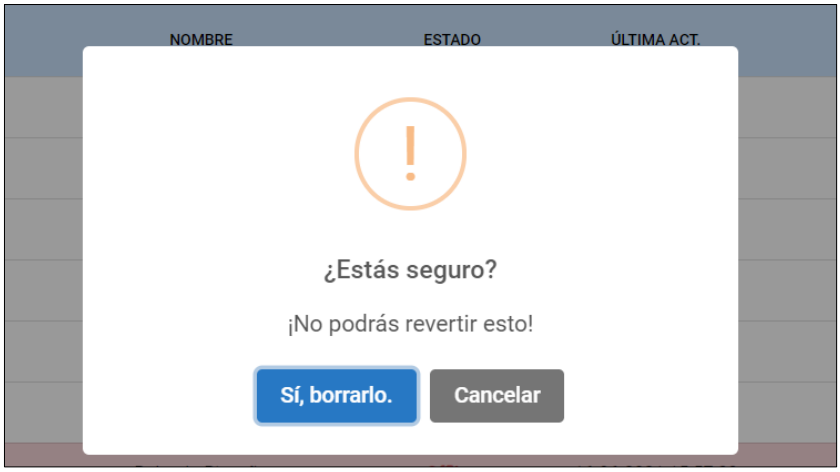

Figura 2.21. Mensaje para eliminar una STA

En la carpeta `frotend/src/admin/equipo` del ANEXO I se encuentran el resto de los archivos que conforman el componente Equipo.

2.6.2.5. Componente Configuración

El Componente Configuración está constituido por varios elementos que permiten generar las interfaces para configurar los parámetros de monitoreo. A continuación, se muestra fragmentos de código para activar o desactivar una tarea programada.

En el Código 2.35 se presenta la columna `estado` (línea 24), por consiguiente, se especifica el nombre de la columna (línea 25), luego, se itera la columna estado, si el `estado` corresponde a `true` se muestra un botón en forma de icono que representa que la tarea está activa (líneas 27-36), caso contrario, si el `estado` es `false` se muestra un botón en forma de icono que representa que la tarea está desactivada (líneas 38-47).

```
24      <ng-container matColumnDef="estado">
25        <th mat-header-cell *matHeaderCellDef class="cabecera-tabla">ESTADO</th>
26        <td mat-cell *matCellDef="let job">
27          <div *ngIf="job.estado === true">
28            <button
29              mat-icon-button
30              class="btn btn-outline-success"
31              matTooltip="Desactivar"
32              (click)="changeEstado(job)"
33            >
34              <mat-icon>toggle_on</mat-icon>
35            </button>
36          </div>
37
38          <div *ngIf="job.estado === false">
39            <button
40              mat-icon-button
41              class="btn btn-outline-danger"
42              matTooltip="Activar"
43              (click)="changeEstado(job)"
44            >
45              <mat-icon>toggle_off</mat-icon>
46            </button>
47          </div>
48        </td>
49      </ng-container>
```

Código 2.35. Columna con animación de activar o desactivar

En el Código 2.36 se presenta el método `changeEstado` (línea 57) que permite cambiar el estado de la tarea programada, de manera que, se personaliza la animación [46] de confirmación para detener o activar la tarea (líneas 58-64), luego, se valida la respuesta (línea 65), si la respuesta es afirmativa se instancia a la clase `EditCronJobDto`, se le asigna el nuevo `estado`, después, se llama al método `editTareaEstado` de la clase `MonitoreoService` para que realice la petición (línea 70), si el proceso fue correcto se refresca los valores de la tabla (línea 71-73), caso contrario, se muestra una animación del error obtenido (línea 74-77).

```
57  changeEstado(tarea: CronJobDto) {
58    Swal.fire({
59      title: SwalTareaChangeEstadoEnum.TITLE,
60      icon: SwalTareaChangeEstadoEnum.ICON,
61      showCancelButton: true,
62      confirmButtonText: SwalTareaChangeEstadoEnum.CONFIRM,
63      cancelButtonText: SwalTareaChangeEstadoEnum.CANCEL,
64    }).then((result) => {
65      if (result.value && tarea) {
66        let editCronTarea = new EditCronJobDto();
67        editCronTarea.nombreTarea = tarea.nombre;
68        editCronTarea.estado = !tarea.estado;
69
70        this.monitoreo.editTareaEstado(editCronTarea).subscribe(
71          (res) => {
72            this.getCronJob();
73          },
74          (err) => {
75            this.toastr.error(err.error.error, MensajeEquipoEnum.ERROR, {});
76          }
77        );
78      }
79    });
80  }
```

Código 2.36. Método para editar el estado del parámetro

En el Código 2.37 se presenta el método `editTareaEstado` (línea 67) que permite crear una petición HTTP *PUT* con la ayuda de la clase `HttpClient` para editar el estado de una tarea programada en el `backend` (líneas 68-71).

```
67  public editTareaEstado(changeEstado: Partial<EditCronJobDto>) {
68    return this.httpClient.put<any>(
69      `${environment.url_api}/monitoreo/editCron`,
70      changeEstado
71    );
72  }
```

Código 2.37. Creación petición HTTP *PUT*

Finalmente, en la Figura 2.22 se presenta la interfaz de configuración.

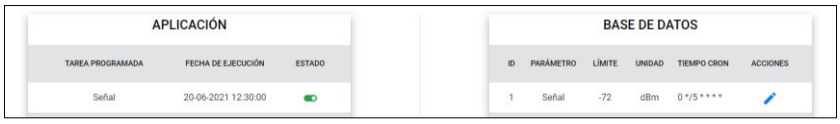

Figura 2.22. Interfaz para activar o desactivar una tarea

En la carpeta `frotend/src/admin/configuration` del ANEXO I se encuentran el resto de los archivos que conforman el componente Configuración.

2.6.2.6. Componente Persona

El Componente Persona está constituido por varios elementos que permiten crear las interfaces para gestionar Personas. A continuación, se proporciona fragmentos de código del formulario para crear una Persona.

En el Código 2.38 se presenta un `input` [47] del formulario Persona, por lo que, se aplica un estilo y una etiqueta al campo texto (líneas 17-18), luego, se agregan propiedades al

`input` [47] (líneas 19-25) y se válida que exista un dato ingresado en el campo (líneas 26-30).

```
17                <mat-form-field appearance="outline">
18                  <mat-label>Nombre</mat-label>
19                  <input
20                    placeholder="Nombre"
21                    formControlName="nombre"
22                    matInput
23                    required
24                    type="text"
25                  />
26                  <mat-error
27                    *ngIf="frmPersona.controls['nombre'].hasError('required')"
28                  >
29                    Ingresar un Nombre
30                  </mat-error>
31                </mat-form-field>
```

Código 2.38. Ingresar nombre de Persona

En el Código 2.39 se presenta el método `buildForm` (línea 48) que permite crear el formulario para ingresar los datos de la Persona, por lo que, se crea un grupo de controles (línea 49), luego, se agrega los elementos del formulario (líneas 50-57). Para hacer validaciones se llama a la clase `Validators`, por ejemplo, el campo de email requiere que el valor ingresado sea un correo electrónico (línea 53).

```
48  private buildForm() {
49    this.frmPersona = this.formBuilder.group({
50      nombre: [null, Validators.required],
51      apellido: [null, Validators.required],
52      nombreUsuario: [null, Validators.required],
53      email: [null, Validators.email],
54      password: [null, [Validators.minLength(8)]],
55      estado: new FormControl(null, Validators.required),
56      rol: new FormControl(null, Validators.required),
57      enviarCorreo: new FormControl(null, Validators.required),
58    });
59  }
```

Código 2.39. Método para construir el formulario Persona

Finalmente, en la Figura 2.23 se presenta la interfaz del formulario para crear una Persona.

