Diseño de una Plataforma IOT para el Monitoreo y Reporte de Fallas del Alumbrado Público Tipo LED en Zonas Urbanas.

María Fernanda Vera Negrón y Esteban Andrés Niño Méndez

Trabajo de Grado para Optar el Título de Ingeniero de Sistemas

Director

Ing. Jose Geralbert Rubiano

Especialista en Redes de Computadoras

Universidad Industrial de Santander

Facultad de Ingenierías Físico Mecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2021

Dedicado a

Mi madre Argenis Méndez Jiménez por la ayuda y apoyo que me brindo en el transcurso de la carrera y de mi vida además de motivarme cada día a ser mejor.

Mis hermanos que igualmente me ayudaron mientras transcurría la carrera, por acompañarme en este camino y por no abandonarme.

Mi director de proyecto por instruirme y guiarme en el desarrollo del proyecto de grado, por cada minuto que dedico para poder culminar el trabajo y además por los consejos que me dio.

Y además quiero agradecer a todos mis amigos y profesores que estuvieron en cada momento a lo largo de la carrera, por la paciencia que tuvieron conmigo, por cada tiempo que compartimos, cada momento bueno y malo que vivimos y por la sabiduría que me brindaron

Agradezco a la vida por esta linda etapa vivida.

Esteban Andrés Niño Méndez

Dedicado a

A mis padres, María Teresa y Jorge, por su amor y dedicación, por acompañarme y apoyarme en cada una de las decisiones que he tomado y motivarme siempre a dar lo mejor de mí misma.

A mi hermana, Karen Daniela, por escucharme siempre en los momentos buenos y malos, brindarme su apoyo incondicional y ayudarme a ver las cosas desde otras perspectivas.

A mis familiares y amigos, por acompañarme en las diferentes etapas de mi vida, por los momentos compartidos y los consejos brindados.

A mi gata, Kira, por ser la mejor compañía en momentos difíciles.

Maria Fernanda Vera Negrón

Agradecimientos

Al profesor José Rubiano, por su sabiduría, dirección y apoyo, por el tiempo que nos brindó para cumplir nuestros objetivos en el proyecto y por ser un excelente instructor.

A nuestros colegas y amigos, Natalia Gómez, Andrea Villamizar y Bernardo Vega, por su gran apoyo y colaboración a lo largo de estos 5 años de carrera donde sus enseñanzas tanto a nivel profesional como personal hicieron de esta una experiencia inolvidable.

A cada uno de nuestros familiares, por su apoyo incondicional en estos momentos complicados que estamos viviendo, por esa motivación y ayuda tan importante que nos aportaron.

A todos nuestros amigos y profesores, por cada momento que compartimos y cada colaboración que nos dimos, fueron 5 años de amistades construidas y de etapas buenas y malas vividas, pero lo importante son las enseñanzas y lecciones aprendidas.

¡Muchísimas gracias!

Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Glosario

AWS: Amazon Web Service.

BackEnd: Es la parte de un programa que maneja la parte lógica, por ende, permanece oculto ante el usuario.

Broker: Es un programa intermediario que sirve de puente para que los dispositivos electrónicos se puedan comunicar con la aplicación.

Callback: Es una función respuesta de una función padre.

Cloud Computing: Es la entrega de recursos computacionales bajo demanda desde aplicaciones hasta centro de datos, a través de internet por un pago por uso base.

DMA: Direct Memory Access, Permite a un cliente acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento

DNS: Domain Name System (Sistema de nombre de dominio) Cambia la dirección IP a un nombre para lectura humana.

Endpoints: Son rutas incluidas en una URL desde la cual las API a los recursos necesarios para poder realizar cierta función.

Firewall: Es el sistema que nos permite dar seguridad a nuestra aplicación.

FrontEnd: Es la parte de un programa del cual el usuario puede interactuar directamente.

HTTP: Protocolo de transferencia de hipertexto (Hypertext Transfer Protocol).

I2C: Inter-Integrated Circuit, es un puerto y protocolo de comunicación serial, define la trama de datos y las conexiones físicas para transferir bits entre 2 dispositivos digitales.

IaaS: Infrastructure As A Service (Infraestructura como servicio).

IAM: Identity and Access Management, es un servicio de AWS para poder administrar accesos a los recursos.

ICSP: In Circuit Serial Programming, Es un conector que consiste de las señales de MOSI, MISO, SCK, RESET, VCC, GND.

IoT: Internet of Things.

JSON: JavaScript Object Notation.

JWT: Json Web Token, Nos sirve para la creación de tokens con el cual podemos conocer la identidad del usuario.

Local storage: permite almacenar datos de manera local en el navegador y sin necesidad de realizar alguna conexión a base de datos.

Middlewares: Es la parte que incluye todos los archivos que ofrecen servicios y funciones comunes para las aplicaciones, en él independientemente del software, se pueden tomar acción en cualquier código, permitiendo desarrollar con mayor eficiencia actuando de una forma uniforme.

MISO: Master Input Slave Output, Es la señal de entrada a nuestro dispositivo, por aquí se reciben los datos desde el otro integrado.

MOSI: Master Output Slave Input, Transmisión de datos hacia el otro integrado.

MQTT: Message Queue Telemetry Transport.

NodeJs: Entorno de ejecución de JavaScript con el que se puede ejecutar todo lo programado en este código.

Nube Híbrida: Es la unión de la nube pública y privada, maneja servicios que pueden ser compartidos como parte que son privadas para la empresa, es el tipo de nube que más se está utilizando actualmente.

Nube Privada: Su infraestructura es de uso único, con este tipo de nube no se puede navegar desde una dirección ip pública y tampoco se comparte la información a terceros.

Nube Publica: Cuando todos los servicios son puestos por el proveedor y de forma compartida, además se puede navegar desde el internet público, es decir, es compartida por otras empresas y usuarios.

Paas: Platform As A Service (Plataforma como servicio).

PIO: Entradas y Salidas programables (Programmable Inputs and Outputs).

Putty: Es un emulador gratuito que permite conectar por medio de SSH de un sistema operativo Windows a un servidor Unix o Linux.

PWM: pulse-width modulation, sirve para variar la energía recibida por un dispositivo electrónico variando rápidamente la energía que este recibe, cambiando entre apagado y encendido.

ReactJs: Es una librería de JavaScript la cual nos permite crear interfaces de usuario y facilitar el desarrollo de aplicaciones en cuanto al estilo.

REST: Transferencia de estado representacional (representational state transfer).

SaaS: Software As A Service (Software como servicio).

SCLK: Señal de reloj del bus. Esta señal rige la velocidad a la que se transmite cada bit.

SPA: Single Page Application.

SPI: Serial Peripheral Interface, Es un bus de comunicación de datos entre maestros y esclavo se realiza en dos líneas independientes.

UART: Universal Asynchronous Receiver-Transmitter, es el dispositivo que controla los puertos y dispositivos serie.

VPS: Virtual Private Service (Servidor privado virtual).

Resumen

Título: Diseño de una plataforma IOT para el monitoreo y reporte de fallas del alumbrado público tipo led en zonas urbanas[*]

Autor: María Fernanda Vera Negrón, Esteban Andrés Niño Méndez[**]

Palabras Clave: Alumbrado público, Sensores, Computación en la nube, Plataforma IoT, Arquitectura.

Descripción:

El servicio de alumbrado público constituye un consumo de cerca del 3% del total de energía producida en Colombia según el estudio del Ministerio de Minas y Energía y la Unidad de Planeación Minero-Energética (2016). Este es un servicio esencial cuya prestación debe ser garantizada por las autoridades municipales. Los costos producidos por la prestación de este servicio son asumidos por la ciudadanía a través del pago del impuesto de alumbrado público. Esto quiere decir que los sobrecostos producidos por el mal funcionamiento de las luminarias se traducen en cobros adicionales para los usuarios del servicio. Por otro lado, las luminarias en mal estado o inactivas pueden generar sensación de inseguridad en la población, incluso, ocasionar accidentes debido a la poca iluminación de la zona.

Por esta razón, surge la necesidad de diseñar una plataforma IoT que permita monitorear y reportar fallas del alumbrado público tipo LED en zonas urbanas que facilite el mantenimiento de las luminarias. El sistema propuesto medirá el consumo eléctrico del alumbrado público por sectores y reportará las anomalías que se encuentren. Los datos serán almacenados con el fin de hacer un seguimiento a los reportes y que la plataforma pueda desplegar las estadísticas de los fallos. Además, el sistema deberá mostrar la localización geográfica de los sensores dispuestos. Por último, la plataforma permitirá controlar el suministro energético por zonas remotamente en caso de que sea necesario.

[*] Trabajo de Grado

[**] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Jose Geralbert Rubiano. Especialista en Redes de Computadoras

Abstract

Title: Design of an IoT platform for monitoring and reporting LED-type street lighting failures in urban areas[*]

Author: Maria Fernanda Vera Negrón, Esteban Andrés Niño Méndez[**]

Key Words: Street lighting, Sensors, Cloud computing, IoT platform, Architecture.

Description:

The street lighting service constitutes a consumption about 3% of the total energy produced in Colombia according to the study of the Ministerio de Minas y Energía and Unidad de Planeación Minero Energética (2016). This is an essential service whose provision must be guaranteed by the municipal authorities. The citizens through the payment of the street lighting tax assume the costs produced by the provision of this service. This means that the cost overruns produced by the malfunctioning of the luminaires translate into additional charges for the users of the service. On the other hand, luminaires in poor condition or inactive can generate a feeling of insecurity in the population, or even cause accidents due to poor lighting in the area.

For this reason, the need arises to design an IoT platform that allows monitoring and reporting failures of LED-type street lighting in urban areas that facilitates the maintenance of luminaires. The proposed system will measure the electricity consumption of street lighting by sectors and will report any anomalies that are found. The data will be stored in order to follow up on the reports and that the platform can display the statistics of the failures. In addition, the system must show the geographical location of the established sensors. Finally, the platform will allow to control the energy supply by zones remotely if necessary.

[*] Project of grade

[**] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Jose Geralbert Rubiano. Especialista en Redes de Computadoras

Introducción

El alumbrado público es el servicio público no domiciliario que se presta con el objeto de proporcionar exclusivamente la iluminación de los bienes de uso público y demás espacios de libre circulación con tránsito vehicular o peatonal, dentro del perímetro urbano y rural de un municipio o distrito (ESSA, 2020). Estos pueden comprender los parques, avenidas, túneles, puentes, entre otros.

Actualmente en Colombia este servicio es responsabilidad de la administración municipal, de ellos depende el mantenimiento de las luminarias, modernización y ampliación, donde estas mismas pueden contratar a terceros para realizar este trabajo, siendo un servicio en el cual los recursos se obtienen del impuesto que cobran al alumbrado público, inducido en la factura de la luz en los hogares colombianos, ya que estos se benefician de manera directa o indirectamente.

Desde luego, se debe aclarar que “no se considera servicio de alumbrado público la semaforización, los relojes digitales y la iluminación de las zonas comunes en las unidades inmobiliarias cerradas o en los edificios o conjuntos de uso residencial, comercial, industrial o mixto, sometidos al régimen de propiedad horizontal, la cual estará a cargo de la copropiedad. Así como la iluminación ornamental y navideña en los espacios públicos, solo se considera dentro del sistema de alumbrado público lo que comprende el conjunto de luminarias, redes eléctricas, transformadores y postes de uso exclusivo, los desarrollos tecnológicos asociados al servicio de alumbrado público, y en general todos los equipos necesarios para la prestación del servicio de alumbrado público que no forman parte del sistema de distribución de energía eléctrica” (EVA, 2018).

Al mismo tiempo hay que recalcar que estos sistemas pueden ser afectados por diversas fallas, ya sean lámparas LED que se encuentran apagadas, con encendido intermitente o no existente. Además de errores en las fotoceldas ocasionando mal funcionamiento. Asimismo, existe la posibilidad de robo de energía eléctrica al sistema de alumbrado público por parte de pobladores cercanos, ocasionando un consumo que se traduce en gastos de energía y cobros adicionales al estado. Estos problemas no se resuelven hasta que la ciudadanía reporte el daño por medio de un formulario que debe diligenciar vía web o realizando una llamada telefónica sin costo.

Según lo expuesto anteriormente se presenta una interrogante, ¿existen actualmente herramientas que ayuden a los entes encargados del alumbrado público a obtener aviso oportuno de fallas en el sistema? Al plantear esta pregunta surge la esencia de este trabajo a fin de diseñar un prototipo, con el cual se pueda monitorear y reportar las fallas del alumbrado público tipo Led en zonas urbanas con la ayuda de una plataforma IOT, permitiendo a los entes encargados obtener señales en tiempo real del estado de la placa LED, para así optimizar la generación de reportes de fallas e historial de las mismas

1. Objetivos

1.1 Objetivo General

Diseñar una plataforma IOT para el monitoreo y reporte de fallas del alumbrado público tipo Led en zonas urbanas.

1.2 Objetivos Específicos

- Identificar las variables y requerimientos para el diseño de una plataforma IOT que permita el monitoreo y reporte de fallas en el alumbrado público tipo led en zonas urbanas.
- Seleccionar los componentes para la infraestructura hardware y software involucrados en el desarrollo de la plataforma IOT.
- Diseñar la estructura del hardware y software bajo los parámetros definidos en los objetivos anteriores.
- Evaluar el desempeño del diseño de la plataforma IOT, mediante la elaboración de un prototipo de la plataforma según los objetivos planteados e interacción con una maqueta a escala del sistema.

2. Marco Referencial

2.1. Marco conceptual

2.1.1. Luminarias de alumbrado público

> "Son aparatos que sirven de soporte y conexión a la red eléctrica a las lámparas, Las luminarias poseen dos clases de componentes, uno óptico y otro eléctrico. El primero se compone de todos los elementos necesarios para controlar y dirigir la luz producida por una bombilla y el segundo de brindar las condiciones adecuadas para que la bombilla reciba la energía requerida para un óptimo funcionamiento, todos éstos elementos están contenidos dentro de una carcasa que por lo general se fabrica en aluminio y que se encarga de brindar protección física a la luminaria manteniendo la hermeticidad deseada medida mediante el Índice de Protección (IP)" (González & Pinilla, 2009).

2.1.1.1 Partes del alumbrado público

- Lámparas: hay varios tipos de lámparas para el alumbrado público, como las de vapor de mercurio, con descarga por inducción, las de halogenuros metálicos, vapor de sodio, fluorescentes, LED.
- Balasto: Es el elemento estabilizador que contrarresta la tendencia al crecimiento de la intensidad consumida por una lámpara suministrando a la lámpara la tensión, frecuencia y potencia

adecuadas para un funcionamiento estable, evitando su destrucción por altos consumos de estas características.

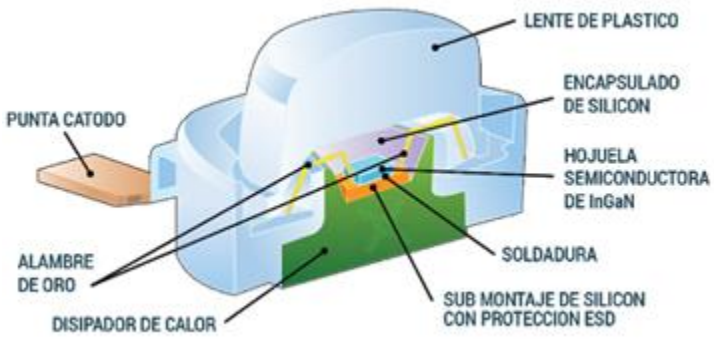
- Condensador: necesario para corregir el factor de potencia del sistema formados por las lámparas y el balasto obteniendo así una disminución en el consumo de energía.
- Arrancador: Es el encargado de generar impulsos de tensión necesarios para el funcionamiento de la lampara.
- Soportes: Abarca todo lo que tiene que ver con columnas, brazos, báculos y demás con el fin de poder mantener la luminaria en la posición deseada.

2.1.2. LED

(proveniente del inglés, Light Emitter Diodes). “son dispositivos fabricados utilizando materiales semiconductores, estos pueden ser, Aluminio (Al), Galio (Ga), Boro (B), Carbono (C), Germanio (Ge), Arsénico (As), Fósforo (P), Silicio (Si). El Silicio y el Galio poseen una propiedad única en su estructura electrónica, ya que tiene cuatro electrones en su órbita externa, esto permite combinar estos electrones con cuatro átomos vecinos, formando con esto una malla cuadriculada o estructura cristalina y de esta forma no queda ningún electrón libre” (Reyes, 2016), emitiendo luz de forma eficiente y con un rendimiento superior a las demás bombillas como las incandescentes, halógenas, fluorescentes, etc. La figura 2 muestra las partes de un Led (Diodo emisor de luz).

Figura 1.

Partes de un LED.



Nota. Tomado de LED Facoel. (2016)

2.1.3. Potencia eléctrica

Es la cantidad de energía que consume un elemento en un tiempo, es decir, la proporción de paso de energía por unidad de tiempo. Su unidad de medida es el vatio (Watt) cuya equivalencia es un julio (J) por segundo.

2.1.4. Internet de las cosas

(proveniente del inglés IOT, Internet of Things). “Internet de las cosas es una red de objetos electrónicos como vehículos, máquinas, electrodomésticos y más, que utiliza sensores y APIs para conectarse e intercambiar datos por internet.” (SAP, s. f.) con el fin de poder controlar de forma remota y/o recibir alertar, así como actualizaciones de los estados del dispositivo.

2.1.5. Sensores

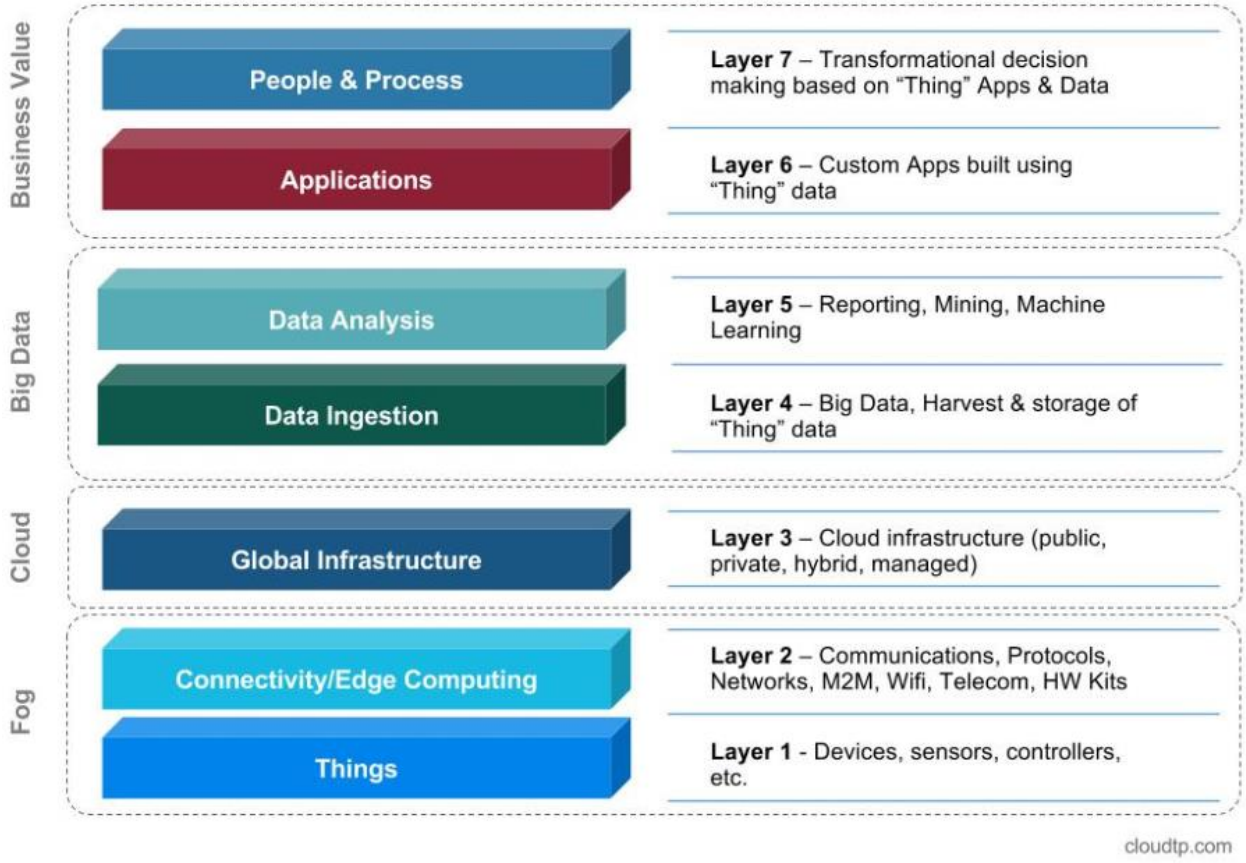
Son dispositivos elementales en una arquitectura IOT, pues tienen la función de recolectar datos del entorno para su posterior tratamiento. Los sensores trabajan detectando magnitudes físicas para cuantificarlas y luego transmitirlas por medio de impulsos eléctricos.