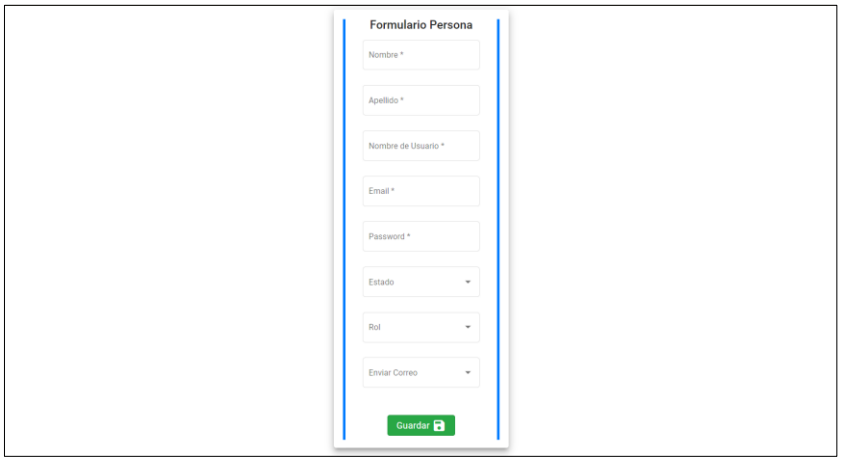

Figura 2.23. Interfaz del formulario para crear una Persona

En la carpeta `frotend/src/admin/persona` del ANEXO I se encuentran el resto de los archivos que conforman el componente Persona.

2.6.2.7. Componente Reporte

El Componente Reporte está constituido por varios elementos que permiten crear las interfaces mostrar y generar los reportes. A continuación, se proporciona un fragmento de código para crear el PDF de un reporte.

Para crear el archivo PDF se usa la librería pdfmake-wrapper [36].

En el Código 2.40 se presenta una parte de la estructura del PDF, así que, se agrega información del archivo PDF (líneas 158-162), luego, se indica el contenido del PDF (líneas 163-174), por ejemplo, al título “Reporte Técnico Semanal” se lo personaliza con un tamaño de fuente de 16, centrado y con negritas (líneas 163-166).

```
158   pdf.info({
159     title: 'Reporte Técnico ST',
160     author: `${nombre} ${apellido}`,
161     subject: 'Semanal',
162   });
163   pdf.add(
164     new Txt('Reporte Técnico Semanal').fontSize(16).alignment('center').bold()
165       .end
166   );
167   pdf.add(pdf.ln(1));
168   pdf.add(new Txt(`${fechaActual}`).alignment('right').end);
169   pdf.add(pdf.ln(1));
170   pdf.add(new Txt('Datos').fontSize(12).alignment('left').bold().end);
171   pdf.add(pdf.ln(1));
172   pdf.add(
173     new Txt(`Generado por: ${nombre} ${apellido}`).alignment('left').end
174   );
```

Código 2.40. Creación del PDF

Finalmente, en la Figura 2.24 se presenta una captura del reporte semanal en formato PDF.

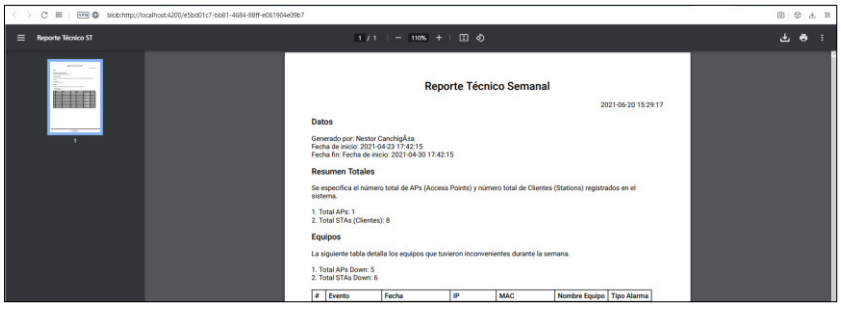

Figura 2.24. Reporte semanal en formato PDF

En la carpeta `frotend/src/admin/reporte` del ANEXO I se encuentran el resto de los archivos que conforman el componente Reporte.

2.7. TABLERO KANBAN ETAPA CUATRO

En la Tabla 2.4 se presenta las actividades de la etapa cuatro, por lo cual, se agregan las tareas que se deben realizarse en la primera columna, se muestran las tareas en proceso en la segunda columna y se indica las tareas realizadas en la tercera columna.

Tabla 2.4. Tablero Kanban etapa cuatro

Tareas	Tareas en proceso	Tareas realizadas
Desplegar el prototipo de aplicación web en la microempresa.	Probar el funcionamiento del `frontend`.	Realizar la primera entrevista al gerente técnico de la microempresa.
		Realizar consultas SNMP de prueba a los equipos (APs y STAs).
		Determinar requerimientos funcionales y no funcionales.
		Revisar características generales de los equipos (APs y STAs).
		Revisar la MIB de la casa fabricante.
		Revisar el lenguaje de programación TypeScript.
		Revisar la documentación de NestJS.
		Realizar una figura de la arquitectura del prototipo.
		Realizar el diagrama entidad relación y relacional de la base de datos.
		Realizar el diagrama de actividades del *backend.*
		Realizar el diagrama de clases del *backend*.
		Realizar la segunda entrevista al gerente técnico de la microempresa y actualizar requerimientos.

Tareas	Tareas en proceso	Tareas realizadas
		Realizar los *mockups.*
		Instalar Visual Studio Code, Node.js, MySQL, Git y Postman.
		Codificar la aplicación `backend`
		Probar el funcionamiento del `backend`.
		Codificar la aplicación `frontend`

2.8. DESPLIEGUE

Para despliegue del prototipo de aplicación web del presente Trabajo de Titulación se utilizó las siguientes herramientas:

- Computadora (Propiedad de la microempresa)
- Sistema Operativo Ubuntu 20.04 LTS
- MySQL versión 8.0
- Node.js versión 12.22.1
- PM2
- Apache versión 2.4.41

Por consiguiente, se crea la base de datos, luego, se construye la aplicación `backend` mediante el comando `$ nest build`, de modo que, en la Figura 2.25 se presenta la generación de la carpeta `dist` que contiene todos los archivos de la aplicación. Después, se procede a ejecutar el archivo `dist/main.js` con el comando `$ sudo pm2 start main.js --name backend`, por tanto, en la Figura 2.26 se muestra la aplicación `backend` desplegada. Por último, se corre el *script* para generar los *triggers*.

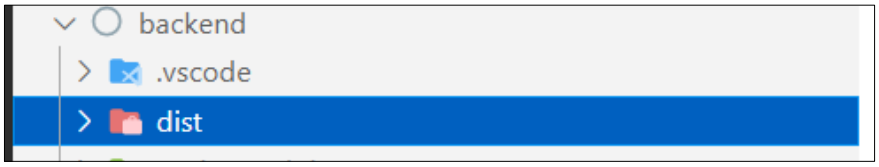

Figura 2.25. Carpeta `backend/dist`

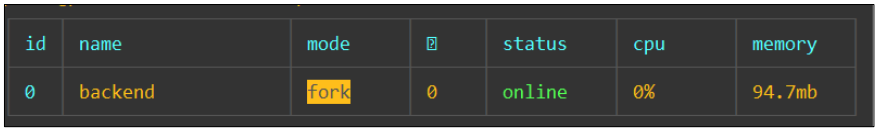

Figura 2.26. Aplicación `backend` desplegada

Para construir la aplicación `frontend` se utiliza el comando `$ ng build --prod`, por lo que, en la Figura 2.27 se presenta la generación de la carpeta `dist/frontend` que contiene todos los archivos de la aplicación. Luego, se procede a copiar los archivos de la carpeta `frontend/dist/frontend` en la carpeta `/var/www/<dominio>` , por lo que, en la Figura 2.28 se muestra la aplicación `frontend` desplegada.