2.1.6. Arquitectura IOT

"La arquitectura describe la estructura de su solución de IOT, lo que incluye los aspectos físicos (esto es, las cosas) y los aspectos virtuales (como los servicios y los protocolos de comunicación). Adoptar una arquitectura con múltiples niveles le permite concentrarse en mejorar su comprensión acerca de cómo todos los aspectos más importantes de la arquitectura funcionan antes de que los integre dentro de su aplicación de IOT. Este enfoque modular le ayuda a gestionar la complejidad de las soluciones IOT" (*Arquitecturas IOT*, 2018), En la figura 2 se muestra un modelo de capas de la arquitectura IOT.

Figura 2.

Modelo de 7 capas de la arquitectura IOT.



Nota. Tomado de Aprendiendo Arduino. Modelos de capas IOT. (2018).

2.1.7 MQTT

MQTT es un servicio de publicación / suscripción TCP / IP simple y muy ligero. Se basa en el principio cliente/servidor. Un servidor llamado broker recopila datos enviados por el Publisher (objeto de comunicación). Ciertos datos recopilados por el broker se envían a un Publisher en particular que previamente solicitó esta información al broker. El Publisher envía

mensajes a un canal llamado Topic. El suscriptor puede leer estos mensajes y recibirlos a su llegada.

2.1.8. Cloud Computing

> "La computación en la nube es un modelo para permitir el acceso a la red ubicuo, conveniente y bajo demanda a una red compartida conjunto de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se puede aprovisionar y liberar rápidamente con un mínimo esfuerzo de gestión o interacción del proveedor de servicios". (National Institute of Standards and Technology, 2011).

2.1.9. API REST

REST (Representational State Transfer) es un estilo arquitectónico independiente del lenguaje de programación para el desarrollo de servicios web (Rouse, M, 2017). Se centra en los recursos del sistema, incluida la forma en que se accede a la información y los estados, y cómo se transfiere a los clientes mediante HTTP (Hypertext Transfer Protocol) utilizando varios lenguajes de programación, y define un conjunto de principios arquitectónicos mediante los cuales se diseñan estos servicios. Hay cuatro principios básicos del diseño de servicios REST (De Seta, L, 2008).

- Utilice el método HTTP.
- Sin estado (no mantiene el estado en varias solicitudes).
- Muestra los URI (identificadores uniformes de recursos) como carpetas.

• Enviar información en formato JSON (JavaScript Object Notation).

2.1.10. Aplicación web

Es un tipo de software que ha sido desarrollado en un lenguaje soportado por los navegadores web, se ejecuta a través de internet o de una red local.

2.1.11. Bases de datos

Se puede definir una base de datos como un sistema de archivos electrónico, pues con esta se busca tener la información de manera organizada, de modo que se pueda acceder a los campos de información de forma eficiente.

2.1.12 Single Page Applications

Una Single Page Application (SPA) o Aplicación de Página Única es una aplicación web que carga todo el contenido en una sola página para mejorar y unificar la experiencia del usuario (Baquero, J, 2017). El navegador no se recarga al moverse de una sección a otra porque todo lo que se muestra y procesa pertenece a la misma página. En cambio, cuando la página se carga por primera vez, los recursos requeridos se procesan primero y luego se van descargando los de las demás secciones de la aplicación. Solo hay una página, pero realmente tiene múltiples vistas y esto es lo que el usuario observa con diferentes elementos e información a medida que se mueve entre ellas.

2.2 Estado del arte

2.2.1 Postes inteligentes de signify.

Los postes inteligentes BrightSites incorporan conectividad de banda ancha, con infraestructura para transmisión de datos de cuarta y quinta generación (4G y 5G) así como los fundamentos para diversas aplicaciones de la "internet de las cosas" (IOT), con esto les dan más funciones a las luminarias, tiene presencia en España, Francia, Turquía, Holanda, Rusia, Argentina y Brasil. (Iluminet, 2019).

2.2.2 CITYMANAGER

Esta aplicación de la empresa Tvilight Empowering Intelligence le permite administrar completamente todas las luminarias conectadas a través de una aplicación web segura y fácil de usar con cualquier navegador web estándar. Accede de forma remota a todos los dispositivos y obtiene información del estado y rendimiento, casi en tiempo real, su centro principal se encuentra en India. (Tvilight, 2020).

2.2.3 CITYTOUCH.

Es una plataforma segura de gestión de la iluminación conectada que contribuye a que las ciudades sean más habitables, eficientes y sostenibles. CityTouch proporciona información y control completos del alumbrado público de la ciudad en un panel centralizado. De este modo,

podrás supervisar y programar de manera segura los puntos de luz y ajustar los niveles de luz según lo necesites. Con la autenticación de dos factores y el cifrado de datos, nos aseguramos de que nuestro software cumpla las normas de seguridad más exigentes. (Philips, s. f.).

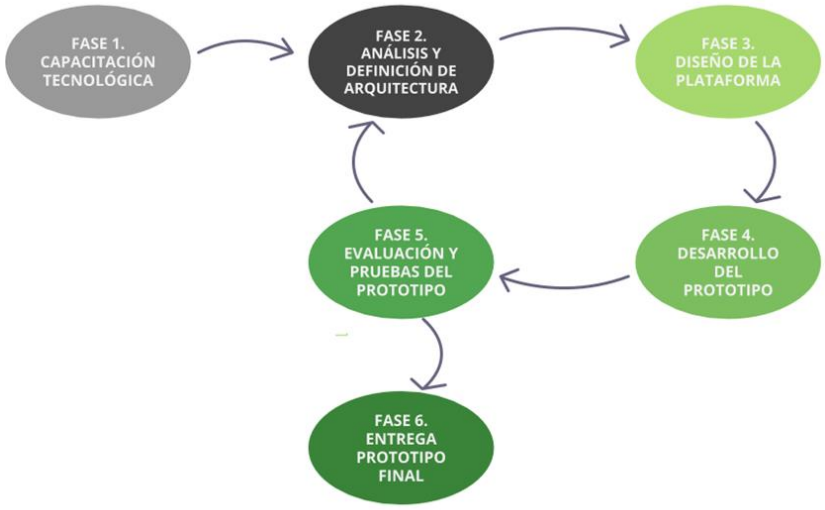
2.2.4 Telegestión alumbrado público Medellín.

El proyecto piloto “Sistema de telegestión para alumbrado público” fue realizado por EPM con un alcance de 3.818 puntos luminosos en tecnologías LED y HID (sigla en inglés de luces de alta intensidad de descarga), buscando la realización del mantenimiento del sistema de alumbrado público de Medellín, reducir costos operacionales, controlar y operar en tiempo real, conocer el consumo de energía por luminaria, garantizar la disponibilidad del servicio, mejorar la calidad de la atención al usuario y reducir el consumo de energía, como contribución al uso eficiente y racional de la energía eléctrica. (EPM, 2020).

3. Marco Metodológico

Figura 3.

Esquema de metodología de trabajo.



3.1 Fase 1: capacitación tecnológica y levantamiento de información

En esta fase se investigaron las tecnologías que se necesitaron para poder finalizar el proyecto, se tomaron conceptos de computación en la nube, redes de comunicación, ingeniería de software, electricidad y electrónica, así como del internet de las cosas. Además, se realizó el levantamiento de información requerido para el diseño de la plataforma.

Las actividades realizadas fueron:

- Recopilación de información necesaria para la implementación de un proyecto de IOT.
- Investigación de casos reales donde se aplique esta tecnología.

- Búsqueda de herramientas que den utilidad en la realización del proyecto.
- Estudio de las Tecnologías de posible utilidad para el desarrollo del prototipo.

3.2. Fase 2: análisis y definición de arquitectura

Se estudió la viabilidad de los proveedores de la nube existentes y de las tecnologías para el Backend y el Frontend que se ajustaran a nuestras necesidades. A su vez, se definieron componentes de hardware para el monitoreo y adquisición de datos de la plataforma.

Las actividades realizadas fueron:

- Estudio del alcance y viabilidad del proyecto.
- Definición de plataforma de software.
- Selección de métricas del proyecto.
- Definición de la arquitectura (Dispositivos, Conectividad, Infraestructura, Aplicación).

3.3. Fase 3: diseño de la plataforma

Una vez determinados los requerimientos del proyecto, se procedió al diseño de la estructura funcional, la cual representa aquellos aspectos de software y hardware que interactúan tanto explícitamente como implícitamente con el usuario.

Las actividades realizadas fueron:

- Diseño de la estructura de la base de datos e interfaz web.
- Diseño de la red de conectividad de los dispositivos.
- Diseño de la arquitectura IOT.

3.4. Fase 4: desarrollo del prototipo

Se procedió a manipular físicamente los dispositivos a utilizar y desarrollar un prototipo funcional de la solución para la plataforma IOT, con la cual se visualizó si lo desarrollado en la etapa de diseño genera los resultados esperados. Las actividades realizadas fueron:

- Implementación de la comunicación entre sensores, actuadores y aplicaciones.
- Almacenamiento de datos en la nube e interacción con ellos.
- Implementación sobre una maqueta a escala.

3.5. Fase 5: evaluación y pruebas del prototipo

Para esta etapa se llevó a cabo la verificación del proceso realizado, se presentaron pruebas de ejecución y se analizó si los resultados cumplieron con los objetivos planteados anteriormente.

Las actividades realizadas fueron:

- Validación de prototipo.
- Ejecución de consultas a la base de datos.
- Despliegue de estadísticas.

3.6. Fase 6: entrega del prototipo final

Procedemos a documentar formalmente la tesis y presentarla ante la escuela de Ingeniería de sistemas e Informática de la Universidad Industrial de Santander, en esta fase concluimos con el proyecto de grado.

Las actividades realizadas fueron:

- Entrega del prototipo final con los ajustes realizados.

4. Desarrollo del proyecto

4.1 Capacitación de tecnologías

Para la realización y mejor inspiración en la realización del proyecto se realización diversos cursos virtuales ya sea por la plataforma de Udemy o por diferentes fuentes gratuitos como lo fue YouTube o Linked In, en Udemy se procedió con el estudio de un curso muy importante con el nombre de IOT MASTERCLASS el cual nos hablaba de Acerca del Broker en Vps Propio, NodeJS, paneles de control, Internet de las cosas y 5G, así como de Arduino.

Además de una consulta exhaustiva acerca de los servicios de Cloud computing, en el cual nos capacitamos para poder manejar los servicios de AWS, estos se realización por video tutoriales en YouTube y también por videos privados a cargo de Walter, con ello se entendió más es concepto de arquitecturas de IOT y como poder optimizar su uso para una funcionalidad eficaz, por otro lado se tomaron cursos de Fernando Torres en la plataforma de Udemy acerca de las tecnologías de NodeJS y ReactJS, las cuales fueron muy útiles para poder realizar este proyecto.

Figura 4.

Cursos estudiados

4.2 Análisis y definición de arquitectura

4.2.1 Comparación y servicios de arquitectura

Tomando como punto de partida las diferentes compañías que ofrecen estos servicios tecnológicos, encontramos las seis más destacadas en este mercado, conformando las principales plataformas de servicios en la nube de la actualidad:

- Amazon Web Services (AWS)
- Microsoft Azure
- Google Cloud Platform
- IBM Cloud
- Alibaba Cloud

- Oracle Cloud

Figura 5.

Calculadora de EC2

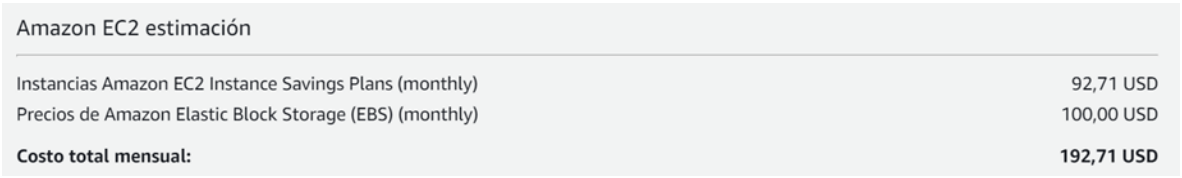


Figura 6.

Calculadora BareMetal IBM

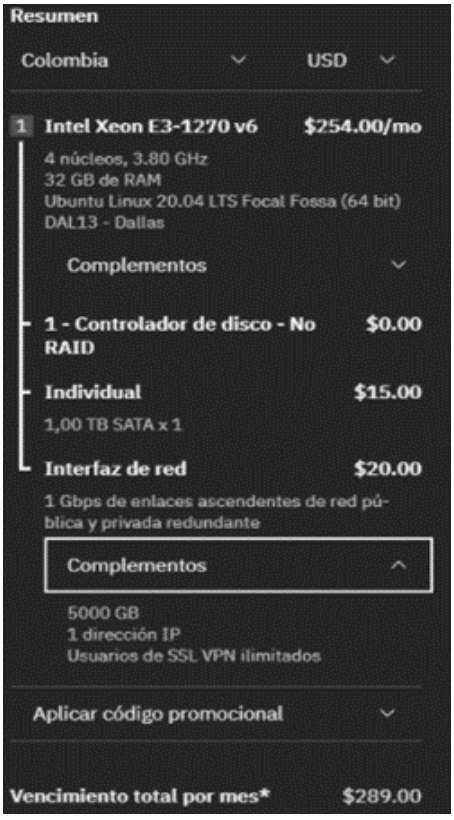


Figura 7.

Calculadora de Google Cloud

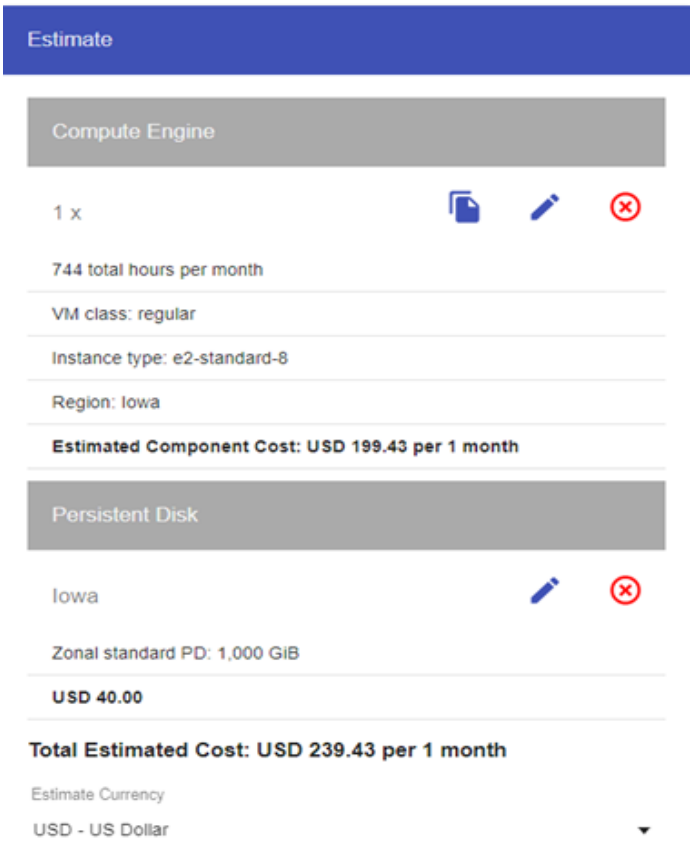


Figura 8.

Calculadora de Servidor Virtual Azure

Microsoft Azure Estimate

Su presupuesto

Service type	Custom name	Region	Description	Estimated monthly cost	Estimated upfront cost
Virtual Machines		West US	1 A4m v2 (4 vCPU, 32 GB de RAM) x 730 Horas; Linux – Ubuntu; Pago por uso; 0 discos administrados: S4, 100 unidades de transacción; Tipo de transferencia interregional, 5 GB de transferencia de datos de salida de West US a East Asia	$216,86	$0,00
Support			Support	$0,00	$0,00
			Licensing Program	Microsoft Online Services Agreement	
			Total	$216,86	$0,00

Disclaimer

All prices shown are in US Dollar ($). This is a summary estimate, not a quote. For up to date pricing information please visit https://azure.microsoft.com/pricing/calculator/
This estimate was created at 6/9/2021 6:36:43 PM UTC.

En las anteriores cotizaciones observamos las diferencias de precios para unas VPS's básicas con 4 núcleos a 32 GB de RAM con el sistema operativo Ubuntu manejando un almacenamiento por disco de 1 TB, hay que destacar que las cuatro manejan una capa gratuita de uso, pero esta es para un promedio de 30 GB de almacenamiento.

Hay que tener en cuenta que las prestaciones de servicios que nos presentan las anteriores compañías son muy buenas para lo que deseamos, Descartamos en primera instancia el servicio de IBM ya que esta no presenta una capa gratuita de manejo, por otro lado Google maneja un inconveniente, ya que este ha sido afectado por cuenta de spam de terceros a la hora de realizar uso de envió de correos, ellos han decidido bloquear todos los puertos de sus servidores de correos electrónicos, y este servicio es fundamental para nuestro proyecto ya que vamos a realizar envíos de correos para notificaciones de fallas en el servicio, quedando en la lista AWS y Azure pero por cuestiones de encontrar información y grupos técnicos en redes sociales además de la experiencia anteriormente tomada se ha decidido escoger a EC2 de Amazon Web Services ya que nos da una capa gratuita que incluye 750 horas mensuales de uso de la instancia t2.micro de Linux y Windows por 12 meses. Si dado el caso llagamos a utilizar dos instancias y la suma de utilidad de estas dos sobrepasan las 750 horas, uno solo pagará por el uso que se les dé a las instancias.

Además de la decisión de utilizar el sistema operativo Linux Ubuntu ya que el manejo es mucho más practico por medio de consolas, la instalación de carpetas además de ficheros y con ello poder realizar seguimiento de tutoriales que se realizaron en este sistema operativo.

4.2.2 Comparación de lenguajes para el Back-End

Siendo el Backend los algoritmos que manejan toda la lógica del lado del servidor como por ejemplo los llamados a las bases de datos, existen lenguajes de programación de alto nivel del cual su comprensión es fácil de leer y de examinar, y programación de bajo nivel siendo estos todo lo contrario al de alto nivel ya que estos constan de un lenguaje de máquina y ensambladores que la CPU entiende.

Algunos lenguajes de programación más populares en el sector del Backend son: JavaScript, Python, Ruby, PHP, Java, C# .NET, Perl, C++, Kotlin, Scala.

Todos han sido utilizados en gran manera para cada una de sus épocas, actualmente los lenguajes que han sobresalido son JavaScript, Python y Java. Por el lado de JavaScript por su desarrollo rápido y fácil de entender, además es de código abierto, en el cual existe una inmensa comunidad de desarrolladores que comentan y ayudan en la codificación con este lenguaje, su reducción en los costos de programación ya que se puede utilizar el mismo lenguaje tanto en Frontend como en Backend, conjunto a ello permite el uso de middlewares que ayudan a resolver desafíos de desarrollo como lo es express.js donde nos ayuda a manejar peticiones con los diferentes verbos del protocolo HTTP.

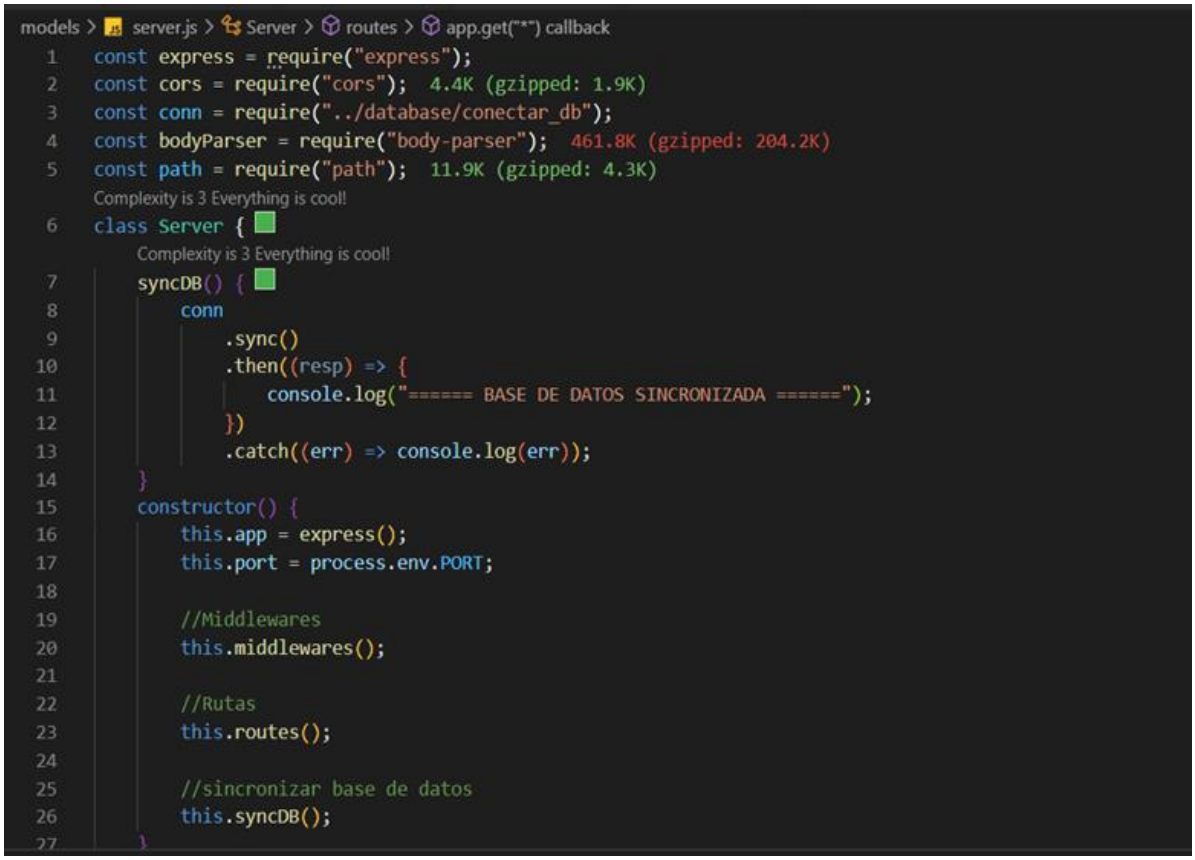
Por el lado de Python al ser un lenguaje interprete al igual que JavaScript, este es relativamente fácil de aprender y de analizar, conjunto a ellos presentan una gran variedad de librerías que ayudan demasiado para el tratamiento de datos o lo que se conoce como la ciencia de datos, siendo también de código abierto, permitiendo obtener numerosas funciones gratuitas para mejorar el rendimiento de nuestra aplicación.

Y por el lado de Java que es un lenguaje orientado a objetos con el fin de organizar la codificación en unidades denominadas clases, que crean objetos permitiendo su relación para generar la funcionalidad de la aplicación, sobresale en la tecnología de Backend porque permite ser escalable con la facilidad de admitir que el servidor ejecute varias instancias. Cabe destacar que es multihilo, lo cual permite la paralelización de código y un alto rendimiento a la hora de funcionar, su estricta seguridad también es un buen factor para incluirlo en el desarrollo Backend además de la amplia gama de librerías de código abierto que presentan.

Con relación al estudio de todas estas tecnologías de backend se ha decidido realizar el proyecto en el lenguaje de programación JavaScript ya que esta nos permite desarrollar en el mismo lenguaje tanto en backend como en frontend y con ello tener menos complicación a la hora del acoplamiento de ambas tecnologías, además cabe resaltar que en la documentación y cursos tomados para guiarnos con este desarrollo del proyecto en su gran mayoría se encuentran codificados en este lenguaje de programación del cual nos ayudamos con los frameworks de Node.js y React.js

Figura 9.

Fragmento del código Back-End



4.2.3 Comparación de Bases de datos

Se encuentran dos tipos de bases de Datos, tanto relacionales como no relacionales, Las bases de datos relacionales el manejo de datos se genera por la organización de tablas estructuradas con el cual cada fila van a obtener los mismos parámetros tengan o no valores además de obtener una llave única e irrepetible con el cual puede relacionarse con otros tipos de datos pertenecientes a esa fila la cual vienen de una segunda tabla, esto nos ayuda a mejorar la consulta de la Base de Datos cuando se llega a obtener una inmensa cantidad de datos, pero hay que tener en cuenta que

la base de datos para que no genere errores debe de estar normalizada normalmente hasta el 5 nivel donde se elimina redundancias, y se mejoran los tipos de relaciones entre las tablas, permitiendo que no se generen volcados de datos en los backups.

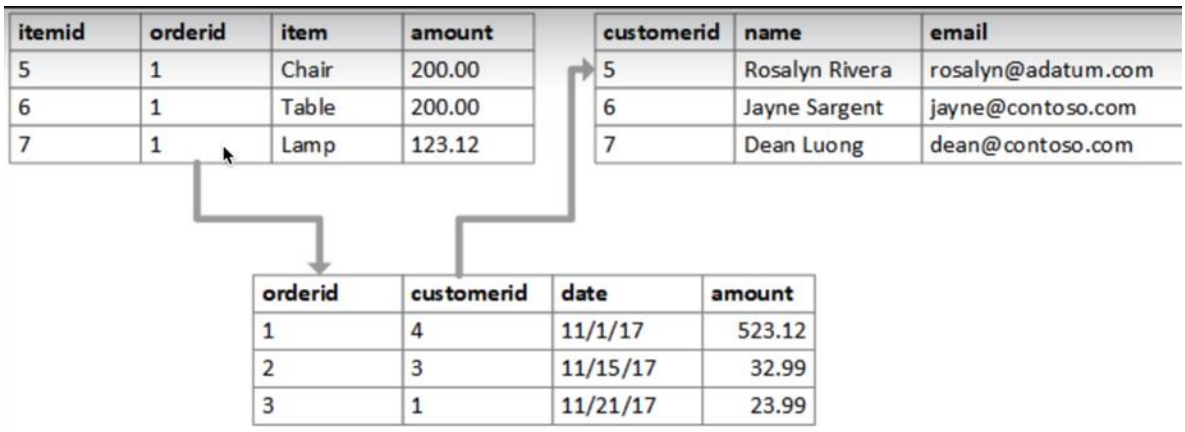
Estas bases de datos para su buen funcionamiento necesitan de un sistema gestor de bases de datos, los más conocidos a la fecha son:

- Db2: Propio de IBM
- Microsoft SQL Server: Propio de Microsoft, pero con una licencia paga.
- MySQL: Junto a PostgreSQL es de código abierto cuyo desarrollo se ocupa la comunidad de open source.
- Oracle Database: Propio de Oracle.
- SQLite: Es de Dominio Público es decir no presenta restricciones y es de uso libre.
- MariaDB: Deriva de MySQL y es de código abierto.

Todos estos gestores para poder realizas un CRUD típico lo realizan por medio de sentencias SQL (Create, Read, Update, Delete). Este tipo de bases de datos presenta sencillez en su programación, genera escasa redundancia de datos, consistencia en los datos, ya que una vez normalizadas todo esto me genera integridad en los mismos, Se pueden realizar procesamiento de conjuntos de datos por medio de las sentencias JOIN y como lo dicho anteriormente mantienen un lenguaje homogéneo para consultas que es SQL

Figura 10.

Estructura de base de datos Relacional



itemid	orderid	item	amount
5	1	Chair	200.00
6	1	Table	200.00
7	1	Lamp	123.12

customerid	name	email
5	Rosalyn Rivera	rosalyn@adatum.com
6	Jayne Sargent	jayne@contoso.com
7	Dean Luong	dean@contoso.com

orderid	customerid	date	amount
1	4	11/1/17	523.12
2	3	11/15/17	32.99
3	1	11/21/17	23.99

Por otro lado, las bases de datos no relacionales son lo opuesto a las no relacionales ya que acá no existe el concepto de tablas sino el concepto de documentos ya que todos los datos se almacenan en archivos de texto plano, en formato JSON que se guardan por medio de objetos, en este tipo de base de datos no existe las relaciones entre documentos, pero el manejo de datos es muy bueno a la hora de recibir toneladas de información, “a menudo hacen concesiones al flexibilizar algunas de las propiedades ACID de las bases de datos relacionales para un modelo de datos más flexible que puede escalar horizontalmente. Esto hace que las bases de datos NoSQL sean una excelente opción para casos de uso de baja latencia y alto rendimiento que necesitan escalar horizontalmente más allá de las limitaciones de una sola instancia.” (Amazon, 2021)

Por ello como lo dicho anteriormente una base de fatos No relacional no requiere de estructuras de datos fijas como las tablas, además no garantiza en tu totalidad las características de ACID y su escalamiento se da muy bien en forma horizontal. Algunos de los sistemas gestores más utilizados que permiten el funcionamiento de estas bases de datos son:

- MongoDB.

- Redis.
- Cassandra.
- Azure Cosmos DB.
- RavenDB.
- ObjectDB.
- Apache CouchDB.
- Neo4j.
- Google BigTable.
- Apache Hbase.
- Amazon DynamoDB.

Conociendo la escalabilidad que se tiene pensado para el proyecto, y que el uso de los dispositivos por alumbrado público me va a generar una masiva concentración de datos para su mejor funcionamiento y posterior tratamiento de datos se decidió utilizar una base de datos relacional con el sistema gestor MySQL y que su uso en muy flexible y la localización de documentación de la misma con la ayuda de tutoriales realizados nos dio de mayor agrado, conociendo que su uso para la tecnología Backend anteriormente utilizada y su enlace con el lenguaje de programación JavaScript se observó que la compatibilidad aunque no es muy flexible se puede realizar sin problema alguno

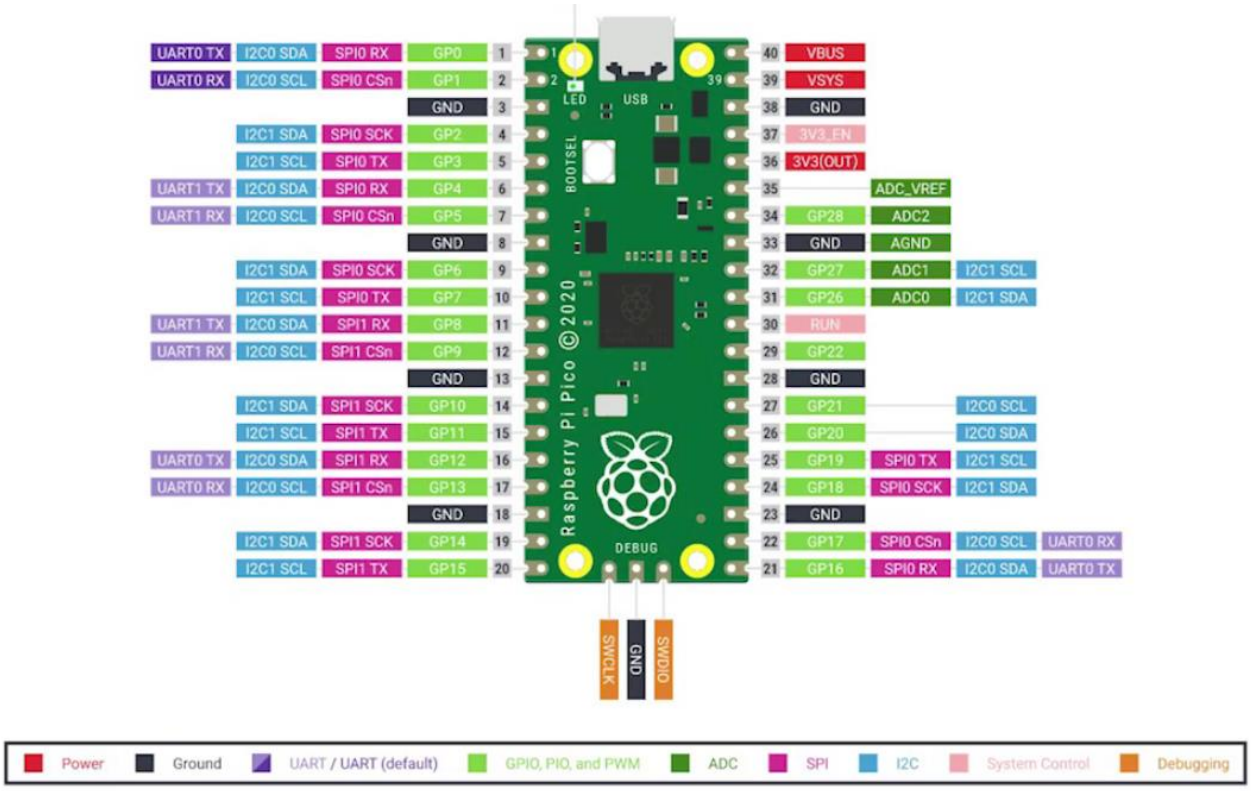
4.2.4 Comparación de dispositivos y sensores necesarios para el hardware

Teniendo en cuenta los dispositivos que actualmente se manejan para el IOT, ya sea Arduino nano, Raspberry Pi Pico y ESP32.

- *Raspberry Pi Pico:* Construido con el chip de microcontrolador RP2040, cuenta con un procesador Arm Cortex-M0+ de doble núcleo con 264 KB de RAM interna y 16 MB de Flash fuera del chip. Incluye dobles puertos I2C, SPI, UART con 16 canales PWM, 8 máquinas de estado de PIO y 26 pines GPIO de los cuales 3 son entradas analógicas, cuenta con 12 canales DMA permitiendo mover los datos de un lado a otro de manera muy rápida, lo que lo diferencia de los demás dispositivos es que para poder manejar todas estas características el nivel de consumo que exige es superior a los demás módulos por ende no es recomendable usarlo por medio de baterías, su fuente de trabajo es de 3.3V y el voltaje de entrada es de 1.8-5.5V, no cuenta con conectividad WiFi no Bluetooth, su valor promedio es de aproximado 4 a 5 dólares. Como es el primer microcontrolador que lanza Raspberry pico, su comunidad de desarrolladores es reducida.

Figura 11.

Pines Raspberry Pi Pico

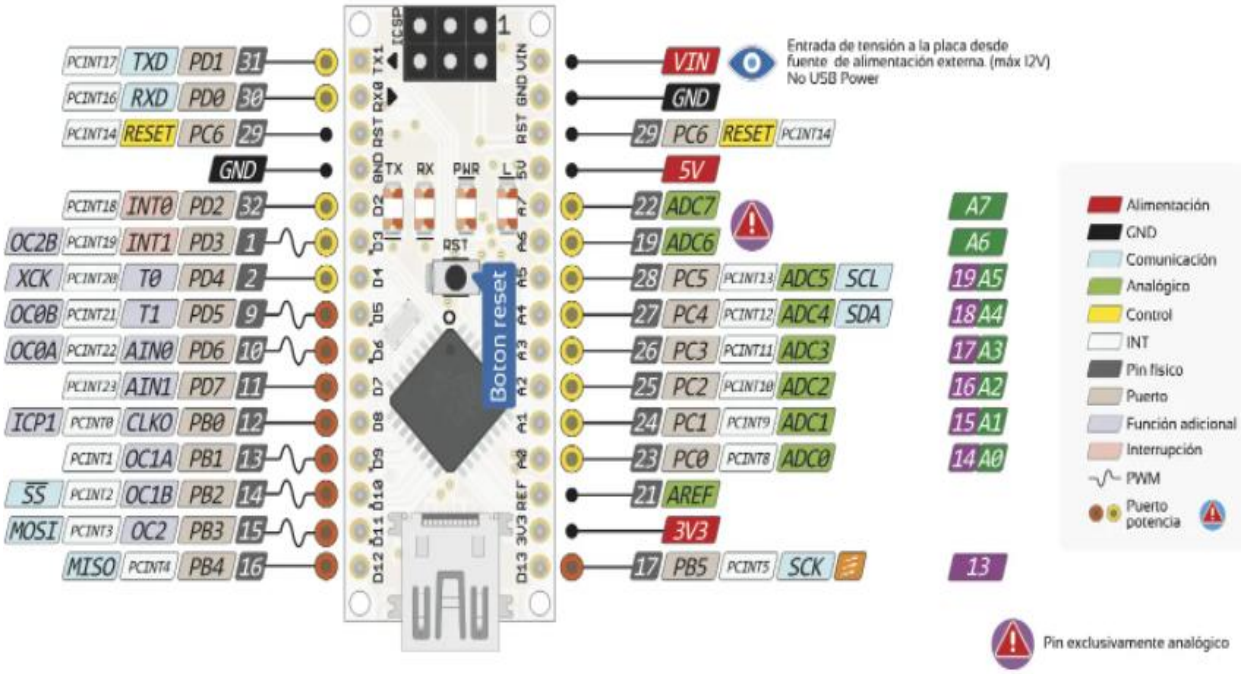


Nota: Tomado de YouTube (s.f.) Recuperado de. https://www.youtube.com/watch?v=vHZyRPHVz0E minuto 1:49

- *Arduino Nano:* Su programación está basado en el microcontrolador ATmega328P a 16Mhz, tiene 14 pines de entrada y salida digital de estos seis pueden ser usados para modulación *por* ancho de pulsos o por sus sigras PWM, 8 entradas analógicas, terminales para conexión ICSP y maneja un puerto UART, SPI, e I2C, su voltaje de trabajo es de 5v y su voltaje de entra varia de 5 a 12v. Tiene un costo aproximado de 20 dolares, ya que cuenta con muchos años en el mercado de los microcontroladores, maneja una amplia gama de comunidades para el desarrollo de proyectos.

Figura 12.

Pines Arduino nano

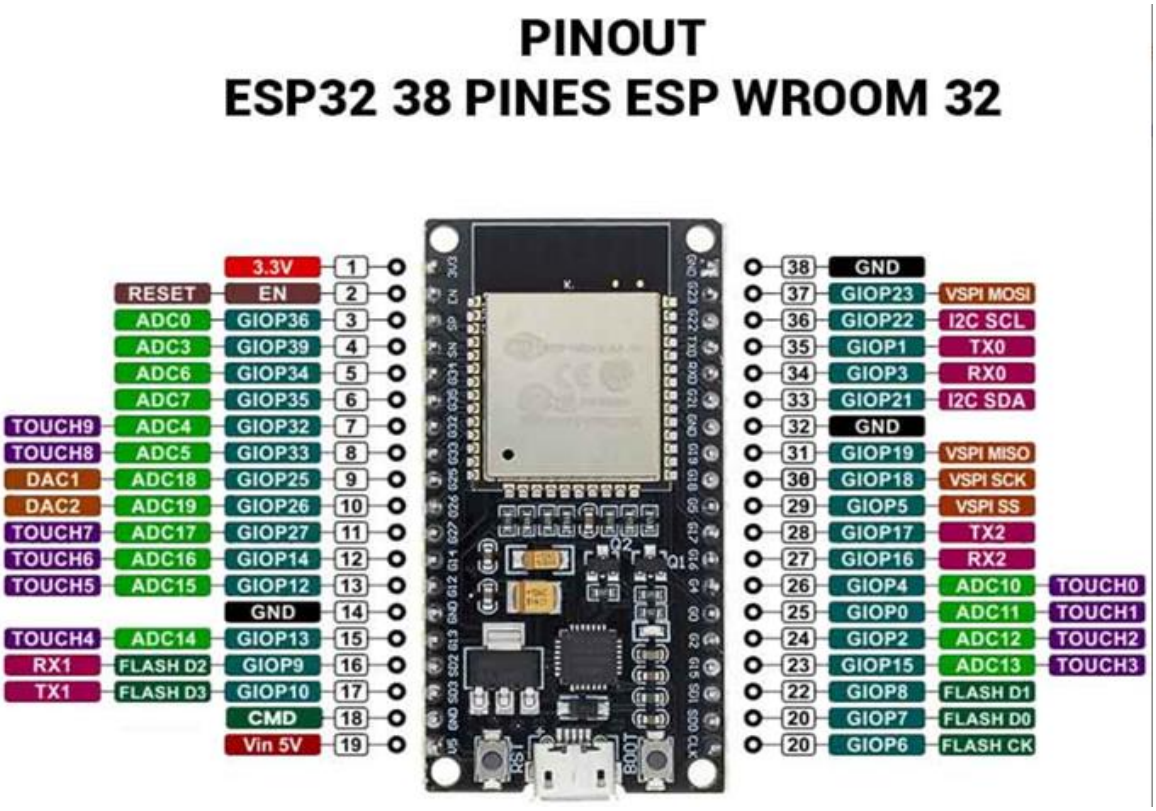


Nota: Tomado de: Arduino (s.f.) Arduino nano Recuperado de: https://arduino.cl/arduino-nano/

- *ESP32:* Maneja un procesador Xtensa LX6 de 32 bits de doble núcleo normalmente a entre 40 a 160MHz, pero puede llegar a los 240MHz, soportando temperaturas de -40 °C y 125 °C, cuenta con 448KiB de ROM, 520 KiB de SRAM para instrucciones y datos, 4MiB Flash, cuenta con modulo para WiFi y Bluetooth, tiene 36 pines GPIO, 16 canales de ADC de 12 bits de resolución (2 ADC), 10 Pines Touch, 16 canales de PWM, 2 de UART o puertas serie, 2 de I2C y 4 de SPI, Alimentación a 3,3V, su costo varía de 4 a 5 dólares. Los pines de este dispositivo se encuentran distribuidos de la siguiente manera:

Figura 13.