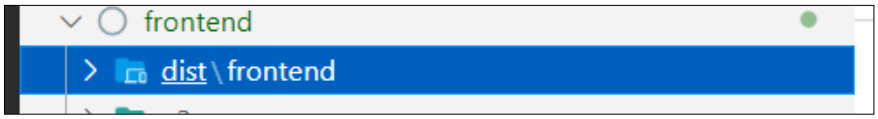

Figura 2.27. Carpeta `frontend/dist/frontend`

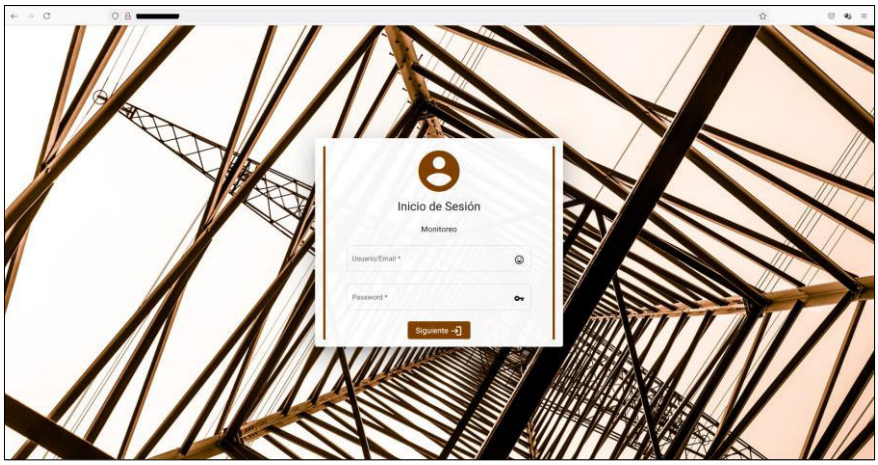

Figura 2.28. Aplicación `frontend` desplegada

3. RESULTADOS Y DISCUSIÓN

Se realizaron 5 encuestas al personal de la microempresa y 5 encuestas a compañeros de la Facultad de Ingeniería Eléctrica y Electrónica para complementar la validación del correcto funcionamiento del prototipo de aplicación web.

3.1. TABLERO KANBAN ACTUALIZACIÓN

En la Tabla 3.1 se presenta las últimas actividades para culminar con el desarrollo del prototipo, por tanto, se agregan las tareas que se deben realizarse en la primera columna y se indica las tareas realizadas en la tercera columna.

Tabla 3.1. Tablero Kanban actualización

Tareas	Tareas en proceso	Tareas realizadas
Realizar las pruebas de funcionamiento de los módulos y componentes de la aplicación.		Realizar la primera entrevista al gerente técnico de la microempresa.
Realizar las encuestas de satisfacción.		Realizar consultas SNMP de prueba a los equipos (APs y STAs).
Realizar las correcciones de los problemas encontrados.		Determinar requerimientos funcionales y no funcionales.
		Revisar características generales de los equipos (APs y STAs).
		Revisar la MIB de la casa fabricante.
		Revisar el lenguaje de programación TypeScript.
		Revisar la documentación de NestJS.
		Realizar una figura de la arquitectura del prototipo.
		Realizar el diagrama entidad relación y relacional de la base de datos.
		Realizar el diagrama de actividades del *backend*.
		Realizar el diagrama de clases del *backend*.

Tareas	Tareas en proceso	Tareas realizadas
		Realizar la segunda entrevista al gerente técnico de la microempresa y actualizar requerimientos.
		Realizar los *mockups.*
		Instalar Visual Studio Code, Node.js, MySQL, Git y Postman.
		Codificar la aplicación `backend`
		Probar el funcionamiento del `backend`.
		Codificar la aplicación `frontend`
		Probar el funcionamiento del `frontend`.
		Desplegar el prototipo de aplicación web en la microempresa.

3.2. PRUEBAS DE FUNCIONAMIENTO

Durante el desarrollo del prototipo de aplicación web se realizaron varias pruebas de funcionamiento con Postman y el navegador Opera, además, se usó un AP y una STA que fueron proporcionados por la microempresa, con el fin de contrarrestar problemas en la fase de implementación.

A continuación, se detalla las pruebas realizadas en un ambiente real, por lo que, direcciones MACs, IPs, URLs, nombres de clientes, nombres de usuarios, contraseñas y correos fueron censuradas. Se utilizaron los siguientes navegadores: Opera, Google Chrome, Mozilla Firefox y Chromium para verificar el funcionamiento del prototipo. Además, la microempresa facilitó el acceso a una subred para realizar las pruebas con 50 STAs (clientes) y 7 APs dando un resultado de 57 equipos en total.

3.2.1. MÓDULO AUTENTICACIÓN Y COMPONENTE INICIO DE SESIÓN

En la Tabla 3.2 se presenta la lista de las pruebas de funcionamiento que se realizaron al Módulo Autenticación mediante el Componente Inicio de Sesión.

Tabla 3.2. Pruebas de funcionamiento Módulo Autenticación y Componente Inicio de Sesión

Pruebas	**Estados**	**Observaciones**
Iniciar sesión con nombre de usuario.	Aprobado	Ninguna
Iniciar sesión con correo electrónico.	Aprobado	Ninguna
Ingresar contraseña incorrecta.	Aprobado	Ninguna
Validar campos vacíos en el formulario.	Aprobado	Ninguna

Como resultado, el módulo autenticación y el componente inicio de sesión pasó las pruebas realizadas, no se obtuvieron observaciones. En la Figura 3.1 se presenta una captura de la prueba de Ingresar contraseña incorrecta.

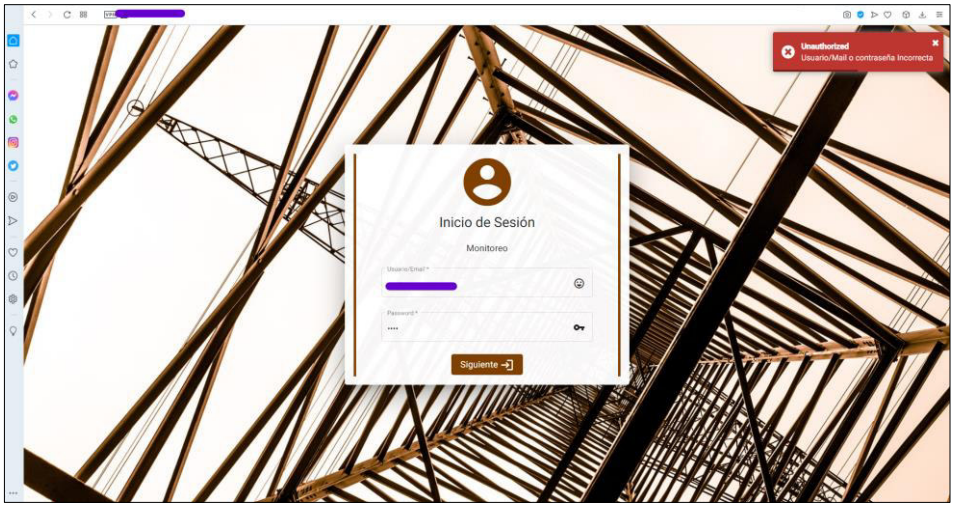

Figura 3.1. Ingresar contraseña incorrecta

3.2.2. MÓDULO PERSONA Y COMPONENTE PERSONA

En la Tabla 3.3 se presenta la lista de las pruebas de funcionamiento que se realizaron al Módulo Persona mediante el Componente Persona.

Tabla 3.3. Pruebas de funcionamiento Módulo Persona y Componente Persona

Prueba	**Estado**	**Observación**
Visualizar las Personas registradas	Aprobado	Ninguna
Crear una Persona con su Rol	Aprobado	Ninguna
Actualizar una Persona	Aprobado	Ninguna
Eliminar una Persona	Aprobado	Ninguna
Validar campos vacíos en el formulario	Aprobado	Ninguna

Como resultado, el Módulo Persona y el Componente Persona pasó las pruebas realizadas, y no se obtuvieron observaciones. En la Figura 3.2 se presenta una captura de la prueba Crear una Persona con su Rol, por lo que, se creó una persona de prueba llamada analista con rol Analista.