Pines del ESP32



Nota. Tomado de: U electronics (s.f.) ESO 32-38 pines ESP Wroom Recuperado de: https://uelectronics.com/producto/esp32-38-pines-esp-wroom-32/

Su programación se puede realizar muy bien con el IDE de Arduino, solo hay que agregarles las librerías del ESP32 y cargarle su respectiva placa para que pueda tener compatibilidad y subirle el programa a la plata respectiva. Y al gestor de URLs adicionales de tarjeta le agregamos la siguiente línea << https://dl.espressif.com/dl/package_esp32_index.json>> con el cual permitirá gestionar nuestra placa del esp32.

Para manejos de librería y cuestiones de costos decidimos elegir el ESP32, además de la cantidad de documentación que existe, con las funciones que cumple es suficiente para la realización de nuestro proyecto.

Para poder seleccionar el sensor que nos va a permitir medir el consumo de corriente en nuestro sistema se estudiaron diferentes dispositivos los cuales eran compatibles con Arduino además que generaban datos de salidas digitales como:

- Sensor de corriente Acs712
- Sensor de voltaje y corriente INA3221
- Sensor de corriente INA219 I2C
- Sensor de corriente INA226

Tabla 1.

Precios de los sensores de corriente

Referencia del Sensor	Precio
ACS712	$ 13000 COP
INA3221	$ 13000 COP
INA219 I2C	$ 12000 COP
INA226	$ 23000 COP

Nota: estos precios fueron un promedio sacado de mercadolibre Colombia

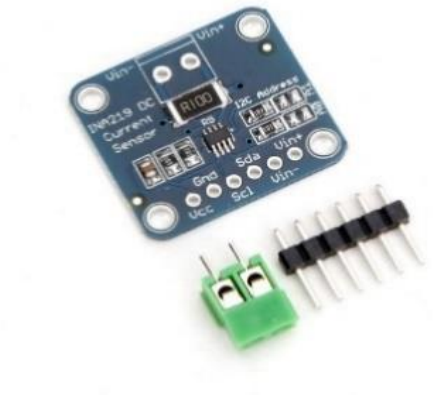
Se pudo apreciar que el costo de estos sensores y sus características eran apropiadas para el proyecto, pero uno sobresalió más con respecto a las librerías que tiene conformadas en el programa de Arduino, el cual hace que su uso sea más prolijo y por ende se pueda facilitar el manejo de datos con este sensor.

Las características que este sensor nos presentan son:

- Su envío de datos de manera bidireccional se basa en una interfaz I2C.
- Detecta Corriente, Voltaje y Potencia al mismo tiempo.
- Cuenta con una resistencia de derivación al 1%.
- Su rango de detección de corriente es de más o menos 3.2 A
- Mide voltaje en DC hasta +26 V.
- Su voltaje de alimentación es de 3V a 5V.
- Mide potencias de hasta 83.2 W.
- Puede operar a temperaturas de – 40°C a 125°C.

Figura 14.

Sensor INA219



Nota. Tomado de: Electronilab (s.f.) Recuperado de: https://electronilab.co/

4.3 Diseño del Hardware y Software

4.3.1 Definición de requerimientos funcionales, no funcionales y roles

4.3.1.1 Roles. Se definieron unos roles para los usuarios de la plataforma, los cuales tienen unas funciones determinadas relacionadas con su cargo en el flujo de actividades de la gestión del alumbrado público

Tabla 2.

Descripción de los roles de la plataforma

Rol	Descripción
Administrador	Tiene acceso a todos los módulos del sistema
Líder de cuadrilla	Puede ver la ubicación de los dispositivos y los valores que se están registrando en tiempo real, además puede ver la información de su cuadrilla y confirmar el arreglo de un fallo
Empleado	Puede ver la información de ubicación del dispositivo y su detalle, Puede ver la información de la cuadrilla a la que pertenece

4.3.1.2 Requerimientos funcionales. Para el desarrollo de esta plataforma se describieron una serie de requerimientos mínimos para el correcto funcionamiento del sistema.

Tabla 3.

Definición de requerimientos funcionales y roles permitidos

No.	Descripción	Roles
RF-1	El sistema debe permitir iniciar sesión con correo electrónico y contraseña, y validar las transacciones efectuadas durante la sesión por medio de tokens	Administrador Líder de escuadrón, Empleado
RF-2	Se debe ver el consumo promedio de corriente, voltaje y potencia, por día en un rango de fecha determinado	Administrador
RF-3	Se podrá obtener el tiempo de respuesta promedio de cada cuadrilla en un rango de tiempo determinado	Administrador
RF-4	Se deben observar el número de fallos por sector en un rango de tiempo determinado	Administrador
RF-5	Se podrá obtener un top 10 de los dispositivos que presentaron más fallos en un rango de tiempo determinado	Administrador
RF-6	El sistema debe mostrar el total de fallos agrupados por tipo y día en un rango de tiempo	Administrador
RF-7	Se requiere generar reporte con detalle de las fallas y consumo en un archivo Excel	Administrador
RF-8	El sistema debe permitir añadir nuevos dispositivos, y recolectar los datos de geolocalización	Administrador
RF-9	Se debe mostrar la ubicación geográfica de los dispositivos y su información	Administrador Líder de escuadrón, Empleado
RF-10	El sistema mostrará los valores de consumo de corriente, potencia y voltaje por cada dispositivo en tiempo real	Administrador Líder de escuadrón
RF-11	El sistema notificará por correo electrónico a la cuadrilla encargada de un sector cuando un dispositivo dentro de este presente un fallo	Líder de escuadrón Empleado
RF-12	Se requiere mostrar los valores de consumo de un dispositivo en las últimas 24 horas	Administrador Líder de escuadrón

No.	Descripción	Roles
RF-13	Se debe mostrar la información de cuadrillas y de sus miembros, se podrá filtrar por comuna	Administrador Líder de escuadrón, Empleado
RF-14	Se deben mostrar los fallos pendientes por arreglo de cada cuadrilla	Administrador Líder de escuadrón, Empleado
RF-15	Se podrá marcar como arreglado un fallo, el líder de cuadrilla solo podrá acceder a la información de su cuadrilla, mientras el administrador accede a la de todas	Administrador Líder de escuadrón
RF-16	El sistema permitirá la creación de usuarios y la asignación de un rol	Administrador
RF-17	Se podrá crear un empleado asociado a un usuario previamente creado	Administrador
RF-18	Se deben visualizar los usuarios del sistema con su respectiva información	Administrador
RF-19	El sistema debe registrar los fallos en el momento que se produzcan y actualizar el estado del dispositivo que falló	Ningún usuario
RF-20	El sistema debe guardar los valores de consumo de los dispositivos cada 15 segundos	Ningún usuario

4.3.1.3 Requerimientos no funcionales. Para un funcionamiento óptimo de la plataforma IOT se tuvieron en cuenta los siguientes requerimientos no funcionales del sistema

Tabla 4.

Definición de requerimientos no funcionales

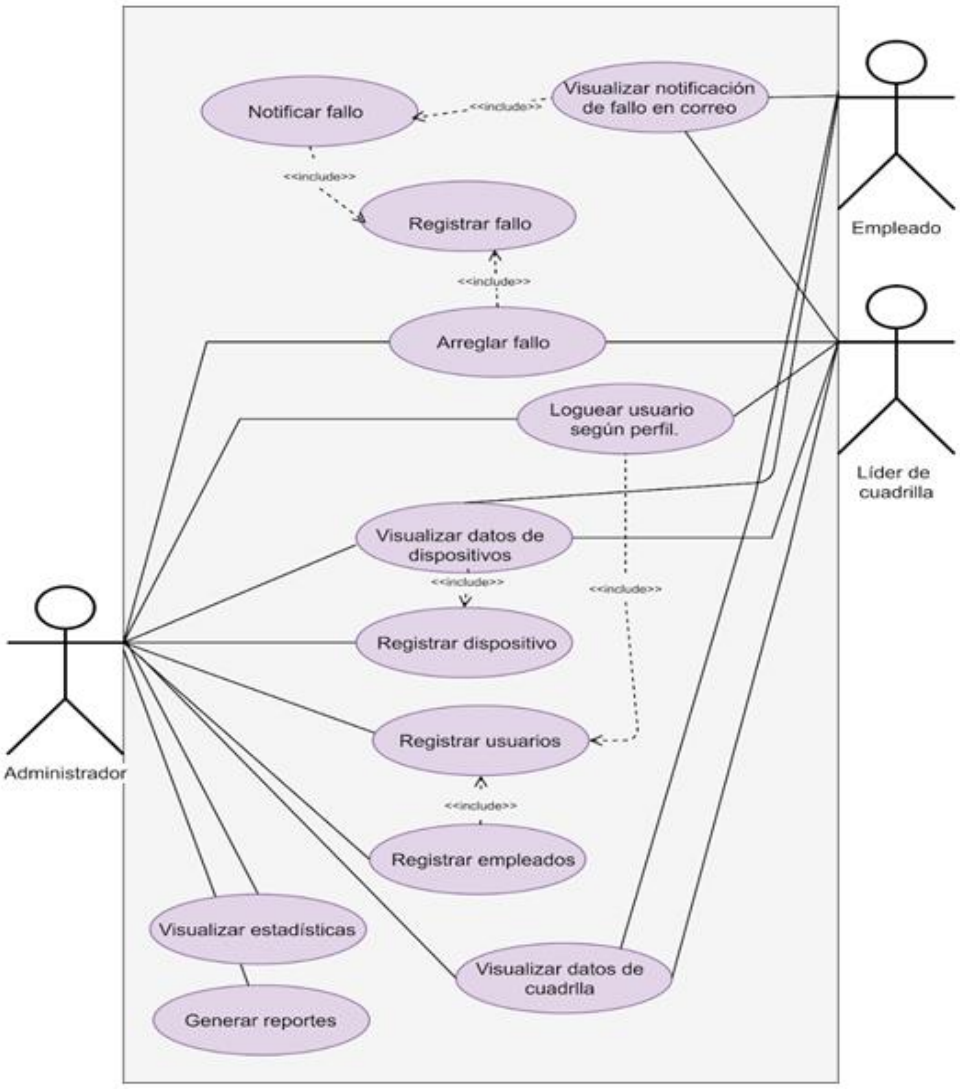
No.	Nombre	Descripción
RNF-1	Escalabilidad	• Diseñar la arquitectura de manera que los servicios y vistas sean modulares e independientes • Permitir que sea escalable a otras arquitecturas IOT.
RNF-2	Usabilidad	• Su interfaz será agradable y de fácil uso para los usuarios.
RNF-3	Soporte	• Será soportado en diferentes navegadores (Chrome, Mozilla)
RNF-4	Seguridad	• Los datos sólo se mostrarán a usuarios autorizados. • Las transacciones serán validadas por medio de tokens • La plataforma deberá contar con los certificados SSL • Las contraseñas están encriptadas
RNF-5	Conectividad	• Se requiere conexión wifi para su funcionamiento. • Se requiere una cuenta en proveedor de Cloud Computing.

4.3.2 Casos de uso

Se construyó un diagrama de casos de uso con el propósito de determinar el comportamiento del prototipo con base en la interacción con los usuarios. Tomamos los roles previamente definidos para identificar las acciones que podrán realzar en el sistema.

Figura 15.

Diagrama de casos de uso



4.3.3 Diagramas de procesos

Se realizaron 4 diagramas de procesos para las principales secuencias de actividades que se realizaran en el aplicativo, con el fin de definir el flujo de los eventos e interacciones entre los distintos componentes de la plataforma

Figura 16.

Diagrama de proceso de registro de valores del dispositivo

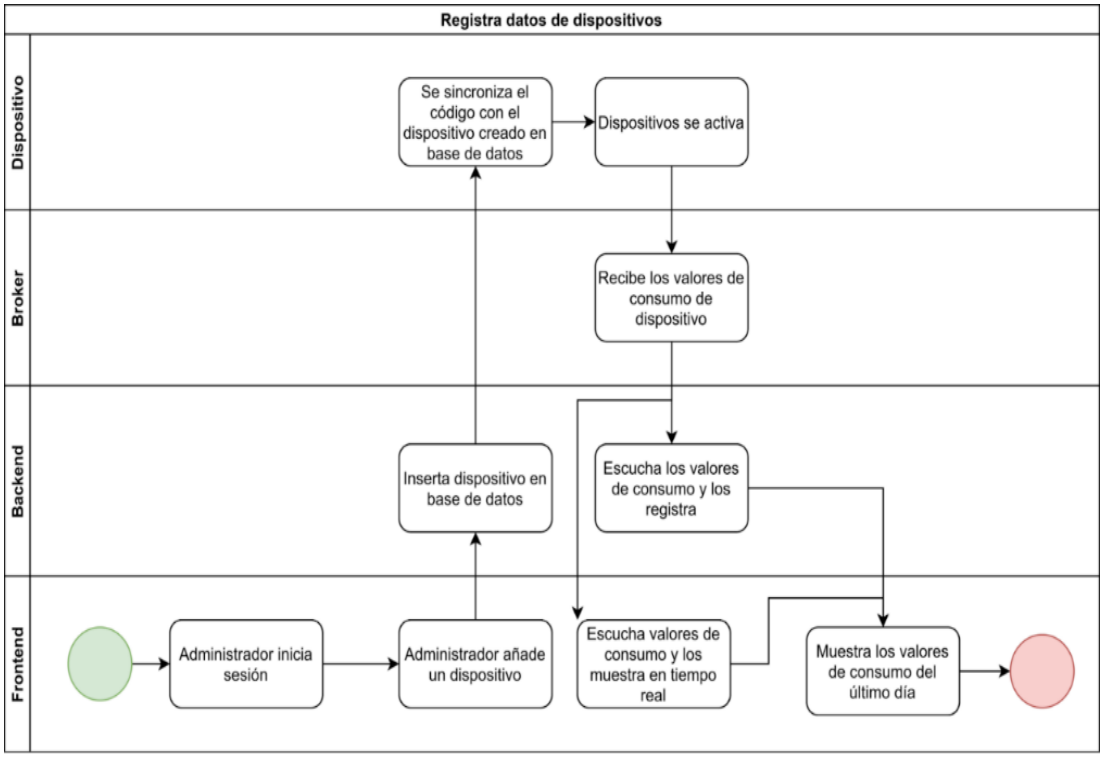


Figura 17.

Diagrama de proceso de gestión de usuarios

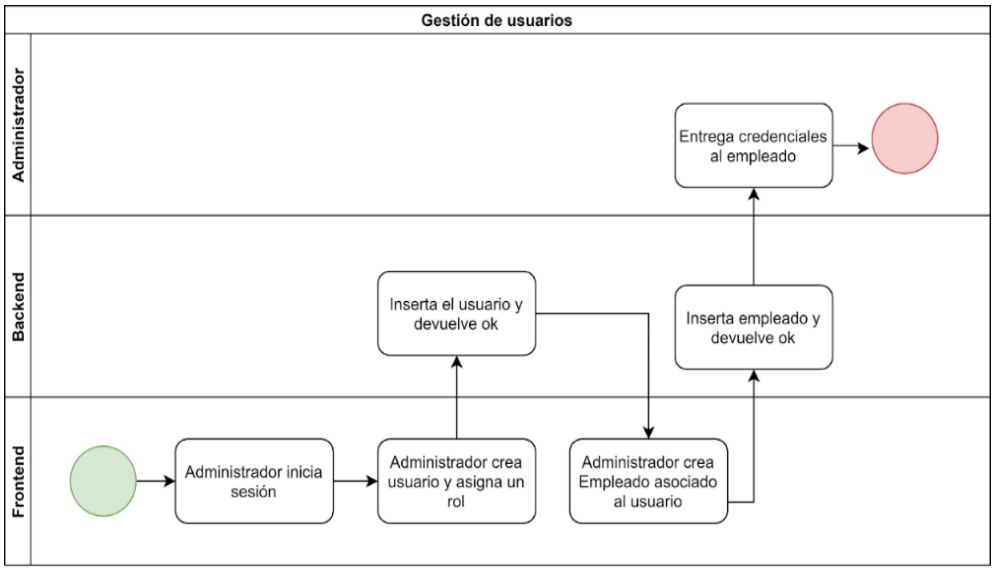


Figura 18.

Diagrama de proceso de reporte de fallos

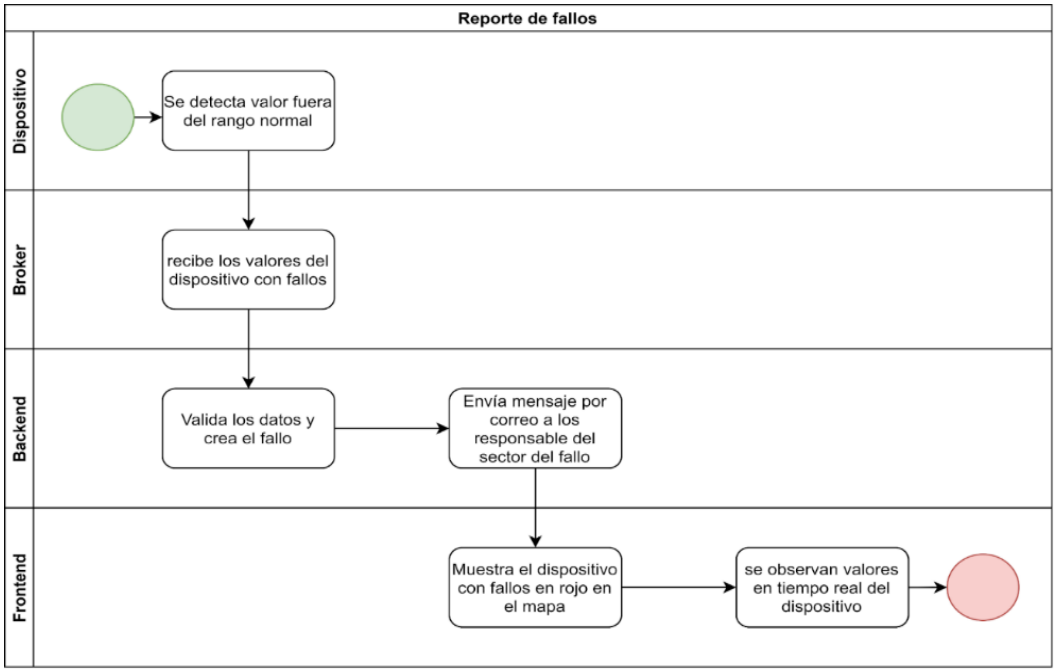
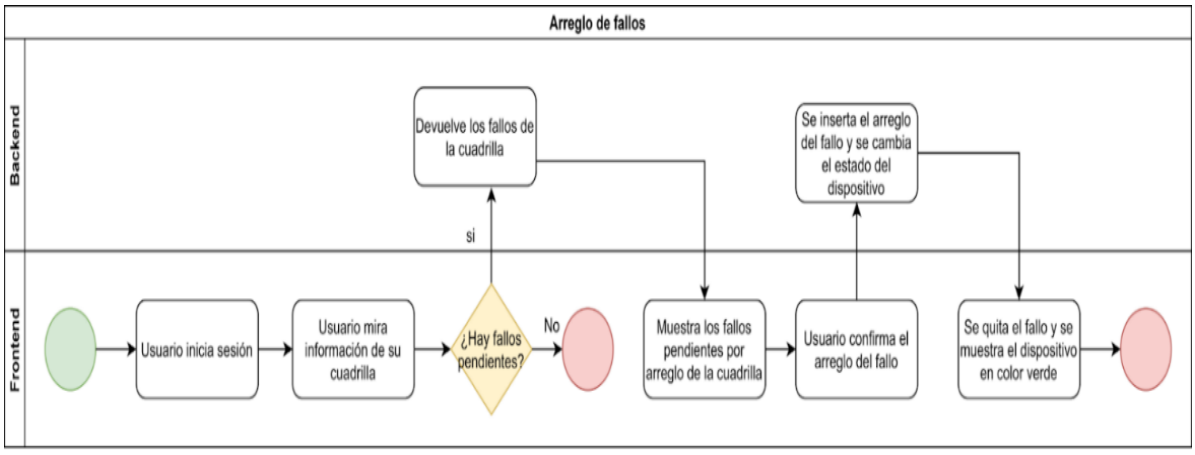


Figura 19.