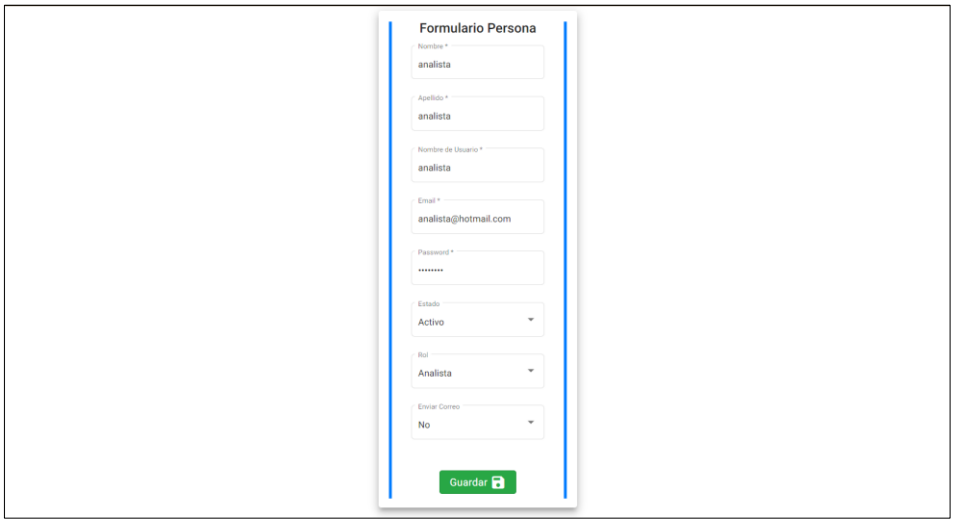

Figura 3.2. Crear una Persona con su Rol

En la Figura 3.3 se presenta una captura de la prueba Visualizar las Personas registradas, de modo que, se puede observar dos Personas válidas (Fila 1 y 2) y dos Personas de prueba (Fila 3 y 4).

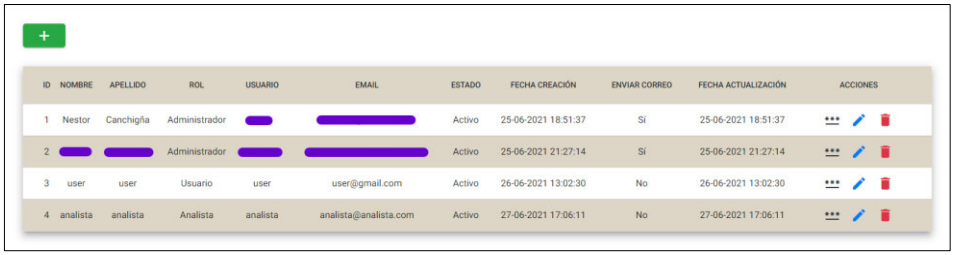

Figura 3.3. Visualizar las Personas registradas

3.2.3. MÓDULO EQUIPO Y COMPONENTE EQUIPO

En la Tabla 3.4 se presenta la lista de las pruebas de funcionamiento que se realizaron al Módulo Equipo mediante el Componente Equipo.

Tabla 3.4. Pruebas de funcionamiento Módulo Equipo y Componente Equipo

Prueba	**Estado**	**Observación**
Visualizar las APs y STAs	Aprobado	Ninguna
Crear un AP y STAs	Aprobado	Cambiar la frase STAs encontrados por STAs encontradas
Eliminar AP y STA	Aprobado	Ninguna
Crear nueva STA automatizada	Aprobado	Ninguna
Validar dirección de IP	Aprobado	Ninguna

Como resultado, el Módulo Equipo y el Componente Equipo pasó las pruebas realizadas, se obtuvo una observación que no implica un mal funcionamiento del prototipo, pero se debe corregir. En las Figura 3.4, Figura 3.5 y Figura 3.6 se presentan capturas de la prueba Crear un AP y STAs, así que, se ingresa la IP del AP, después, se realiza una consulta SNMP al AP para obtener sus datos (Modelo, Nombre, SSID, etc.) y se lo guarda, por último, se obtiene las STAs del AP mediante consultas SNMP al AP y se procede a guardarlas.

Figura 3.4. Ingresar IP de AP

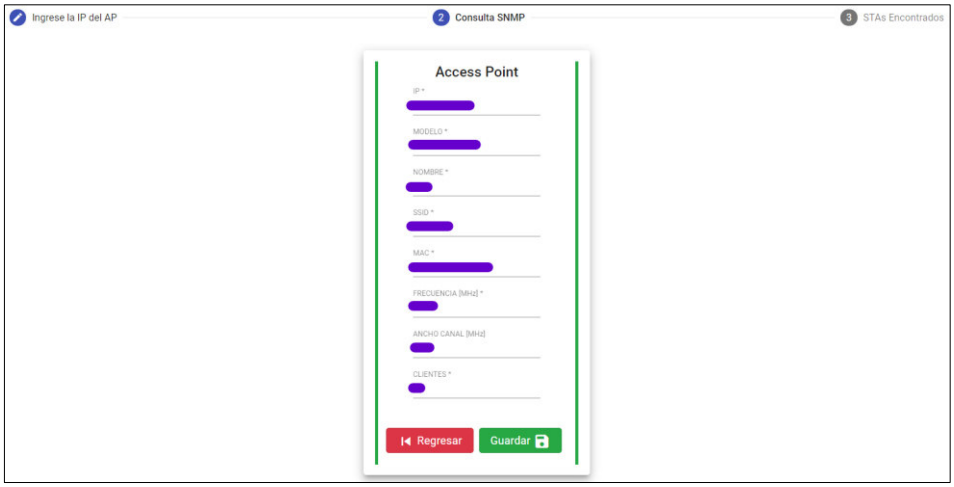

Figura 3.5. AP obtenido mediante una consulta SNMP

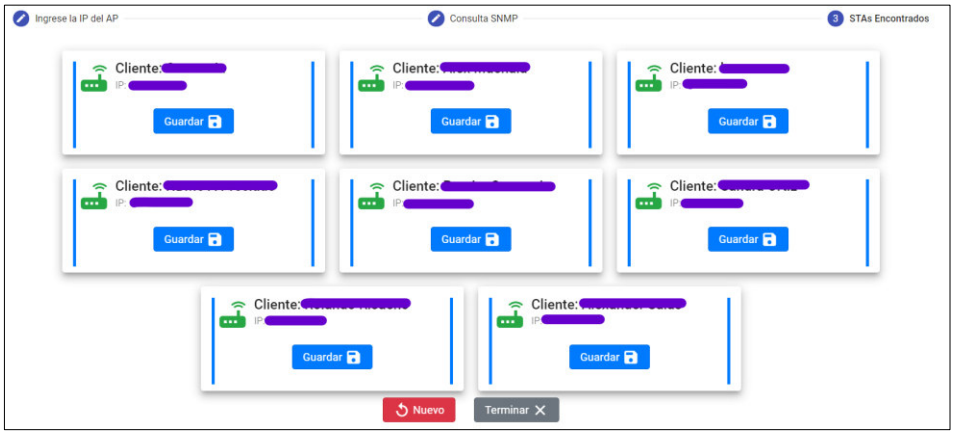

Figura 3.6. STAs encontradas mediante consultas SNMP al AP

3.2.4. MÓDULO MONITOREO Y COMPONENTE REPORTE MONITOREO

En la Tabla 3.5 se presenta la lista de las pruebas de funcionamiento que se realizaron al Módulo Monitoreo mediante el Componente Reporte Monitoreo.

Tabla 3.5. Pruebas de funcionamiento Módulo Monitoreo y Componente Reporte Monitoreo

Prueba	**Estado**	**Observación**
Visualizar los Reportes de Monitoreo	Aprobado	Ninguna
Verificar consultas por parámetro de monitoreo	Aprobado	Ninguna

Como resultado, el Módulo Monitoreo y el Componente Reporte Monitoreo pasó las pruebas realizadas, y no se obtuvo observaciones. En las Figura 3.7 se presenta una captura de la prueba Visualizar los Reportes de Monitoreo, de modo que, se muestra el reporte del parámetro señal.

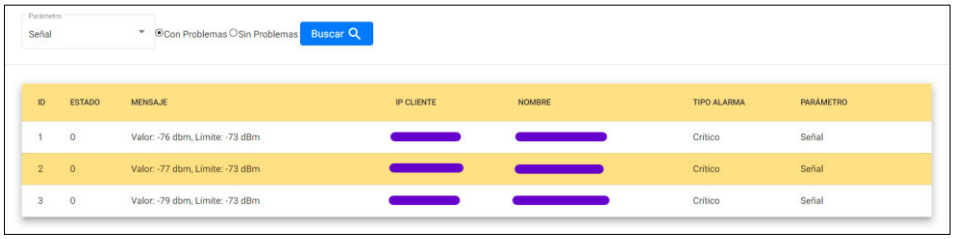

Figura 3.7. Visualizar el reporte del parámetro señal

3.2.5. MÓDULOS Y COMPONENTES EN CONJUNTO

En la Tabla 3.6 se presenta la lista de las pruebas de funcionamiento que se realizaron a los Módulos Log, Alarma y Evento junto con los Componentes Nav, Alarma STA y *Dashboard* .

Tabla 3.6. Pruebas de funcionamiento en conjunto

Prueba	**Estado**	**Observación**
Recepción de *logs*	Aprobado	Revisar filtrado de *logs*.
Asignación de alarmas a APs o STAs según los *logs* receptados	Reprobado	Cuando se procedió a reiniciar un AP se generaban dobles alarmas.

Prueba	Estado	Observación
Emisión de eventos del *backend*	Aprobado	Revisar emisión de eventos
Recepción de eventos en el *frontend*	Aprobado	Revisar recepción de eventos
Actualización de datos del *dashboard*	Aprobado	Cambiar las palabras *Up* y *Down* por *Online* y *Offline* respectivamente.

Como resultado, en la Tabla 3.6 se obtuvo algunas observaciones porque se detectó un problema que hace que la aplicación asigne dos alarmas de forma consecutiva al mismo equipo (Equipo Apagado y Equipo Encendido) (Ver Figura 3.8). Dicho lo anterior, las pruebas realizadas deben ser sometidas a revisión para encontrar el error.

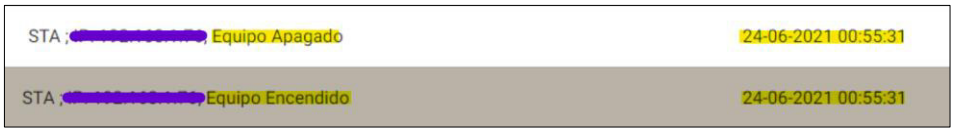

Figura 3.8. Problema detectado

3.2.6. MÓDULO REPORTE Y COMPONENTE REPORTE GENERAL

En la Tabla 3.7 se presenta la lista de las pruebas de funcionamiento que se realizaron al Módulo Reporte mediante el Componente Reporte Monitoreo.

Tabla 3.7. Pruebas de funcionamiento Módulo Reporte y Componente Reporte General

Prueba	Estado	Observación
Revisar que la aplicación haya creado el reporte.	Aprobado	Ninguna
Revisar las fechas del reporte.	Aprobado	Ninguna
Verificar que el correo se envíe cuando un AP pierda conexión.	Aprobado	Ninguna

Como resultado, el Módulo Reporte y el Componente Reporte General pasó las pruebas realizadas, y no se obtuvo observaciones. En la Figura 3.9 se presenta una captura de la prueba Revisar las fechas del reporte, de modo que, se muestra el reporte en formato PDF.

Figura 3.9. Reporte en formato PDF

3.2.7. MÓDULO MONITOREO Y COMPONENTE CONFIGURACIÓN

En la Tabla 3.8 se presenta la lista de las pruebas de funcionamiento que se realizaron al Módulo Monitoreo mediante el Componente Configuración.

Tabla 3.8. Pruebas de funcionamiento Módulo Monitoreo y Componente Configuración

Prueba	**Estado**	**Observación**
Visualizar las tareas programadas de la aplicación.	Aprobado	Ninguna
Activar o desactivar un parámetro de monitoreo.	Aprobado	Ninguna
Editar el límite y el tiempo de monitoreo de un parámetro	Aprobado	Ninguna

Como resultado, el Módulo Monitoreo y Componente Configuración pasó las pruebas realizadas, y no se obtuvo observaciones. En la Figura 3.10 se presenta una captura de la prueba Visualizar las tareas programadas en la aplicación, por lo que, se muestra dos tablas con información de las tareas programadas (izquierda) y de los parámetros que se pueden editar (derecha).

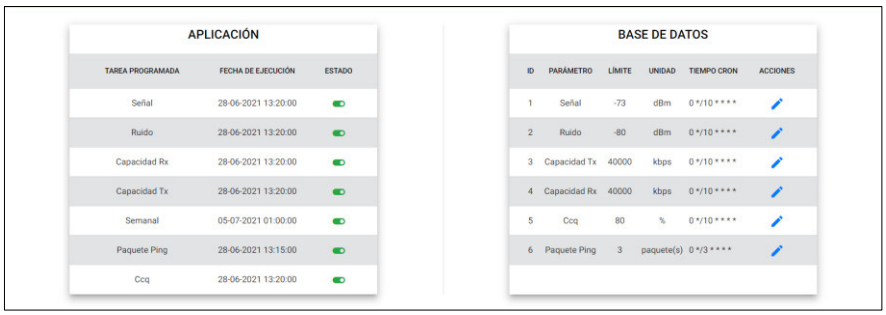

Figura 3.10. Visualizar tareas programadas

3.2.8. ACCESO BASADO EN ROLES

En la Tabla 3.9 se presenta la lista de pruebas que se realizaron al prototipo de aplicación web para verificar los permisos que posee cada rol.

Tabla 3.9. Acceso Basado en Roles

Prueba	Permiso			Estado	Observación
	Administrador	Analista	Usuario		
Ver Personas.	Sí	No	No	Aprobado	Ninguna
Crear Personas.	Sí	No	No	Aprobado	Ninguna
Actualizar Personas.	Sí	No	No	Aprobado	Ninguna
Eliminar Personas.	Sí	No	No	Aprobado	Ninguna
Ver Equipos.	Sí	Sí	No	Aprobado	Ninguna
Crear Equipos.	Sí	Sí	No	Aprobado	Ninguna
Actualizar Equipos.	Sí	Sí	No	Aprobado	Ninguna
Eliminar Equipos.	Sí	Sí	No	Aprobado	Ninguna
Editar tiempo de monitoreo y límite de parámetro.	Sí	Sí	No	Aprobado	Ninguna
Ver alarmas y reportes de Equipos.	Sí	Sí	Sí	Aprobado	Ninguna

Como resultado, el Acceso Basado en Roles pasó las pruebas realizadas y no se obtuvo observaciones.

En las Figura 3.11, Figura 3.12 y Figura 3.13 se presentan capturas del *dashboard* para los roles Administrador, Analista y Usuario respectivamente.