Diagrama de proceso de arreglo de fallos

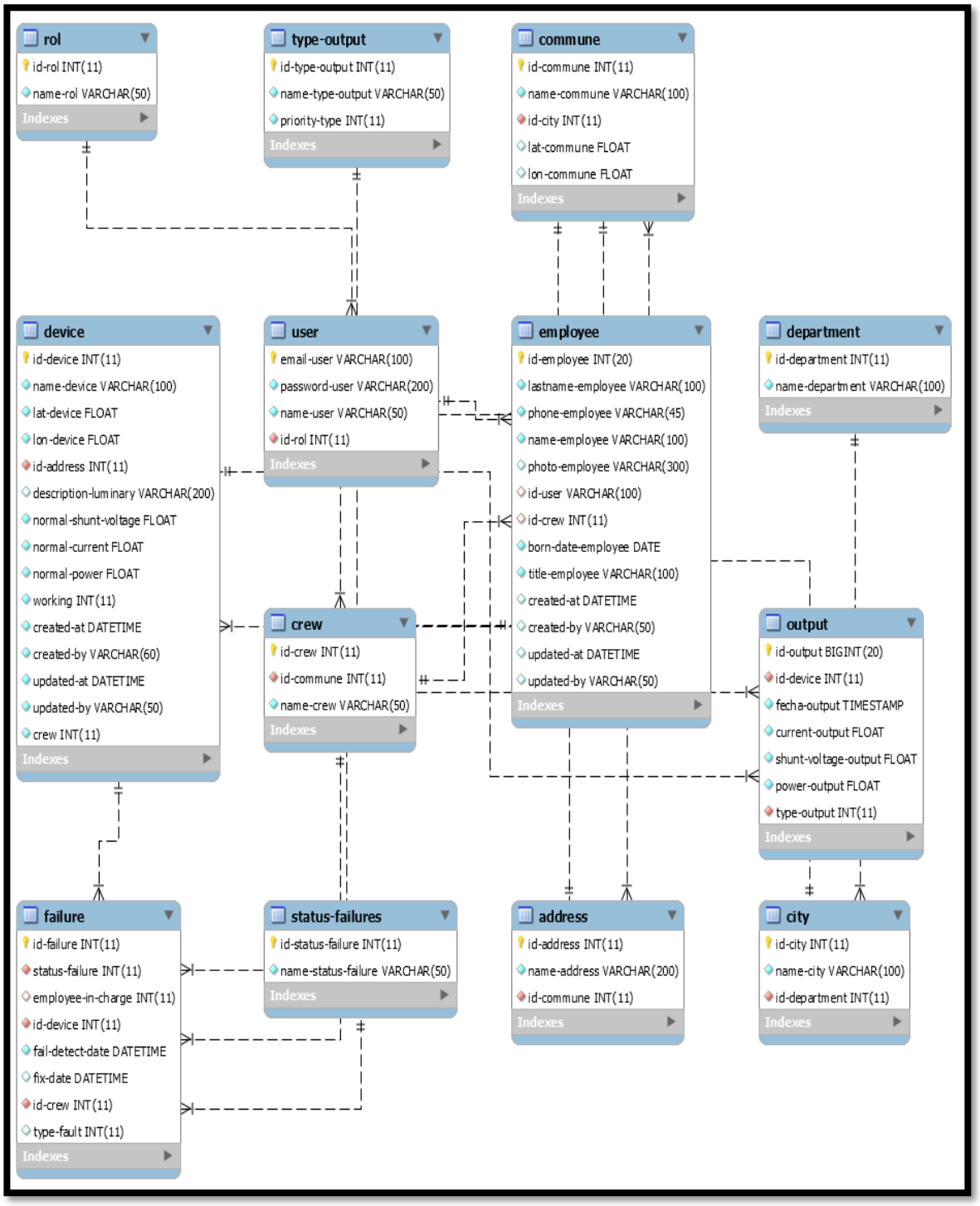


4.3.4 Modelo de datos

En esta base de datos se almacena la información de los usuarios que usan el sistema, los dispositivos físicos y su ubicación, además de los datos que capturan estos. Se utilizó el motor de base de datos MySQL, el cual es de libre distribución y de código abierto.

Figura 20.

Modelo de datos



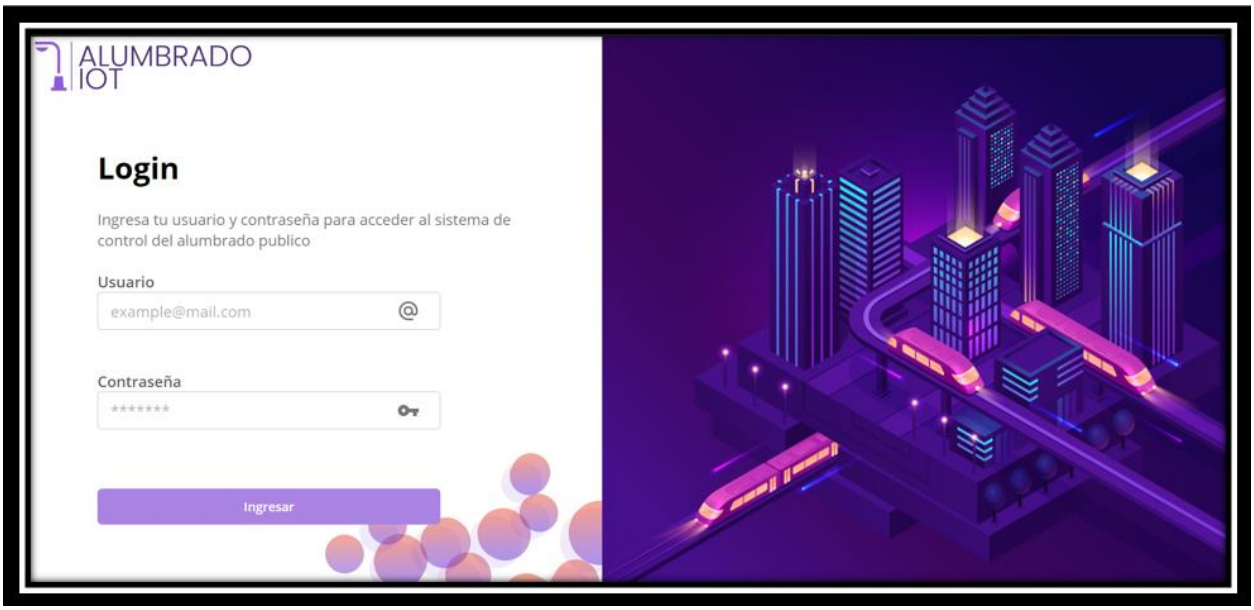
4.3.5 Diseño del Mockup

Después de definir los requerimientos del sistema y los procesos principales, se procedió a diseñar la interfaz del sistema, a continuación de muestran las vistas de cada módulo que conforma el aplicativo

4.3.5.1 Modulo de login. Este módulo es el inicial, que cualquier cliente puede visualizar aún sin estar autenticado

Figura 21.

Vista del login



4.3.5.2 Modulo de Dashboard. Esta sección es exclusiva para el administrador de la plataforma en la cual se visualizan estadísticas basadas en los datos recolectados por los

dispositivos, además permite generar un reporte con el detalle del consumo por dispositivo y los fallos registrados filtrados por fecha.

Figura 22.

Vista del dashboard – filtros

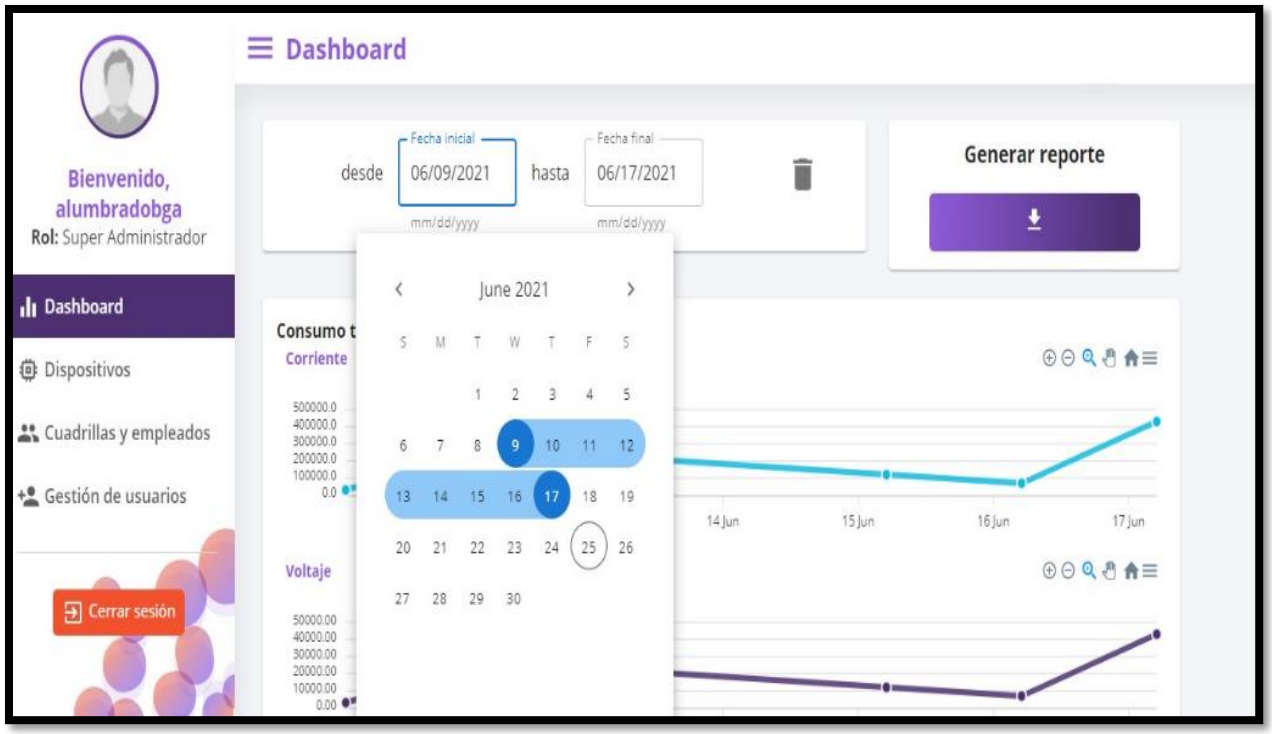
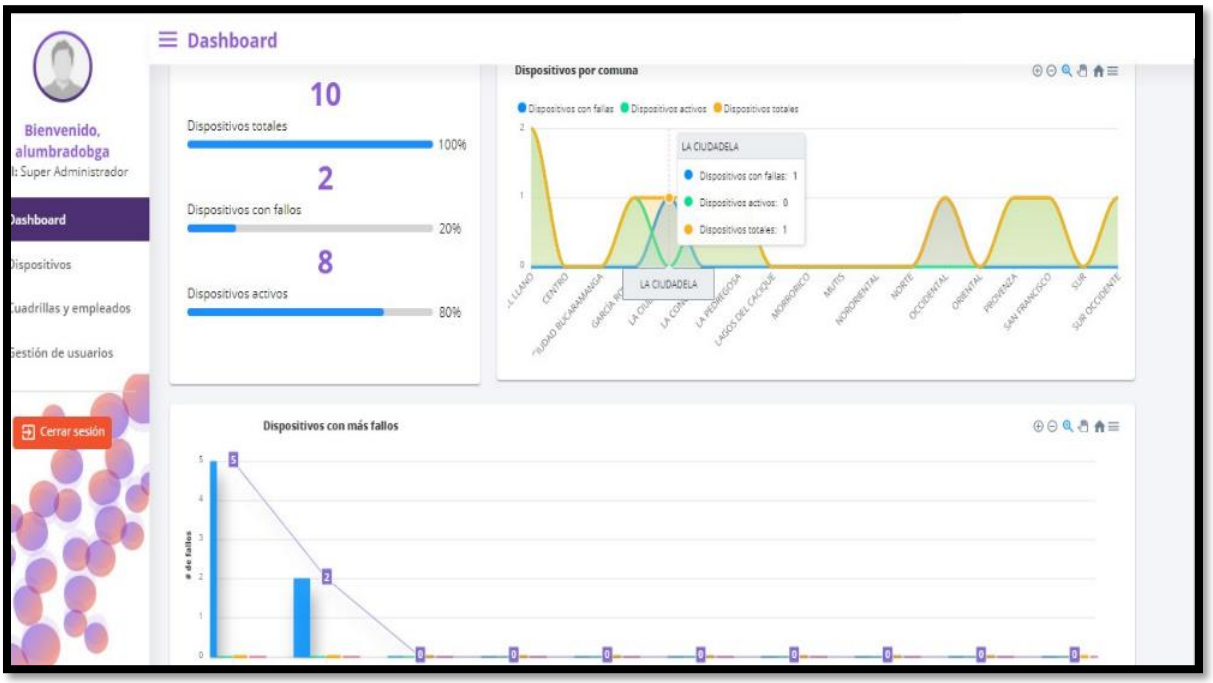


Figura 23.

Vista del dashboard – estadísticas



4.3.5.3 Modulo de Dispositivos. La vista de dispositivos permite observar los valores en tiempo real de un dispositivo, además de observar la geolocalización de estos en el mapa. Si el usuario autenticado es un administrador tendrá la opción de agregar dispositivos

Figura 24.

Vista de dispositivos - valores de consumo

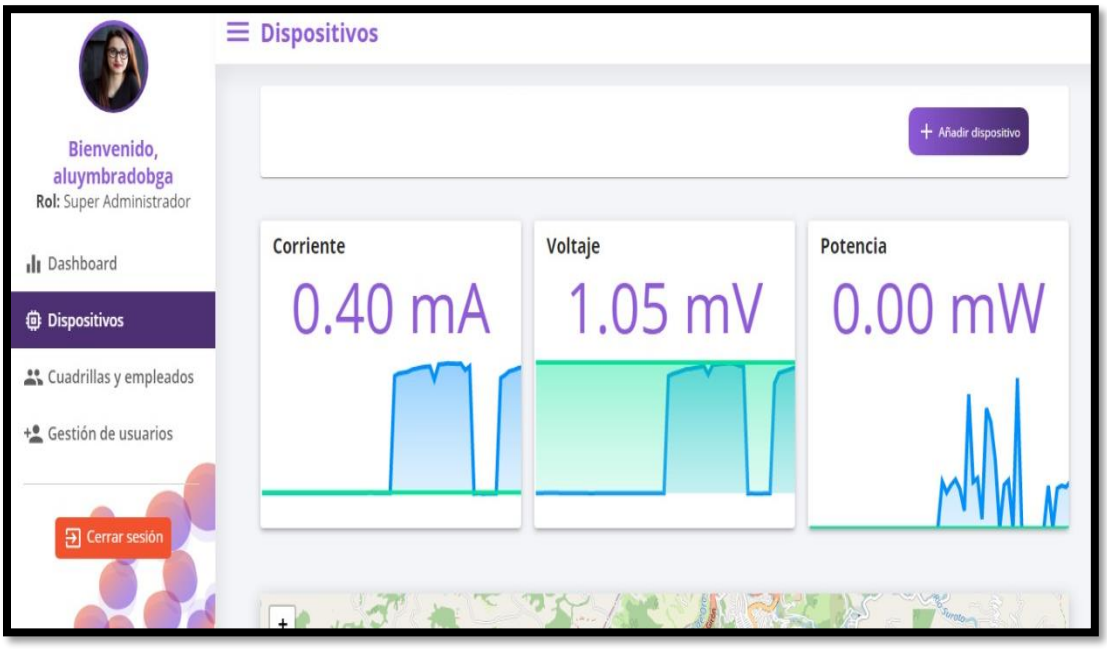
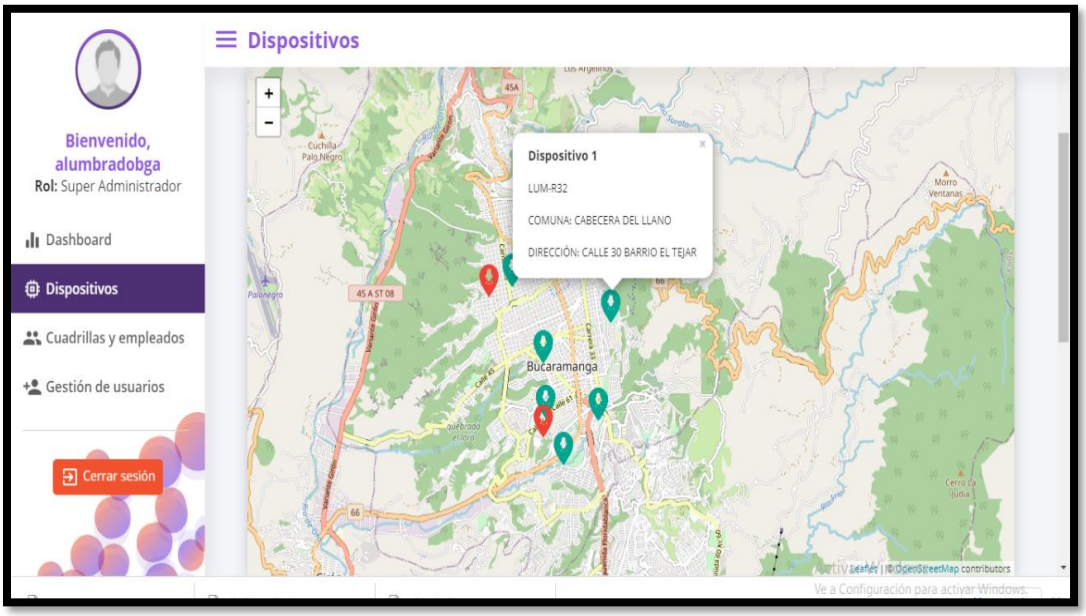


Figura 25.

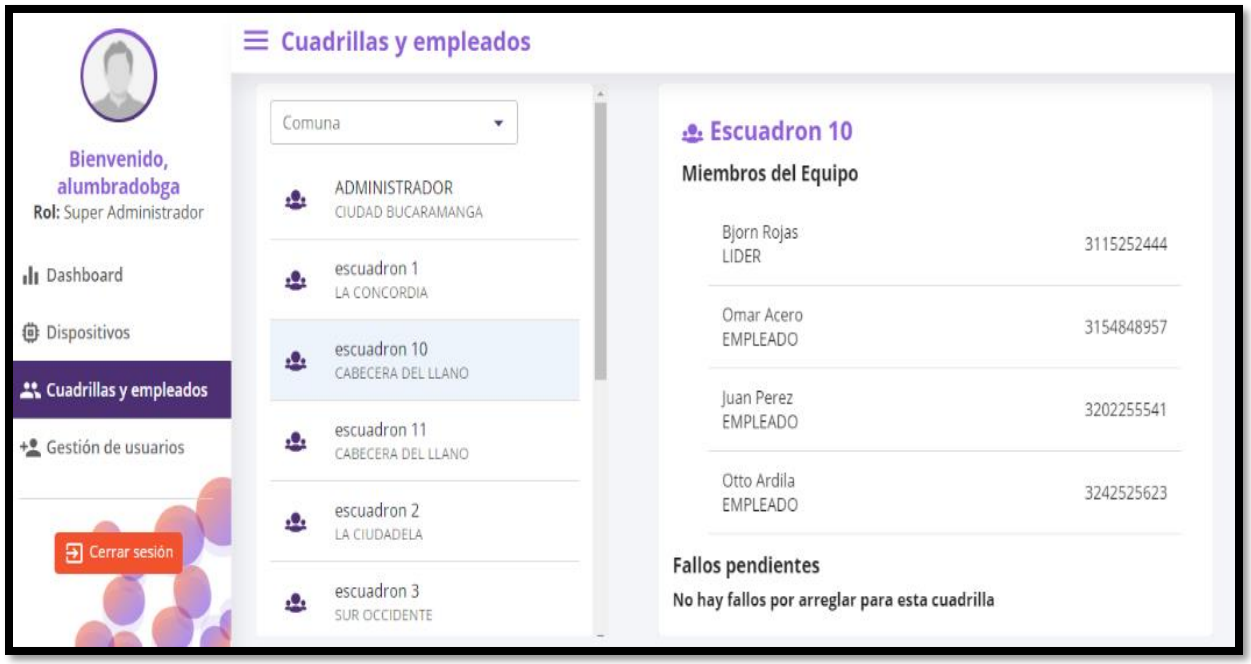
Vista de dispositivos – Mapa



4.3.5.4 Modulo de Cuadrillas. Módulo cuyo objetivo es visualizar las cuadrillas existentes y sus fallos pendientes, el líder de cuadrilla podrá confirmar el arreglo de un fallo pendiente.

Figura 26.