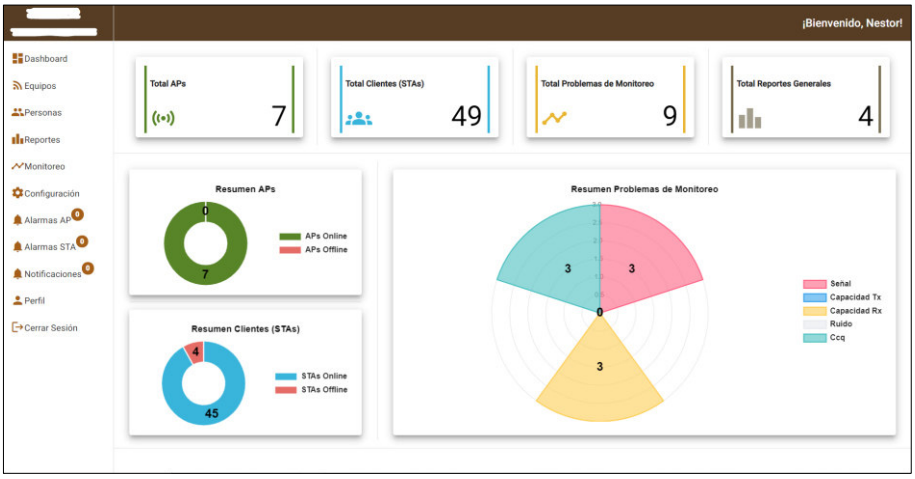

Figura 3.11. *Dashboard* para rol Administrador

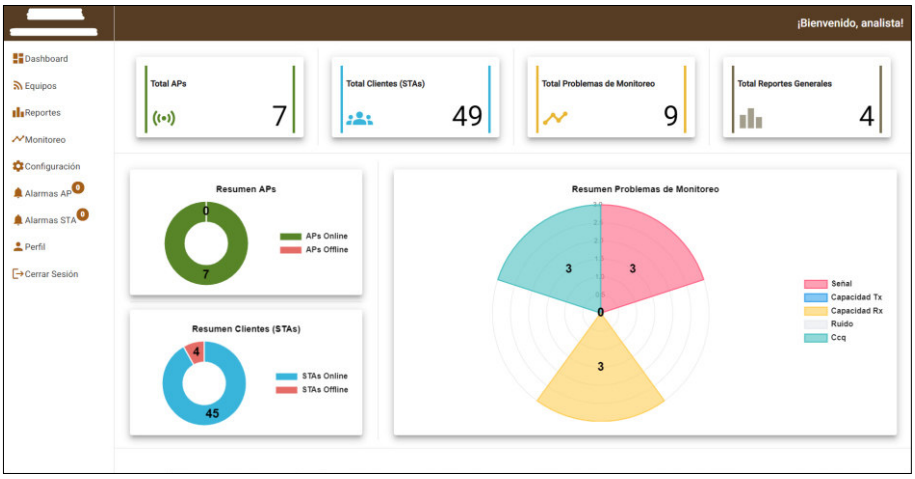

Figura 3.12. *Dashboard* para rol Analista

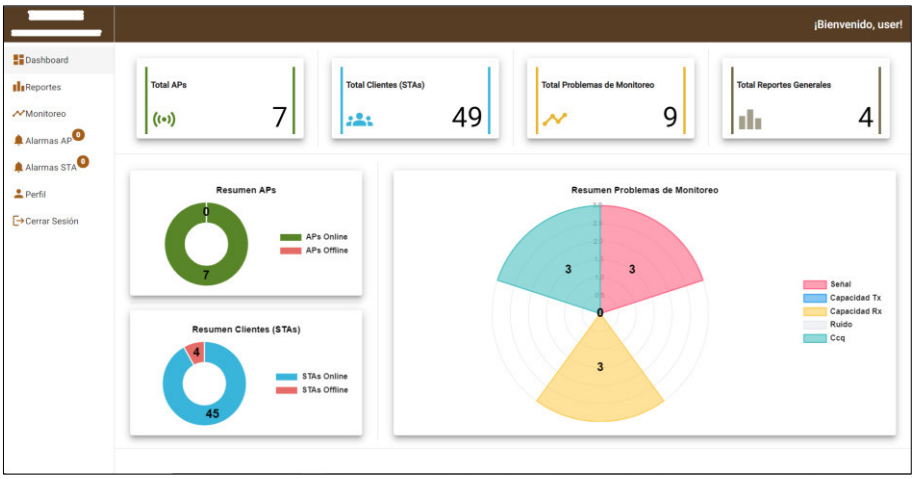

Figura 3.13. *Dashboard* para rol Usuario

3.3. ENCUESTAS DE SATISFACCIÓN

Para llevar a cabo las encuestas de satisfacción, se hizo una reunión con al administrador de la red de la microempresa para realizar la validación de los requerimientos funcionales de la aplicación, también, se realizaron siete preguntas a cinco personas de la microempresa y a cinco compañeros de la FIEE (Facultad de Ingeniería Eléctrica y Electrónica), con el fin de tener una retroalimentación respecto al producto final.

En la Tabla 3.10 se presenta la validación de los requerimientos funcionales que se realizó al administrador de la red de la microempresa.

Tabla 3.10. Validación de los requerimientos funcionales

No	Pregunta	Respuesta	
		Sí	No
1	¿La aplicación cuenta con tres roles de usuario (Administrador, Analista y Usuario)? • Administrador: puede ver, crear, actualizar y eliminar Personas con rol Analista o Usuario, asimismo, podrá leer, crear, actualizar o eliminar dispositivos (APs y STAs). Además, puede ver las alarmas y los reportes de los dispositivos (APs y STAs). De igual importancia, edita el tiempo de monitoreo y el límite de los parámetros a monitorear • Analista: puede ver, crear, actualizar o eliminar dispositivos (APs y STAs), también, puede ver las alarmas y los reportes de los dispositivos (APs y STAs). Igualmente edita el tiempo de monitoreo y el límite de los parámetros a monitorear. • Usuario: puede ver las alarmas y los reportes de los dispositivos (APs y STAs).	X	
2	¿Pudo ver las alarmas y reportes de APs y STAs?	X	
3	¿Recibió un correo electrónico cuando un AP se desconectó?	X	
4	¿La aplicación tiene información de APs (IP, Nombre del Equipo, MAC, Modelo del Equipo, SSID, Frecuencia, Ancho de Canal, Número de Clientes) y STAs (IP, Nombre del Equipo, MAC), además, se muestra en tablas y existe un filtro de búsqueda?	X	
5	¿La aplicación monitorea los parámetros: señal, capacidad de trasmisión, capacidad de recepción, ruido y CCQ?	X	
6	¿La aplicación genera reportes automatizados cada lunes a la 01:00 AM y se puede ver el detalle de cada reporte en PDF?	X	
7	¿Puede ingresar a la aplicación mediante correo electrónico o nombre de usuario?	X	
8	¿La aplicación muestra el número total de APs, número total de STAs (Clientes), número total de problemas de monitoreo, número total de reportes generados por la aplicación, número total de APs *online* y *offline*; número total de STAs (Clientes) *online* y *offline*; y número total de problemas de monitoreo por parámetro?	X	
9	¿La aplicación muestra los últimos 10 eventos ocurridos en la aplicación?	X	
10	¿La aplicación actualiza la página principal cuando un AP o STA se desconecta?	X	
11	¿La aplicación guarda automáticamente los clientes (STAs) nuevos?	X	

En la Tabla 3.11 se presenta los resultados de las encuestas de satisfacción realizadas a cinco personas de la microempresa.

Tabla 3.11. Resultados de las encuestas de satisfacción (microempresa)

No	Pregunta	Resultado (%)
1	Al interactuar con la aplicación, ¿cuál fue el nivel de dificultad?	
	a) Muy fácil	100
	b) Fácil	
	c) Difícil	
	d) Muy difícil	
2	¿Cómo calificaría la interfaz gráfica?	
	a) Excelente	100
	b) Muy buena	
	c) Buena	
	d) Regular	
3	¿Considera que la página principal (*dashboard* web) contiene información relevante del monitoreo de equipos (*Access Points* y *Stations*)?	
	a) Sí	100
	b) No	
	c) Tal vez	
4	Al ingresar un *Access Point*, ¿cuál fue el nivel de dificultad?	
	a) Muy fácil	20
	b) Fácil	80
	c) Difícil	
	d) Muy difícil	
5	Al buscar alarmas de APs (*Access Points*) o STAs (*Station*), ¿cuál fue el nivel de dificultad?	
	a) Muy fácil	60
	b) Fácil	40
	c) Difícil	
	d) Muy difícil	
6	Al buscar alarmas de monitoreo, ¿cuál fue el nivel de dificultad?	
	a) Muy fácil	40
	b) Fácil	60
	c) Difícil	
	d) Muy difícil	
7	¿Considera que la aplicación ayuda a centralizar el monitoreo de *Access Points* y *Stations*?	
	a) Sí	100
	b) No	
	c) Tal vez	

En la Tabla 3.12 se presenta los resultados de las encuestas de satisfacción realizadas a compañeros de la FIEE (Facultad de Ingeniería Eléctrica y Electrónica).

Tabla 3.12. Resultados de las encuestas de satisfacción (compañeros FIEE)

No	Pregunta	Resultado (%)
1	Al interactuar con la aplicación, ¿cuál fue el nivel de dificultad?	
	e) Muy fácil	20
	f) Fácil	80
	g) Difícil	
	h) Muy difícil	
2	¿Cómo calificaría la interfaz gráfica?	
	e) Excelente	80
	f) Muy buena	20
	g) Buena	
	h) Regular	
3	¿Considera que la página principal (*dashboard* web) contiene información relevante del monitoreo de equipos (*Access Points* y *Stations*)?	
	d) Sí	100
	e) No	
	f) Tal vez	
4	Al ingresar un *Access Point*, ¿cuál fue el nivel de dificultad?	
	e) Muy fácil	20
	f) Fácil	80
	g) Difícil	
	h) Muy difícil	
5	Al buscar alarmas de APs (*Access Points*) o STAs (*Station*), ¿cuál fue el nivel de dificultad?	
	e) Muy fácil	20
	f) Fácil	80
	g) Difícil	
	h) Muy difícil	
6	Al buscar alarmas de monitoreo, ¿cuál fue el nivel de dificultad?	
	e) Muy fácil	40
	f) Fácil	60
	g) Difícil	
	h) Muy difícil	
7	¿Considera que la aplicación ayuda a centralizar el monitoreo de *Access Points* y *Stations*?	
	d) Sí	100
	e) No	
	f) Tal vez	

En el ANEXO J se encuentran las encuestas realizadas.

3.4. CORRECCIÓN DE ERRORES

En primer lugar, se revisó las observaciones de las Tabla 3.4 y Tabla 3.6 de las secciones 3.2.3 y 3.2.5 respectivamente, después, en la Tabla 3.13 se procedió a especificar los errores y las correcciones de los problemas encontrados.

Tabla 3.13. Corrección de errores

Error	Corrección
Palabra mal escrita.	Se cambió la palabra “encontrados” por “encontradas”
En el filtrado de *logs* se generan dobles alarmas.	Para filtrar los mensajes logs se utilizó la función `string.includes(string)` que permite buscar una cadena de texto en otra cadena de texto, entonces, al buscar la palabra clave `associated` en algún mensaje log, se encontraban las palabras `associated` y `disassociated`, por consiguiente, se creaban dos alarmas de Equipo Encendido y Equipo Apagado. Para corregir este error se cambió la función `string.includes(string)` por expresiones regulares para garantizar que la aplicación encuentre la palabra clave específica.
Palabras mal escritas.	Se cambió las palabras *Up* y *Down* por *Online* y *Offline* respectivamente.

3.5. TABLERON KANBAN FINAL

En la Tabla 3.14 se presenta todas las tareas realizadas, por consiguiente, se ha culminado con el desarrollo del prototipo de aplicación web.

Tabla 3.14. Tablero Kanban final

Tareas	Tareas en proceso	Tareas realizadas
		Realizar la primera entrevista al gerente técnico de la microempresa.
		Realizar consultas SNMP de prueba a los equipos (APs y STAs).
		Determinar requerimientos funcionales y no funcionales.
		Revisar características generales de los equipos (APs y STAs).
		Revisar la MIB de la casa fabricante.
		Revisar el lenguaje de programación TypeScript.
		Revisar la documentación de NestJS.
		Realizar una figura de la arquitectura del prototipo.
		Realizar el diagrama entidad relación y relacional de la base de datos.
		Realizar el diagrama de actividades del *backend*.
		Realizar el diagrama de clases del *backend*.
		Realizar la segunda entrevista al gerente técnico de la microempresa y actualizar requerimientos.
		Realizar los *mockups.*

Tareas	Tareas en proceso	Tareas realizadas
		Instalar Visual Studio Code, Node.js, MySQL, Git y Postman.
		Codificar la aplicación `backend`
		Probar el funcionamiento del `backend`.
		Codificar la aplicación `frontend`
		Probar el funcionamiento del `frontend`.
		Desplegar el prototipo de aplicación web en la microempresa.
		Realizar las pruebas de funcionamiento de los módulos y componentes de la aplicación.
		Realizar las encuestas de satisfacción.
		Realizar las correcciones de los problemas encontrados.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- En este Proyecto de Titulación se desarrolló un prototipo de aplicación web para el monitoreo de dispositivos en un WISP (*Wireless Internet Service Provider*). Lo más importante del desarrollo de este prototipo fue que se logró centralizar el monitoreo de *Access Points* y *Stations* porque antes de desplegar la aplicación en una subred de la microempresa el monitoreo de *Access Points* y *Stations* se lo hacía de forma manual y descentralizada. Lo que más ayudó a desarrollar este prototipo fueron los *frameworks* NestJS y Angular porque se pudo codificar el *backend* y gran parte del *frontend* con el lenguaje de programación TypeScript. Lo más difícil del desarrollo del prototipo fue crear el método para realizar las consultas SNMP (*Simple Network Management Protocol*) a los equipos porque la librería [3] no proporciona funciones que retornen Promesas.
- La aplicación web fue diseñada en base a los requerimientos del administrador de red de una microempresa, ya que, esta microempresa facilitó sus instalaciones para realizar las pruebas de funcionamiento del prototipo de aplicación web, además, fue implementada de forma modular, por lo que, la aplicación puede seguir creciendo.
- En cuanto a la implementación de la tabla de auditoría (`evento_app`) en la base de datos, se crearon *triggers* para escuchar eventos de inserción, eliminación y actualización de registros porque la escucha de eventos que ofrece TypeORM solo funcionan entre la aplicación *backend* y la base de datos, de modo que, en el caso de que alguien logre ingresar a la base de datos e ingrese código SQL (*Structured Query Language*) para insertar, eliminar o actualizar algún registro, TypeORM no escuchará estos eventos, ya que, el código SQL (*Structured Query Language*) se está ejecutando directamente en la base de datos sin el uso del módulo TypeORM de la aplicación *backend*.
- Según las alarmas generadas por la aplicación se determinó que la alarma Equipo Apagado en un cliente implica varios factores, por ejemplo, corte de energía eléctrica en el domicilio del cliente, la antena del cliente no está fija, el cliente apagó el equipo (*Station*) de forma manual cuando se fue a dormir o cayó un rayo cerca de la zona del cliente haciendo que el equipo (*Station*) se queme. Por otra parte, se constató que algunos enlaces presentan intermitencias ocasionando que ciertos clientes se desconecten y se vuelvan a conectar de los *Access Points*.
- Al revisar las MIBs (*Management Information Base*) de la casa fabricante se detectó que los modelos NanoStation M2 trabajan con la MIB (*Management Information*

Base) del año 2014, por consiguiente, estos dispositivos no guardan información respecto a la capacidad de transmisión y capacidad de recepción.

- Durante las pruebas de funcionamiento en un ambiente real, se encontró un error que hacía que la aplicación asigne dos alarmas de forma consecutiva cuando cualquier *Station* se encendía, por tanto, se tuvo que revisar y depurar el código con el fin de solucionar este problema, ya que, se guardaban alarmas erróneas de las *Stations*.
- Por último, el prototipo de aplicación web se ha convertido en una herramienta de apoyo para el administrador de red y el equipo de trabajo de la microempresa, además, sirve para llevar un inventario de los equipos que posee la subred en la que fue desplegada la aplicación.

4.2. RECOMENDACIONES

- Al realizar cambios en los archivos de las Entidades se recomienda borrar la carpeta `dist` del proyecto de NestJS y todas las tablas de la base de datos, para que, cuando se compile el proyecto de NestJS vuelva a generar la carpeta `dist` y las tablas de la base de datos. Además, tener una base de datos de pruebas para realizar ensayos de la aplicación.
- Para manejar atributos de fecha y hora trabajar con el tipo `datetime` en la base de datos y en la aplicación (*backend* y *frontend*) usar el tipo `Date` porque si se maneja distintos formatos de fecha al codificar las partes de la aplicación puede ocasionar que se generen mal las consultas a la base de datos cuando se requiere obtener registros por fechas.
- Al crear la tabla auditoría para escuchar eventos (insertar, actualizar o eliminar) elegir las tablas más críticas, ya que, se tendrá información redundante y conforme pase el tiempo se podría necesitar más memoria en el disco duro.
- Al generar la aplicación *frontend*, en el archivo `environment.prod.ts` colocar de forma correcta la dirección IP donde está trabajando la aplicación *backend* porque si la dirección fue ingresada mal nunca se podrá conectar el *frontend* con el *backend*.
- Usar el archivo `.env` para evitar mostrar contraseñas o información sensible en el código de la aplicación.
- Usar *frameworks* alternativos para implementar el *frontend* por ejemplo VueJS o crear una aplicación móvil con Ionic, a fin de comparar el rendimiento entre la aplicación web y una aplicación móvil.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] NestJS, «NestJS,» [En línea]. Available: https://www.nestjs.com/. [Último acceso: 07 06 2021].

[2] Angular, «Angular,» [En línea]. Available: https://www.angular.io. [Último acceso: 07 06 2021].