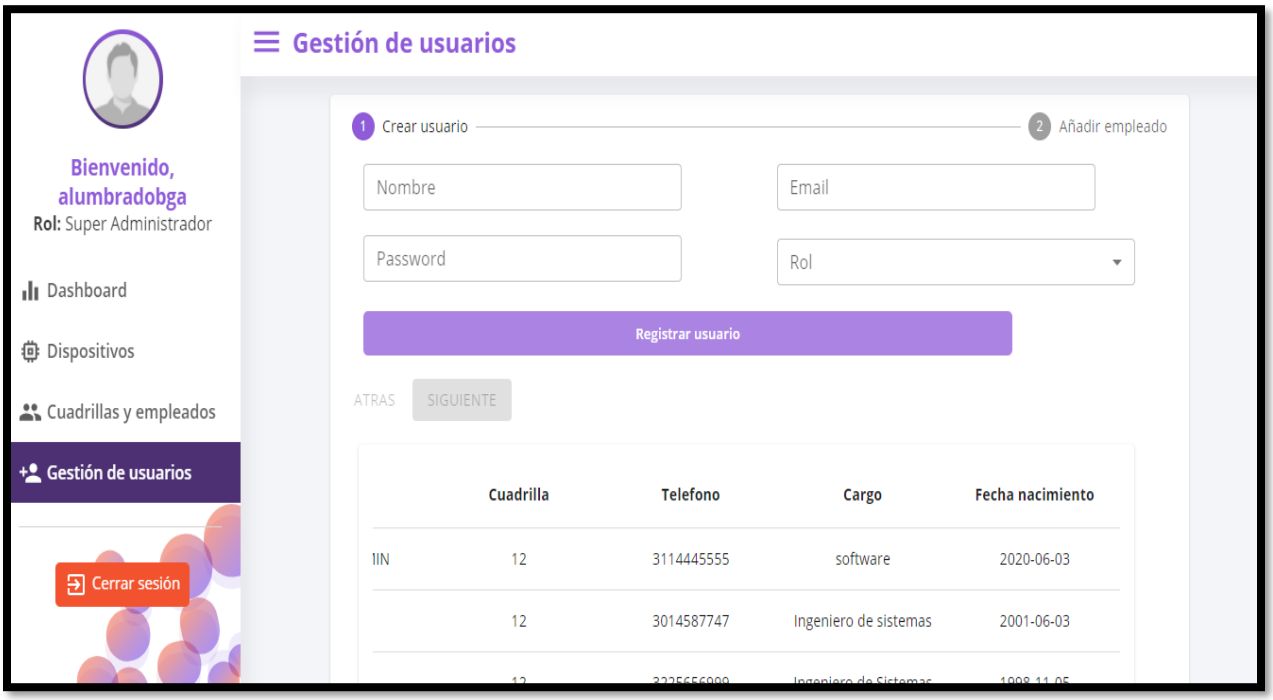
Vista de cuadrillas



4.3.5.5 Modulo de Gestión de usuarios. Vista exclusiva para el administrador la cual permite la creación de usuarios para el uso del sistema y la visualización de estos. Se puede asignar un empleado al usuario creado.

Figura 27.

Vista de gestión de usuarios

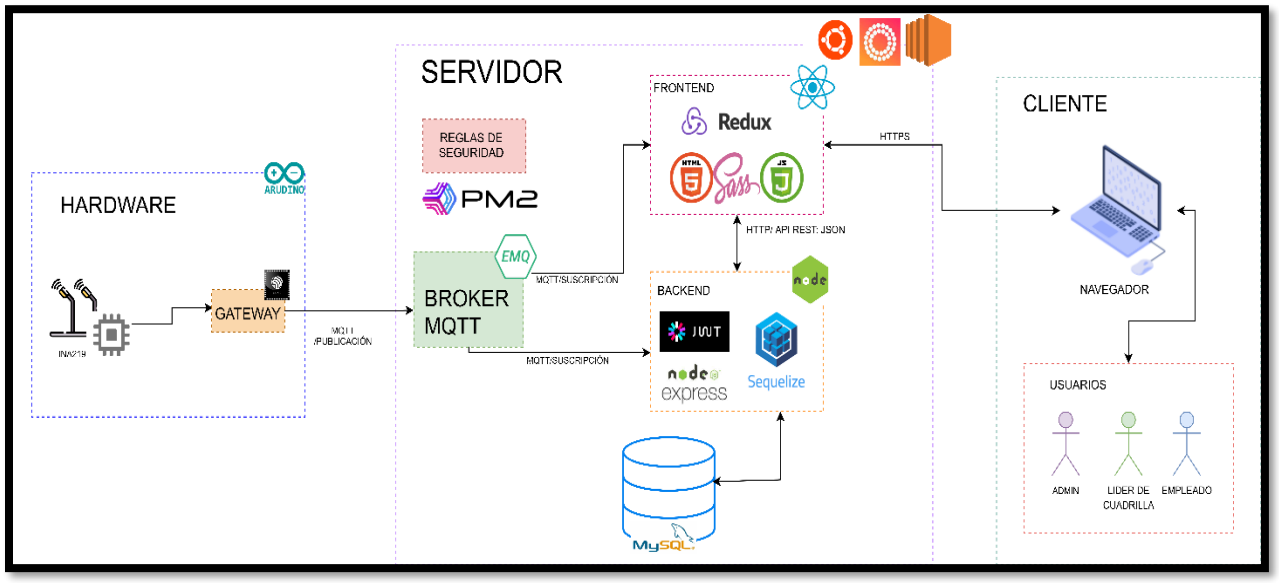


4.3.6 Diseño de la arquitectura

Continuando con el flujo de actividades, luego de diseñar los mockups y definir requerimientos se procede a diseñar la arquitectura del sistema, para posteriormente realizar un prototipo de la misma, verificar el cumplimiento de los objetivos planteados y, repetir el ciclo revisando y haciendo mejoras en caso de ser necesario a la arquitectura de la solución.

Figura 28.

Diagrama de arquitectura de la plataforma



El diagrama está conformado por 3 módulos principales, los cuales son:

- Hardware: donde se encuentran los actuadores, sensores y el Gateway
- Servidor: en el cual se ubica el broker, el frontend, el backend y la base de datos
- Cliente: en este encontramos los usuarios del sistema y el navegador

A continuación, se mostrará más a profundidad cada una de estas secciones

4.3.6.1 Arquitectura Backend. La arquitectura propuesta consta de 5 módulo principales que interactúan entre sí para el funcionamiento óptimo de los servicios REST de este Backend. Los módulos son:

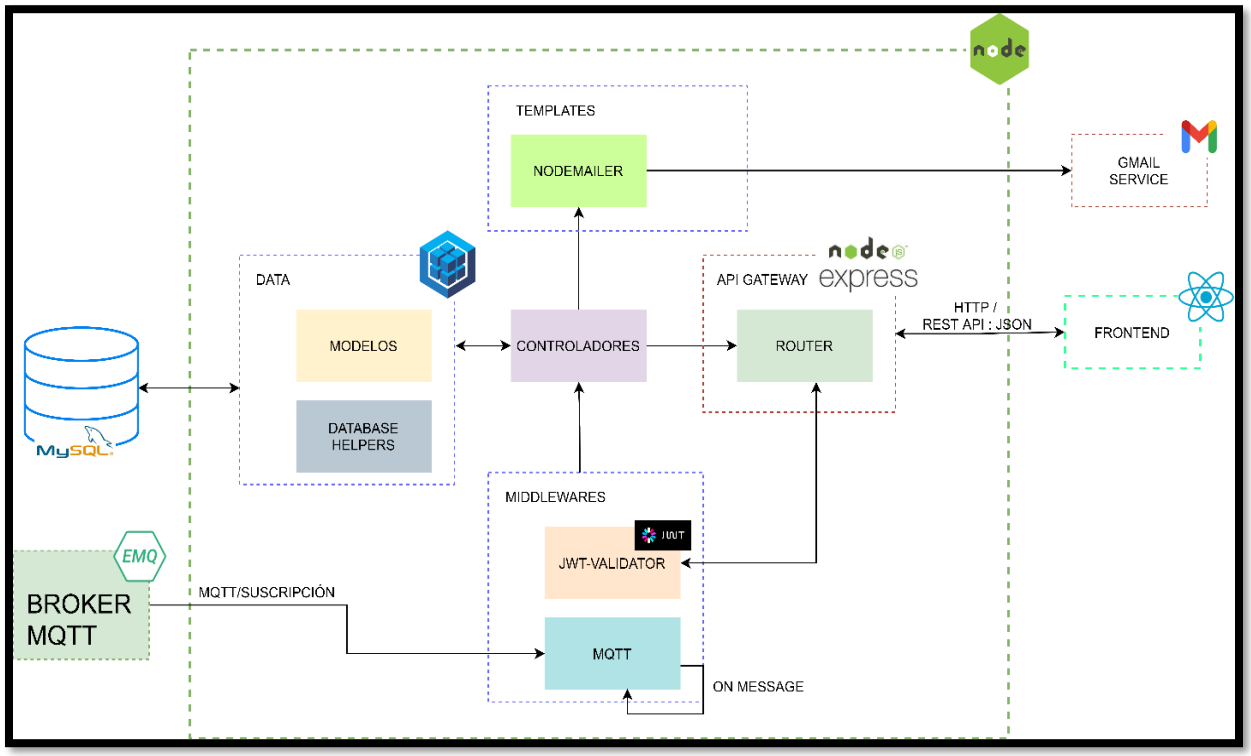
- Data: En esta se hace la conexión a la base de datos con la librería Sequelize que es un ORM para Nodejs que permite agilizar bastante los desarrollos que incluyan bases de datos

relacionales como MySQL o Postgress. Se encuentran 2 formas de realizar la petición a la base de datos, una por medio de los modelos definidos con sequelize y otra por medio de querys.

- API Gateway: En esta sección se define las rutas de los servicios con la librería express, cada endpoint está relacionado con un controller y cada controller se relaciona con un endpoint. Este módulo es el encargado de recibir las peticiones provenientes del Frontend
- Controllers: Son los que se encargan de la lógica del negocio, donde se valida el rol, se manejan los errores y se construye la respuesta.
- Middlewares: En este módulo se encuentran 2 tipos de middlewares, uno se encarga de la validación del token con JWT para los servicios que lo requieren y el otro se conecta al broker ejecutando un callback cada vez que se recibe un nuevo mensaje al tópico suscrito en el momento que se inicia la aplicación. Este callback llama a un controller que revisa los valores para posteriormente insertarlos en la base de datos.
- Templates: Plantilla para enviar la notificación al correo electrónico de los integrantes de la cuadrilla a cargo del dispositivo que presente fallos

Figura 29.

Diagrama de arquitectura backend



4.3.6.2 Arquitectura Frontend. Para la arquitectura del Frontend se definieron 6 módulos los cuales son:

- **UI:** Donde se agrupan los componentes que integran las vistas de la interfaz. Estos componentes son controlados por el router que es el encargado de manejar los permisos según el rol.
- **Assets**: En este módulo se encuentran las hojas de estilo y los archivos multimedia de la web.
- **Router:** El encargado de definir las rutas de la página y que componente mostrar según la ruta.

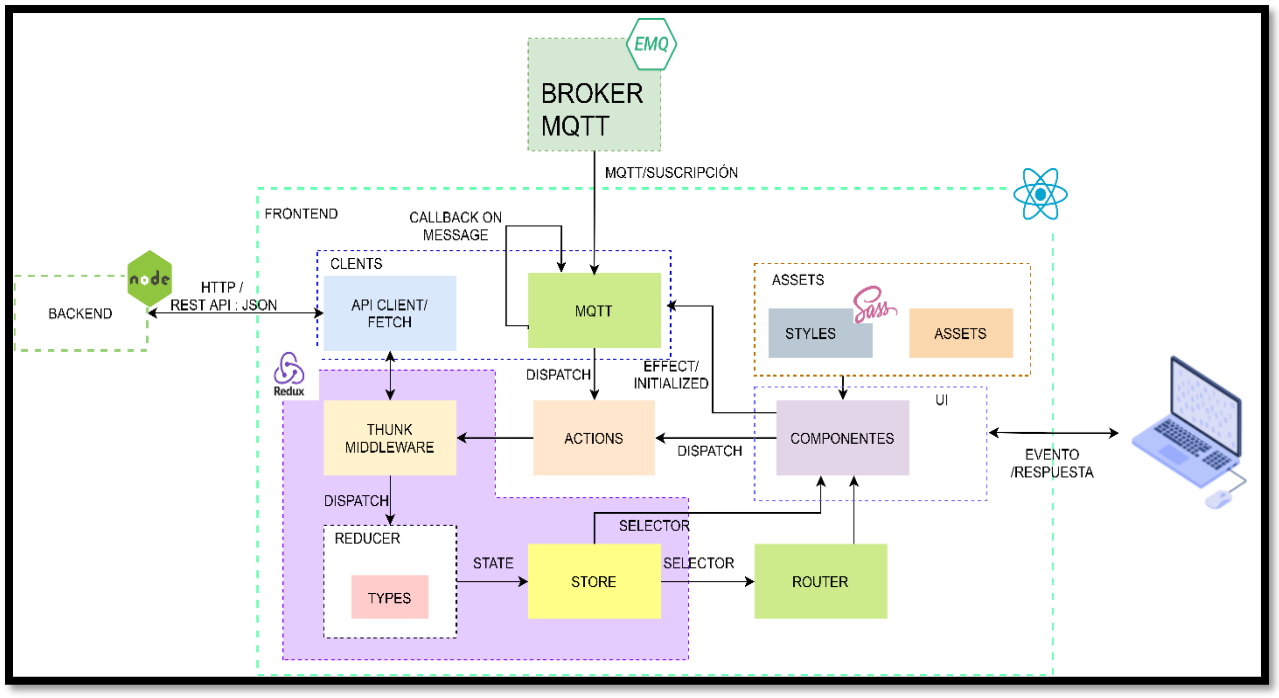
- **Clients:** En esta sección se encuentran el API Fetch con el cual se hacen peticiones http y se tiene el Client MQTT encargado de la conexión al broker y la recepción de los datos enviados al tópico suscrito

- **Actions:** Aquí se ubican los archivos de las peticiones agrupados por la vista. Las acciones se llaman usando un dispatch desde el componente.

- **Patrón Redux:** El patrón redux se conforma por el store, reducer y thunk middleware. El thunk middleware recibe la acción, si esta es asíncrona espera a que la petición tenga respuesta para hacer dispatch al reducer y que este almacene los valores en el store, estos valores son consultados por los componentes por medo del selector

Figura 30.

Diagrama de arquitectura del Frontend



4.3.7. Control de versiones – GITLAB

Es necesario en un proyecto de software manejar el código con una herramienta para el control de versiones, para este proyecto se usó GITLAB ya que permite gestionar las versiones y cuenta con una tabla para el seguimiento de las actividades, algo de suma importancia para establecer el orden y definir que desarrollador se está encargando de cada requerimiento y en qué estado está.

4.3.7.1 Repositorio

Figura 31.

Repositorio del proyecto

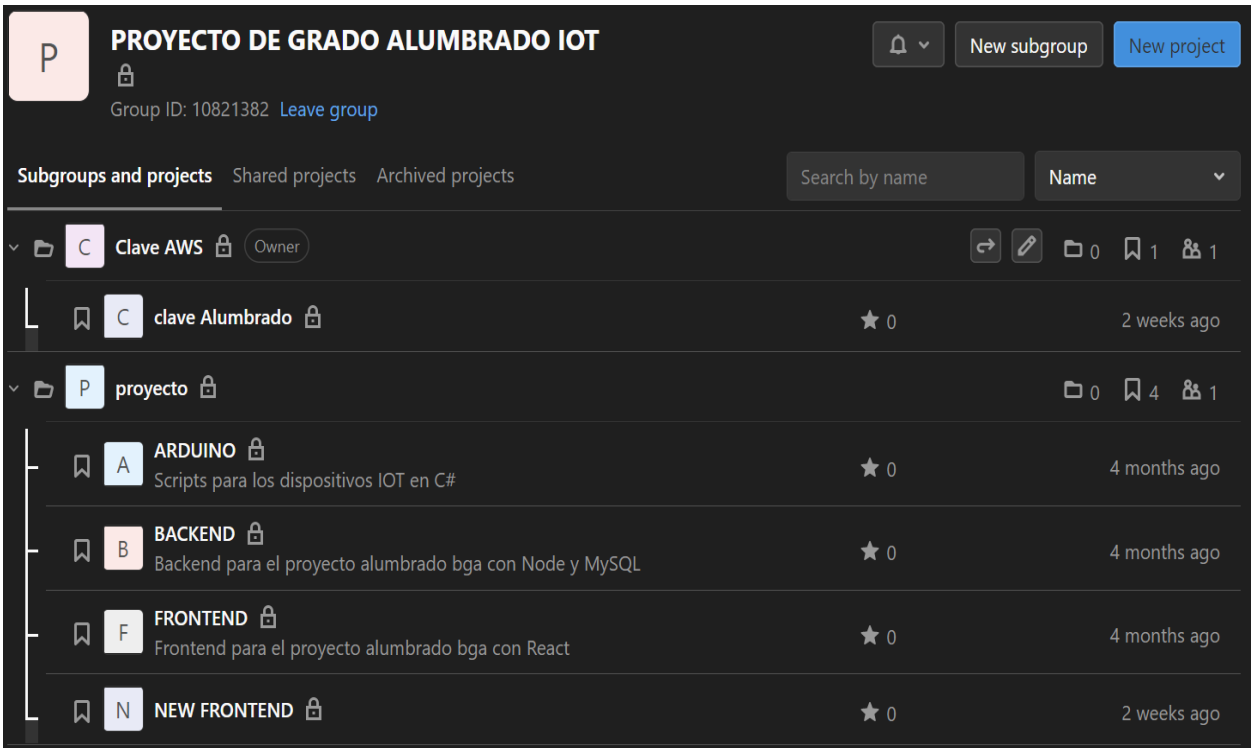


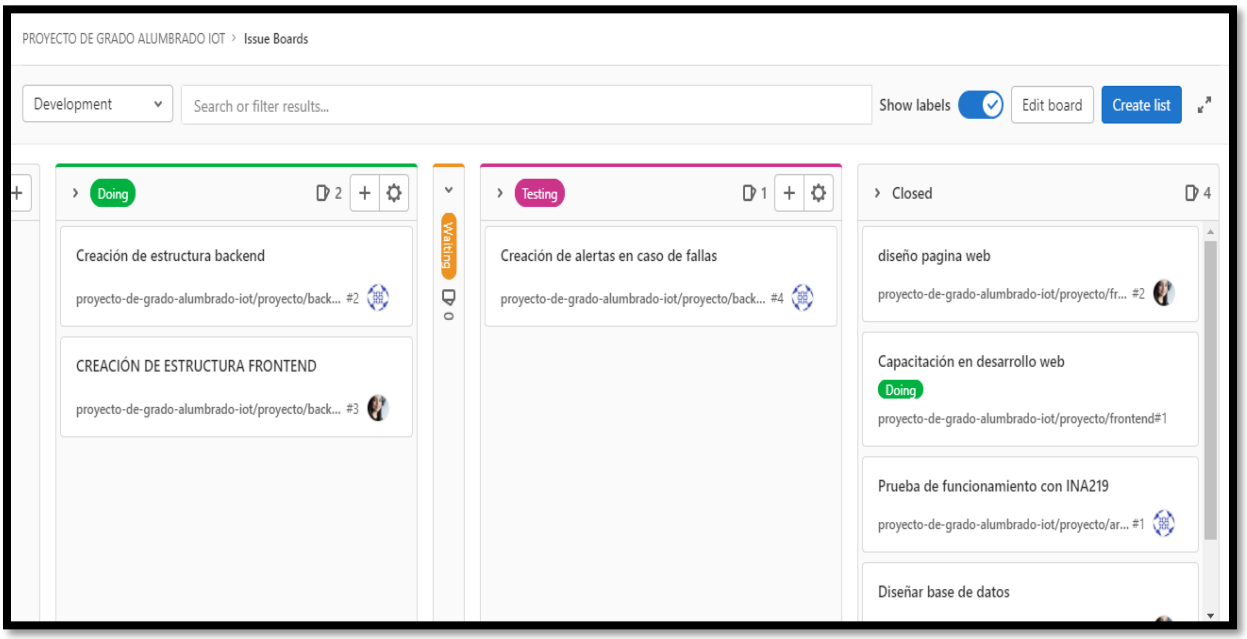
Figura 32.

Grafica de commits

4.3.7.2 Panel de gestión de actividades

Figura 33.