[3] M. Abrahams, N. Ltd y S. Vickers, «net-snmp,» [En línea]. Available: https://www.npmjs.com/package/net-snmp. [Último acceso: 07 06 2021].

[4] «airOS8 User Guide,» [En línea]. Available: https://dl.ubnt.com/guides/airOS/airOS_UG_V80.pdf . [Último acceso: 07 06 2021].

[5] D. Zelisko, «ping - npm,» [En línea]. Available: https://www.npmjs.com/package/ping. [Último acceso: 07 06 2021].

[6] M. P. Arteaga y D. S. Guamán, «Desarrollo de una interfaz gráfica de usuario para la administración de dispositivos de red por medio de NETSNMP en el sistema operativo Linux,» Escuela Politécnica Nacional, Quito, 2010.

[7] R. Gerhards y A. GmbH, «The Syslog Protocol,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc5424. [Último acceso: 07 06 2021].

[8] R. Alex, «Servicios Web de RESTful: Los aspectos básicos,» [En línea]. Available: https://developer.ibm.com/es/technologies/web-development/articles/ws-restful/. [Último acceso: 08 06 2021].

[9] Mozilla, «Códigos de estado de respuesta HTTP,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Status. [Último acceso: 08 07 2021].

[10] Mozilla, «Server-sent events,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events. [Último acceso: 08 06 2021].

[11] I. Kantor, «Server Sent Events,» [En línea]. Available: https://javascript.info/server-sent-events. [Último acceso: 08 06 2021].

[12] VSC, «Descripción General,» [En línea]. Available: https://code.visualstudio.com/docs. [Último acceso: 09 06 2021].

[13] Node.js, «Introducción a Node.js,» [En línea]. Available: https://nodejs.dev/learn/introduction-to-nodejs. [Último acceso: 09 06 2021].

[14] Node.js, «Diferencias entre Node.js y el navegador,» [En línea]. Available: https://nodejs.dev/learn/differences-between-nodejs-and-the-browser. [Último acceso: 09 06 2021].

[15] NestJS, «Introducción NestJS,» [En línea]. Available: https://docs.nestjs.com/#introduction. [Último acceso: 09 06 2021].

[16] NestJS, «Módulos,» [En línea]. Available: https://docs.nestjs.com/modules#modules. [Último acceso: 09 06 2021].

[17] NestJS, «Controladores,» [En línea]. Available: https://docs.nestjs.com/controllers#controllers. [Último acceso: 09 06 2021].

[18] NestJS, «Providers,» [En línea]. Available: https://docs.nestjs.com/providers#providers. [Último acceso: 09 06 2021].

[19] NestJS, «Servicios,» [En línea]. Available: https://docs.nestjs.com/providers#services. [Último acceso: 09 06 2021].

[20] TypeORM, «TypeORM,» [En línea]. Available: https://typeorm.io/#/. [Último acceso: 09 06 2021].

[21] Angular, «¿Qué es Angular?,» [En línea]. Available: https://angular.io/guide/what-is-angular. [Último acceso: 09 06 2021].

[22] Angular, «Descripción general de los componentes de Angular,» [En línea]. Available: https://angular.io/guide/component-overview#angular-components-overview. [Último acceso: 09 06 2021].

[23] Angular, «Introducción a los servicios y la inyección de dependencia,» [En línea]. Available: https://angular.io/guide/architecture-services#introduction-to-services-and-dependency-injection. [Último acceso: 09 06 2021].

[24] Postman, «Postman,» [En línea]. Available: https://www.postman.com/. [Último acceso: 10 06 2021].

[25] J. Edge, «Kanban: La guía definitiva de la metodología Kanban para el desarrollo de software ágil,» Kindle, 2018.

[26] «Descargar VSC,» [En línea]. Available: https://code.visualstudio.com/download. [Último acceso: 11 06 2021].

[27] «Descargar NodeJS,» [En línea]. Available: https://nodejs.org/es/download/. [Último acceso: 11 06 2021].

[28] NestJS, «Instalar NestJS,» [En línea]. Available: https://docs.nestjs.com/first-steps#setup. [Último acceso: 11 06 2021].

[29] NestJS, «Instalar TypeORM con MySQL,» [En línea]. Available: https://docs.nestjs.com/techniques/database#typeorm-integration. [Último acceso: 11 06 2021].

[30] Angular, «Instalar Angular,» [En línea]. Available: https://angular.io/guide/setup-local#install-the-angular-cli. [Último acceso: 11 06 2021].

[31] A. Material, «Instalar Angular Material,» [En línea]. Available: https://material.angular.io/guide/getting-started.

[32] GitHub, «GitHub,» [En línea]. Available: https://github.com. [Último acceso: 11 06 2021].

[33] Postman, «Descargar Postman,» [En línea]. Available: https://www.postman.com/downloads/. [Último acceso: 11 06 2021].

[34] L. Thess, «Instalar syslog-server,» [En línea]. Available: https://www.npmjs.com/package/syslog-server. [Último acceso: 11 06 2021].

[35] A. Reinman, «Instalar nodemailer,» [En línea]. Available: https://www.npmjs.com/package/nodemailer. [Último acceso: 11 06 2021].

[36] Insomniocode, «Instalar pdfmake wrapper,» [En línea]. Available: https://github.com/Lugriz/pdfmake-wrapper. [Último acceso: 11 06 2021].

[37] Queenkunkun, «Instalar angular-sse-client,» [En línea]. Available: https://www.npmjs.com/package/angular-sse-client. [Último acceso: 11 06 2021].

[38] GitHub, «Creating a personal access token,» [En línea]. Available: https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token. [Último acceso: 11 06 2021].

[39] «Instalar class validator,» [En línea]. Available: https://www.npmjs.com/package/class-validator. [Último acceso: 15 06 2021].

[40] «Instalar eventemitter2,» [En línea]. Available: https://www.npmjs.com/package/eventemitter2. [Último acceso: 20 06 2021].

[41] A. Material, «Input,» [En línea]. Available: https://material.angular.io/components/input/overview. [Último acceso: 23 06 2021].

[42] S. Cooper, «ngx-toastr,» [En línea]. Available: https://www.npmjs.com/package/ngx-toastr. [Último acceso: 22 06 2021].

[43] A. Material, «Card,» [En línea]. Available: https://material.angular.io/components/card/overview. [Último acceso: 23 06 2021].

[44] valor-software, «Instalar ng2-charts,» [En línea]. Available: https://www.npmjs.com/package/ng2-charts. [Último acceso: 23 06 2021].

[45] A. Material, «Table,» [En línea]. Available: https://material.angular.io/components/table/overview. [Último acceso: 23 06 2021].

[46] Sweetalert2, «Instalar sweetalert2,» [En línea]. Available: https://www.npmjs.com/package/sweetalert2. [Último acceso: 24 06 2021].

[47] A. Material, «Form,» [En línea]. Available: https://material.angular.io/components/form-field/overview. [Último acceso: 24 06 2021].

ANEXOS

ANEXO A. Entrevista

ANEXO B. Diagrama entidad-relación

ANEXO C. Diagrama relacional

ANEXO D. Diagrama de actividades

ANEXO E. Diagrama de clases

ANEXO F. *Mockups*

ANEXO G. Aplicación `backend` (comprimido)

ANEXO H. *Script* para generar *triggers* en la base de datos

ANEXO I. Aplicación `frontend` (comprimido)

ANEXO J. Encuestas

ANEXO K. Siglas y acrónimos

ANEXO K

Listado de siglas y acrónimos utilizados en el presente Trabajo de Titulación.

- **AP:** *Access Point*
- **CLI:** *Command-Line Interface*
- **HTTP:** *Hypertext Transfer Protocol*
- **ICMP:** *Internet Control Message Protocol*
- **IP:** *Internet Protocol*
- **MAC:** *Medium Access Control*
- **MIB:** *Management Information Base*
- **ORM:** *Object Relational Mapping*
- **PYME:** Pequeña y Mediana Empresa
- **REST:** *Representational State Transfer*
- **SNMP:** *Simple Network Management Protocol*
- **SQL:** *Structured Query Language*
- **SSE:** *Server Sent Events*
- **SSID:** *Service Set Identifier*
- **STA:** *Station*
- **WISP:** *Wireless Internet Service Provider*

ORDEN DE EMPASTADO