Panel de gestión de actividades



4.4 Desarrollo del Software

4.4.1 Creación del servidor y publicación

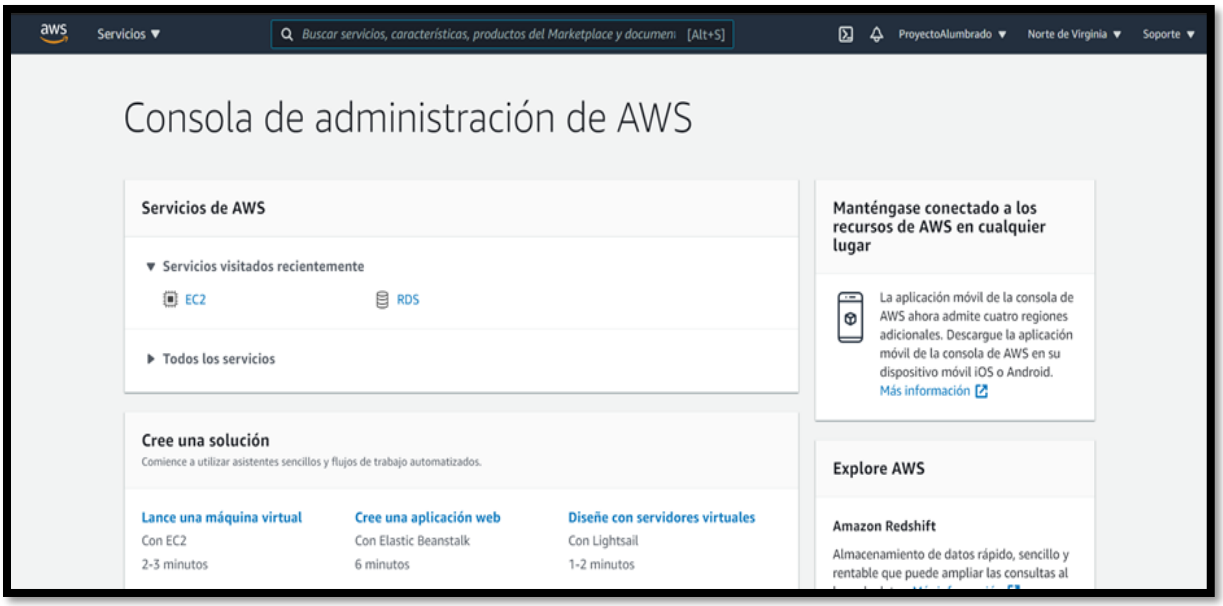
4.4.1.1 Ubuntu. Para adquirir un servidor en AWS por los servicios de EC2 se dio la necesidad de crear una cuenta con Amazon Web Services e iniciar sesión con la consola de ellos.

Para la creación de la cuenta, se requieren datos como un correo electrónico, la contraseña para ingresar a la consola de AWS y un nombre de cuenta junto a datos personales como el nombre, celular, dirección de residencia y por último la información de facturación de una tarjera visa o

master card, esta con el fin de poder realizar cobros por si sobrepasa la capa gratuita. Ya con la cuenta creada y validada se puede ingresar a la consola de AWS.

Figura 34.

Consola de administración de AWS



Desde la consola de AWS podemos observar la interfaz que nos ofrece esta compañía para poder obtener los respectivos servicios, en la caja de búsqueda “Search” escribimos EC2 y seleccionamos este servicio que lleva como descripción Servidores virtuales en la nube.

Una vez dentro de ella nos indica los recursos que tenemos actualmente funcionan con este servicio y también nos muestra las opciones de lo que podemos configurar para que nuestra máquina virtual funcione de la mejor forma para nuestros propósitos.

Figura 35.

Recursos Instancia EC2

Como nos muestra en esta captura de pantalla se observa que actualmente estoy utilizando solo una máquina virtual que es el equivalente a una instancia con una dirección IP fija que es la que me permite no estar configurando la dirección en lo web Hosting ni en el protocolo MQTT, un par de claves que si es creado personalmente, con el que puedo descargar una llave privada y una pública y con ella poder ingresar al servidor desde la terminal de mi computador, además de unos grupos de seguridad uno creado por defecto y otro creado personalmente con el que puedo configurar los puertos de comunicación entre el servidor y la aplicación, y el volumen de disco que oír defecto son 16 GB a 100 IOPS.

Figura 36.

Dashboard EC2

En la dashboard de AWS presenciamos que una vez aplicamos el clic en la opción de instancias se nos presenta un tablero de las instancias que tenemos actualmente corriendo o en su defecto en ejecución, si deseamos lanzar una nueva instancia presionamos en la opción de “Lanzar instancia”

Para la creación del servidor virtual en el servicio de AWS se nos presentan 7 pasos que me van a permitir general diversas acciones, estos pasos son:

1. Elija AMI.
2. Elegir tipo de instancia
3. Configurar instancia
4. Adición de almacenamiento
5. Agregar etiquetas.
6. Página configure Security Group
7. Análisis.

en el primer paso se nos pide elegir una imagen de Amazon Machine (AMI) la cual es una plantilla que contiene la configuración de software como lo son sistemas operativos, servidor de aplicaciones y aplicaciones necesarias para poder lanzar la instancia

Figura 37.

Imagen de Amazon Machine

Existen diferentes tipos para poder elegir el servidor, ya sean de Microsoft, Ubuntu, macOS, Debian, Deep Learning, Red had, SUSE Linux Enterprise. Para nuestra opción de manejo decidimos utilizar Ubuntu Server 18.04 LTS (HVM), SSD Volume Type a 64 bitsa (x86) ya que este tiene buena compatibilidad con las versiones de los demás programas que vamos a utilizar y con ello no generar conflictos a futuros.

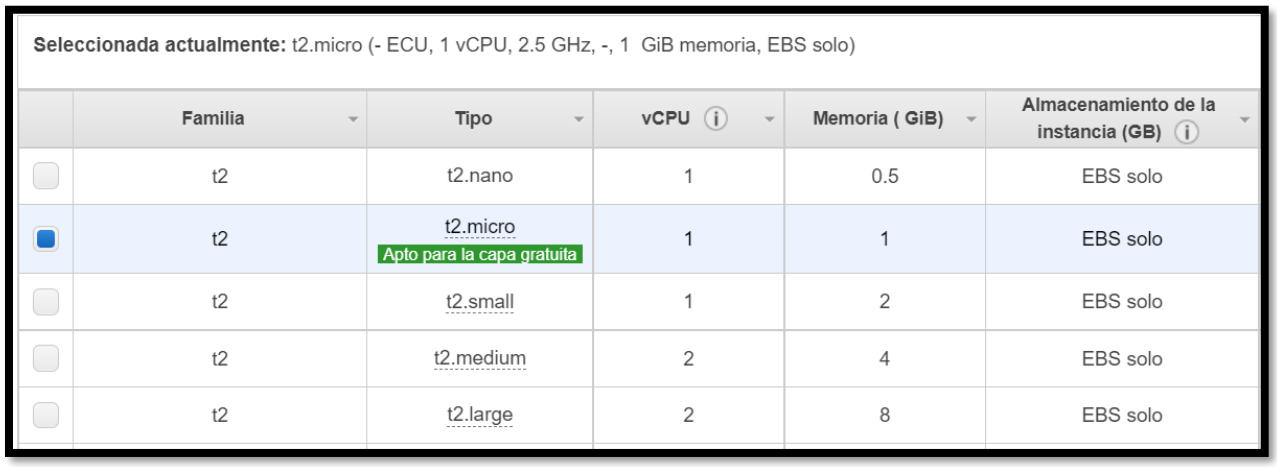
Figura 38.

Sistema Operativo de la instancia

Una vez seleccionamos el sistema operativo en el cual vamos a trabajar el siguiente paso es escoger nuestra máquina virtual, escoges la máquina que se adapte a los vCPU que desea y la memoria que más se ajuste a sus necesidades, para la capa gratuita solo nos permite seleccionar la t2.micro las demás maquina virtuales ya llevan un costo por uso desde el momento que se obtienen.

Figura 39.

Máquina Virtual



	Familia	Tipo	vCPU	Memoria (GiB)	Almacenamiento de la instancia (GB)
☐	t2	t2.nano	1	0.5	EBS solo
☑	t2	t2.micro Apto para la capa gratuita	1	1	EBS solo
☐	t2	t2.small	1	2	EBS solo
☐	t2	t2.medium	2	4	EBS solo
☐	t2	t2.large	2	8	EBS solo

Para la siguiente parte se da la configuración de los detalles de la instancia, en ella uno puede incluir más de una instancia con unas VPC específica y subredes por si se tienen creadas, también se pueden incluir roles de IAM para un manejo más prolijo y código que desee ejecutar una vez la instancia esté en ejecución, en nuestro caso, todo lo vamos a dejar por defecto, tal cual como lo personaliza la página.

Continuando con el cuarto paso, procedemos a configurar el almacenamiento que se puede asociar a los volúmenes de EBS.

En un principio para la capa gratuita AWS nos permite obtener hasta 30 GB de almacenamiento de uso general (SSD) o almacenamiento magnético en EBS.

En este paso también vamos a dejar todo por defecto lo único es que vamos a colocar un tamaño de 16 GB

Figura 40.

Volumen de la máquina virtual

Para el quinto paso se van a agregar las etiquetas, la cual consta de un par de clave-valor con sensibilidad de caracteres y de minúsculas a mayúsculas, esta se puede aplicar a todas las instancia y volúmenes con un máximo de 50 etiquetas por instancia. Nosotros vamos a dejar

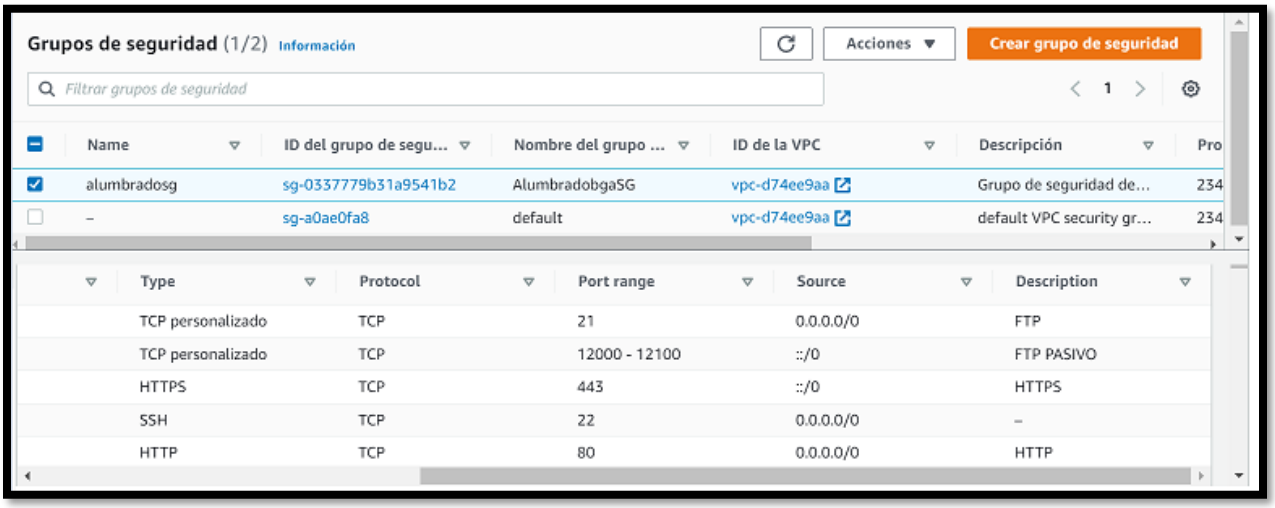
nuestra instancia sin etiquetas para evitar futuros inconvenientes con las claves a incluir, por ello lo dejamos por defecto.

En el sexto paso se dan las configuraciones del Security Group la cual es un conjunto de reglas del firewall con el fin de controlar el tráfico de las instancias. En esta opción lo que se da es la configuración de los puertos que me permiten el acceso sin restricción a los puertos de http y https. Por ejemplo, en nuestro proyecto nosotros manipulamos el puerto 8080 para podernos comunicar al panel de control de nuestro web hosting (Vesta).

Se pueden crear nuevos grupos de seguridad o crear uno existente, para nuestro caso seleccionamos el de AlumbradobgsSG el cual se ha configurado para poder permitir la comunicación con los respectivos dispositivos y hosting.

Figura 41.

Grupos de seguridad



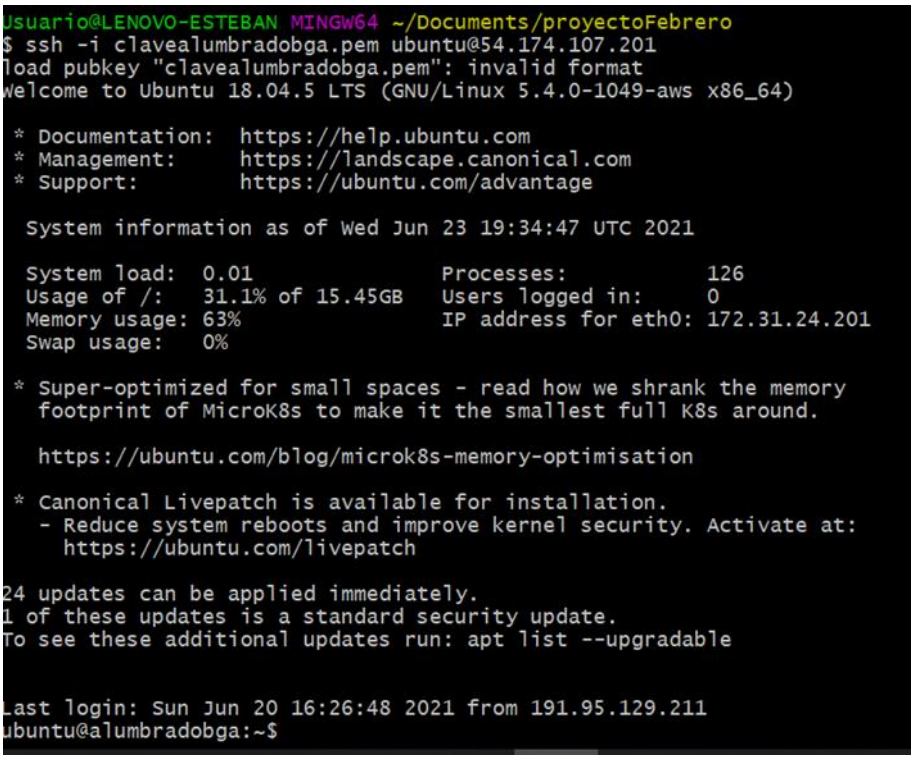
Y el último paso lo que nos muestra es un resumen de todas las opciones y características que hemos decidido para nuestra máquina virtual y con ella poder lanzarla para así tener uso de la

misma, pero antes de lanzarla nos genera un par de claves, unas publica y otra privada la cual nos permite conectarnos desde nuestra terminal al servidor y así poder instalarle programas a la máquina virtual ya que es un servicios de IaaS, nos prestan toda la infraestructura de la nube pero se nos he permitido instalar diferentes plataformas y editarlas. Que para nuestro caso viene siendo nuestro proyecto de alumbrado público manejado desde nodeJS, React.js.

Es importante tener este par de clave bien guardada porque una vez perdida es imposible poder acceder a esta instancia misma, una forma de conectarnos desde una maquina propia con sistema operativo IOS o Ubuntu es desde la terminal con el comando ssh -i ruta-clave-pem ubuntu@ip-address

Figura 42.

Consola de la Instancia



Aunque desde Windows también existe un programa que nos permite conectar de esta misma forma y es con git bash lo único que toca realizar de más es darle los permisos al archivo .pem de chmod 400 ruta-clave-pem, pero el otro camino que existe para poderse conectar desde windows es por medio del programa putty, es un poco más tedioso con base a ingresar porque primero se debe realizar una conversión de la llave en putty key generator y guardarla como llave privada, y una vez convertida esta llave se procede al putty configure vincularla y así poder manipular el servidor.

4.4.1.2 Vesta. Una vez creado nuestro servidor virtual en AWS es momento de crear nuestro panel web hosting, este nos permite administrar toda la infraestructura de alojamiento por medio de una dashboard haciéndolo más practico evitando usar comandos en la terminal, estos paneles tienen características básicas como:

- Gestión de archivos y backups.
- Firewall
- Sistema de gestión de correos.
- Administrar base de datos.
- Gestión de DNS
- SHH
- Monitoreo de ancho de banda
- Entre otros

Existen múltiples paneles de control de hosting gratuitos y de código abierto como ISPConfig, CentOs, Virtualmin, Ajenti, Sentora, Kloxo-MR, Vestacp, Froxlor, TinyCP, GNUPanel. El panel que más se adaptó a las características de nuestro proyecto además de las

múltiples opciones de configuración y administración de servicios que presenta, decidimos utilizar vestacp.

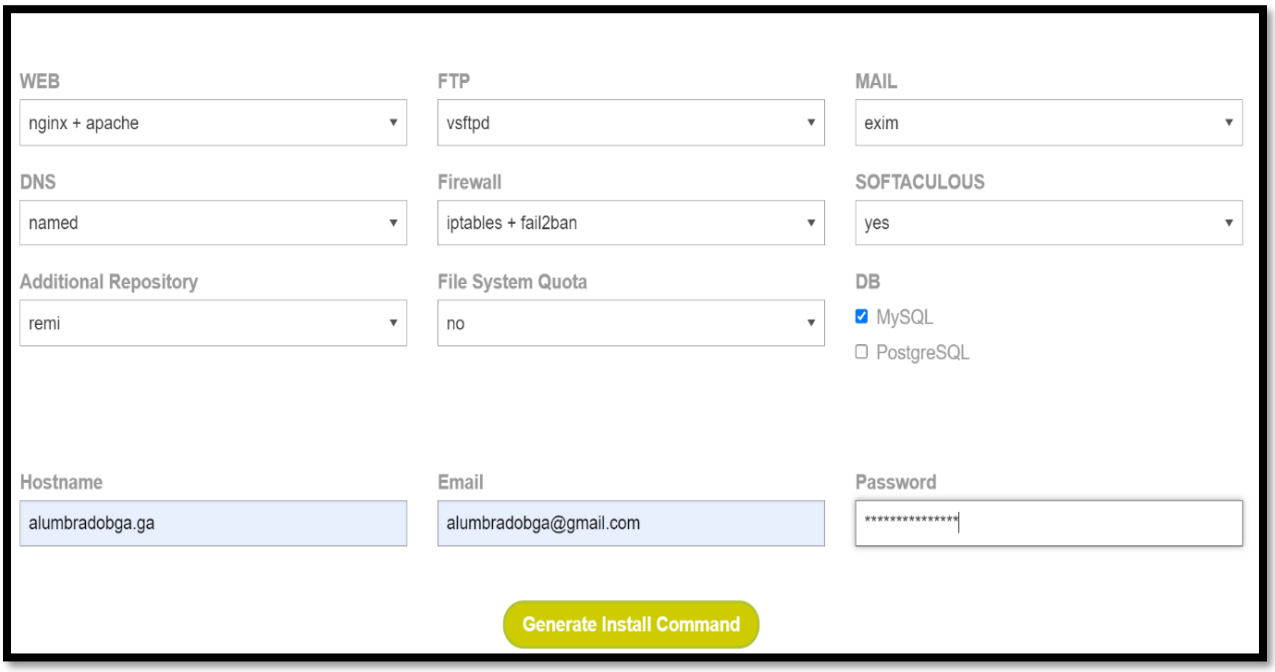
Una de sus principales características es:

- Está traducido a 26 idiomas.
- Contiene dos firewall
- Maneja DNS
- Presenta servidores Web como Nginx, Apache
- Está programado en PHP
- Cuenta con anti-spam
- Maneja un antivirus con el nombre de ClamAv
- Gestiona un servidor de correos
- Tiene base de datos mysql, phpMyAdmin, PostgreSQL
- Maneja un protocolo de transferencia de archivos
- Funciona en CentOS, Debian, Ubuntu
- Funciona con unos mínimos requisitos de 1 CPU, 512 Mb RAM y 20 Gb HDD

Para descargar este panel web hosting se instala desde la página vestacp.com.

Figura 40.

Configuración panel Vesta



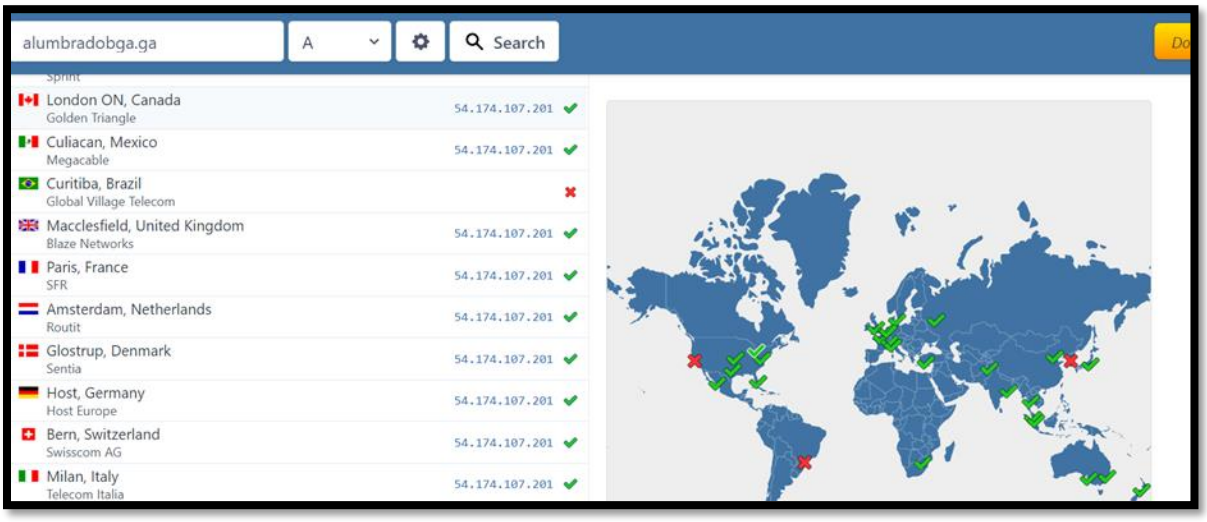
Procedemos a configurar nuestro comando para poderlo descargar con las características de la imagen anterior con la cual se debe chequear antes el Hostname, el Dominio lo obtuvimos desde el link freenom.com, registrándonos en la página con nuestro correo Gmail procedemos a crear un nuevo dominio desde la opción de servicios, escribimos el dominio que deseamos crear y continuamos con la creación del mismo.

Para el nombre del dominio decidimos escoger alumbradobga.ga ya que se encontraba disponible y concuerda con lo que deseamos para el proyecto.

Luego enlazamos el dominio con la IP que generamos en la instancia para que estén vinculadas, y antes de continuar con la instalación de vestacp en el link whatsmydns.net esperamos que nuestro DNS se encuentre en el 95% de los servidores del mundo.

Figura 43.

Mapa de vinculación de nuestro dominio web



Nota. Tomado de Whatsmydns.net

Una vez tengamos el dominio en la mayoría de los servidores del mundo continuamos con la instalación de Vestacp generando el comando de instalación.

Figura 44.

Código para Instalar Vesta

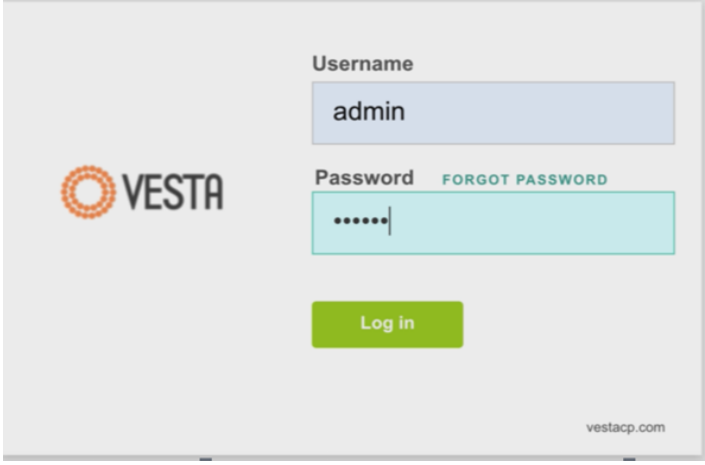
```
1  # Connect to your server as root via SSH
   ssh root@your.server

2  # Download installation script
   curl -O http://vestacp.com/pub/vst-install.sh

3  # Run it
   bash vst-install.sh --nginx yes --apache yes --phpfpm no --named yes --remi yes --vsftpd yes --proftpd no --iptables
   yes --fail2ban yes --quota no --exim yes --dovecot no --spamassassin no --clamav no --softaculous yes --mysql yes -
   -postgresql no --hostname alumbradobga.ga --email alumbradobga@gmail.com --password **********
```

Para ello una vez adentro de la instancia entramos en la opción de super usuario con sudo su y copiamos el segundo comando de la imagen anterior: curl -O http://vestacp.com/pub/vst-install.sh y por último escribimos el tercer comando para poder descargar este panel en la instancia de AWS.

Una vez instalado el programa en la instancia, para verificar que quedó correctamente instalado en nuestro navegador ingresamos nuestro nombre de dominio con el puerto al que apunta, del cual este se configuro en los grupos de seguridad de AWS (alumbradobga.ga:8083) podemos observar una interfaz así:

Figura 45.

Inicio de sesión en Vesta



Donde su usuario será admin y la contraseña es la que ingreso a la hora de generar el comando, una vez adentro ya se puede observar la dashboard de nuestro panel web hosting VestaCP

4.4.1.3 FTP. Llamado protocolo de transferencia de ficheros por sus siglas en ingles File Transfer Protocol, Es un protocolo de red para trasladar archivos entre sistemas conectados a una red manejado con el protocolo de control de transmisión el cual permite la conexión entre dos hosts e compartan datos esto independientemente del sistema operativo que se usen en las maquinas anfitrionas, en otras palabras es básicamente un servidor en el cual podemos subir y bajar ficheros además de compartir, actualizar los ficheros en nuestra página web.

Como bien dicho anteriormente los usos que se le dan al FTP son.

- Ya que las páginas web están conformadas en ficheros, sean estos archivos multimedia o archivos en código html, php, javascript y otra parte se encuentra en base de datos este protocolo nos permite subir estos ficheros y editarlas por medio se clientes FTP o por un gestor de ficheros como el panel de control plesk.
- Para tener tus ficheros en la nube y gestionarlos.
- Para compartir ficheros.
- Para hacer copias de seguridad.

Además, se puede evidenciar ciertas ventajas del uso de FTP

- Sube páginas web de una manera sencilla y rápida.
- Existen muchos clientes de FTP distintos
- Permite cambiar, crear ficheros y directorios, cambiar permisos, renombrar ficheros, borrar ficheros.
- Permite subir en segundo plano y ficheros comprimidos también.

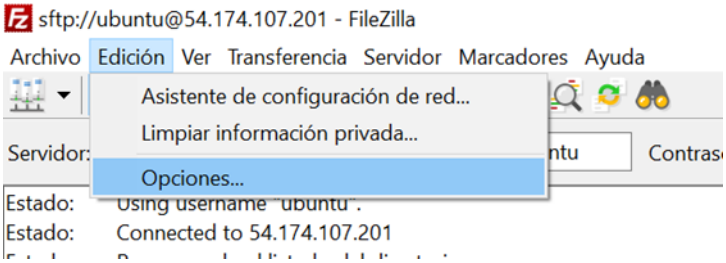
Para el buen uso de este protocolo se debe configurar los puertos en el servidor de AWS, en la instancia, y también en el panel web hosting de Vesta, el puerto que se utiliza por defecto

para este protocolo es el puerto 21, una de las aplicaciones que me permite manejar este protocolo por medio de una interfaz es FileZilla.

Para ello en la barra de herramientas del programa nos dirigimos a edición y le damos click en opciones.

Figura 46.

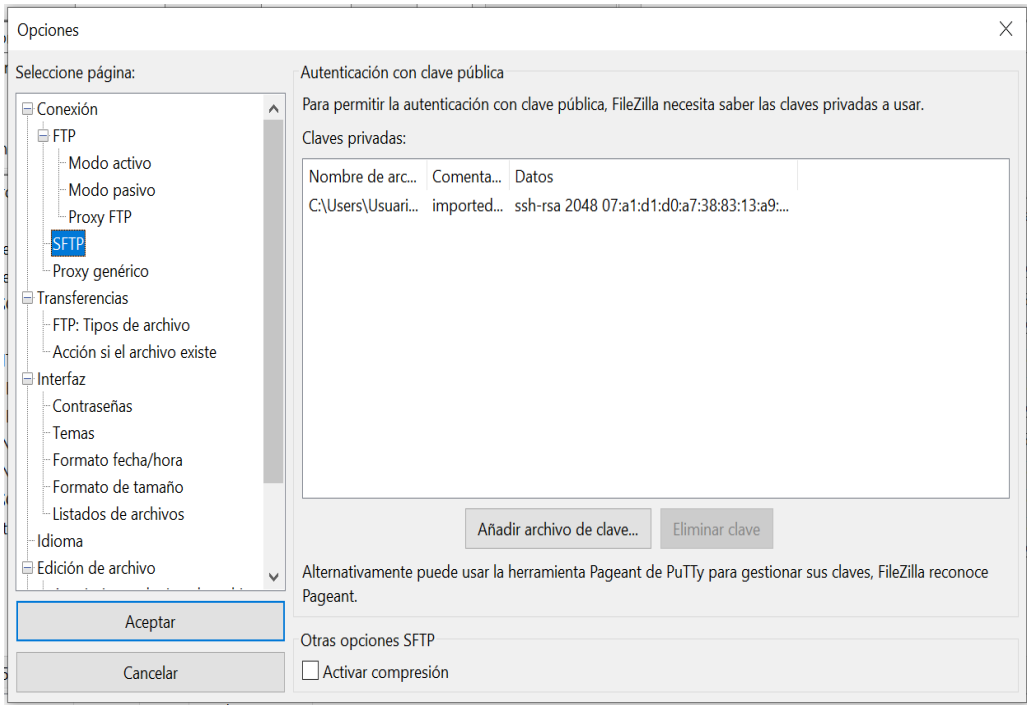
Barra de Herramientas de FileZilla



Nos habilita una ventana la cual debemos incluir nuestra llave pública y privada que la instancia de AWS nos creó una vez lanzada la máquina virtual.

Figura 47.

Opciones de FileZilla



Una vez incluida la llave, agregamos la información del servidor, su dirección ip y el nombre de usuario que para nuestro caso es 54.174.107.201 y Ubuntu, el puerto no es necesario incluirlo porque por defecto se conecta al puerto 21 y la contraseña tampoco es necesario incluirla porque en la llave agregada anteriormente se encuentra registrada. y para tener la conexión le presionamos “Conexión rápida”

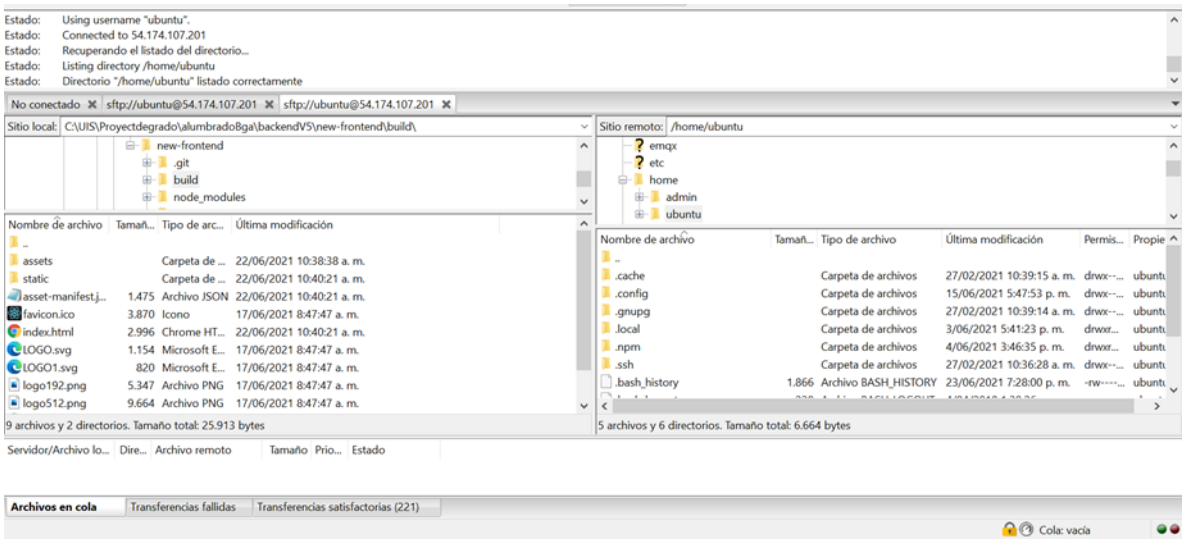
Figura 48.

Conexión a la instancia

Mostrándonos la siguiente interfaz lista para su uso y poder realizar las respectivas transferencias de nuestro computador al servidor, su forma de transferir archivos simplemente es trasladando el archivo de la ventana izquierda que es el explorador de archivos de nuestra maquina hacía la ventana derecha que es el explorador de archivos de nuestro servidor alojado en la nube.

Figura 49.

Interfaz de FileZilla



4.4.1.4 Nginx. Por temas en la seguridad de la aplicación web mediante cortafuegos, se procede a usar este software de código abierto sirviendo como un proxy inverso, también funciona como proxy de correos electrónicos y como balanceador de carga HTTP.

Esta configuración del servidor Nginx por medio de un Proxy inverso se realiza con el propósito de poder proporcionar el mismo contenido a todos los usuarios para las URL asociadas bajo una misma dirección IP del servidor.

Para la instalación de los paquetes de Nginx se requieren los siguientes comandos dentro de la instancia de AWS bajo el perfil de super usuario o escribiendo la palabra sudo. Para actualizar la lista de paquetes apt-get utilizamos el siguiente comando.

Figura 50.

Actualizar Paquetes

```
root@alumbradobga:/home/ubuntu# sudo apt-get update
```

Luego realizamos la instalación de Nginx

Figura 51.

Instalación Nginx

```
root@alumbradobga:/home/ubuntu# sudo apt-get install nginx
```

Y luego le ordenamos a nuestro servidor Ubuntu que a todo lo que se procese en la tarjeta de red en el puerto 80 se redirija al puerto 3000 con el siguiente comando

Figura 52.

Apuntar al puerto 3000 por defecto

```
root@alumbradobga:/home/ubuntu# sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 3000
```

Y reiniciamos el servidor con el comando <<sudo service nginx restart>>

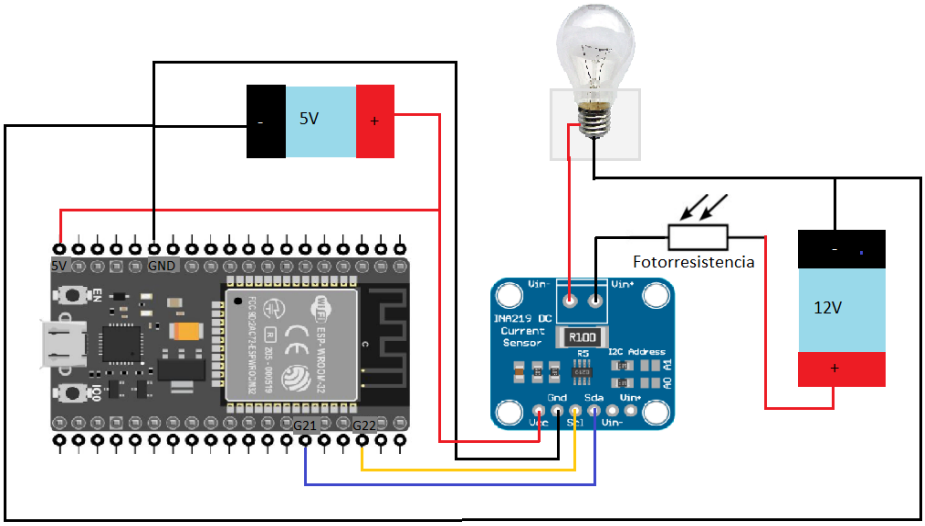
Si en un futuro se desea matar el proceso que está corriendo por el puerto 3000 en el servidor de nginx se necesita conocer el número del proceso en el que está corriendo y esto se conoce con el comando <<sudo netstat -lpn | grep :3000>>, con esto conocemos el número del proceso para poder matarlo con el comando <<kill num_process>>

4.4.2 Construcción del hardware

En el desarrollo del hardware se tomaron en cuenta varios dispositivos electrónicos como el módulo INA219, el módulo ESP32, fotorresistencias, luminarias exploradoras a 12V, y una fuente de 12V con una más de 5V.

Figura 53.

Circuito del Hardware simulado



Para el módulo ESP32 era necesario guardar un programa con el cual, puede enviar los datos que recibe del sensor al Backend de la aplicación para ello se utilizó un programa en Arduino IDE con el que se implementó diversas librerías para su buen funcionamiento y así poderlo cargar en el microcontrolador ESP32, este programa de codificó en el software de Arduino con el lenguaje C++.

4.4.3 Creación de la base de datos y cargue a la nube

Las respectivas conexión de la base de datos se dio a cabo desde un comienzo con Vesta, allí creamos la base de datos con sus respectivas credenciales, como el nombre de la base de datos, el nombre de usuario, su contraseña, se dio el tipo de base de datos, que para nuestro proyecto fue MySQL, con esto, dentro del servidor de base de datos que vesta lleva incluido, nos crea una base de datos que se encuentra alojada en el puerto 3306, para que pueda existir conexión con AWS, en las configuraciones de grupos de seguridad es necesario habilitar este puerto para que pueda existir conexión bidireccional y así esta se cargue en la base de datos de la nube

Figura 54.

Conexión Base de Datos

```
const { Sequelize } = require('sequelize');
const mysql = require('mysql2');
require('dotenv').config();  1.2K (gzipped: 706)

const conn = new Sequelize(
    process.env.dbName,
    process.env.dbUser,
    process.env.dbPassword, {
    dialect: 'mysql',
    host: process.env.dbHost,
    port: process.env.dbPort,
    logging: false, // para no mostrar por consola las sentencias SQL ejecutadas.

    dialectOptions: {

        connectTimeout: 60000

    },
    pool: {
        maxIdleTime: 30,
        idle: 20000,
        evict: 15000,
        acquire: 30000,
        max: 15,
        min: 0,
    }
});


module.exports = conn;
```

4.4.4 Configuración del broker y conexión con el hardware

Es fundamental para una buena conexión configurar en el firewall de vesta los puertos 1883, 8883, 8093, 8094, 8090, 18083 esto se hace tanto en vesta como en AWS, todo esto se realiza con el protocolo TCP.

Una vez descargada EMQ X Broker, realizamos las respectivas configuraciones con la versión 3.0.1, EMQ toma por defecto el puerto 8083 como MQTT/WebSocket Protocol pero ese puerto ya se encuentra ocupado por vesta por el cual es necesario realizar el cambio al puerto 8093, para ello nos dirigimos al archivo emqx.conf que se encuentra en la carpeta de emqx y reescribimos

el listener.ws.external con el puerto 8093 y también el management.listener.http con este mismo puerto que se encuentra en el documento de emqx_management.conf dentro de etc/plugins, una vez realizado esto para ejecutar el bróker es necesario insertar el comando "./bin/emqx start" en la carpeta principal de esta misma.

Ahora para poder permitir la conexión con el hardware se creó un middleware que me generara este tipo de actividad con el cual me lo conecta, me genera las respectivas subscripciones y permite en envió de mensajes.

Figura 55.

Conexión al broker

```
const mqtt = require("mqtt");  154.6K (gzipped: 44.7K)
const { process_msg } = require("../controllers/esp32_controllers");

const clientId = "alumbradoMar_" + Math.random().toString(16).substr(2, 8);
const host = "ws://alumbradobga.ga:8093/mqtt";
//No funciona con 8094 porque esta corriendo localmente
const options = {
    keepalive: 60,
    clientId: clientId,
    username: "web_client",
    password: "private",
    protocolId: "MQTT",
    protocolVersion: 4,
    clean: true,
    reconnectPeriod: 1000,
    connectTimeout: 3 * 1000,
    will: {
        topic: "WillMsg",
        payload: "Connection Closed abnormally..!",
        qos: 0,
        retain: false,
    },
};

console.log("Connecting mqtt client");
const client = mqtt.connect(host, options);
Complexity is 4 Everything is cool!
client.on("connect", () => {
    console.log("Mqtt connect with WS! Congratulations!!");
    Complexity is 3 Everything is cool!
    client.subscribe("values", { qos: 0 }, (error) => {
        if (!error) {
            console.log("subscription Successful");
        } else {
            console.log("subscription failed");
        }
    });
});

client.on("message", (topic, message) => {
    //console.log("Mensaje recivido bajo topico: ", topic, message.toString());
    process_msg(topic, message);
});

client.on("error", (err) => {
    console.log("Connection error: ", err);
    client.end();
});

client.on("reconnect", () => {
    console.log("Reconnecting...");
});
```

4.4.5 Desarrollo de backend y endpoints

En el Backend se realizó toda la lógica del programa para que pueda funcionar de las formas más óptima y segura posible

4.4.5.1 Controladores. Es esta carpeta controllers, ubicamos todos los archivos que responden directamente a cada solicitud que se le hace a la aplicación por medio de HTTP, para ello se generaron diferentes archivos que me permitían navegar con rapidez hacia las peticiones que el usuario me exige, como por ejemplo todo lo que tiene que ver las autorizaciones para navegar a endpoints diferentes, así como el manejo de la dashboard y sobre todo con el EP32.

4.4.5.1.1 Tipos de fallas en el alumbrado público. Existen multitudes de fallas en el alumbrado público que pueden afectar el transito seguro por parte de los ciudadanos, ya sea por inoperatividad de la farola, que esta no esté dando la potencia adecuada, que la luminaria se encuentre rota, que el difusor de la luminaria se encuentre inoperable, se encuentre funcionando cuando aún está la luz del sol, la existencia de objetos que interfieran con una amplia difusión de luz como lo son los árboles, o que la lampara prenda y apague en todo momento. En el presente proyecto nos vamos a enfocar en cuatro tipos de errores: error por bajo y alto consumos de energía eléctrica y que se encuentre encendido de día o se encuentre apagado de noche.

Para un funcionamiento óptimo de nuestra luminaria funcionando con una fuente de 12 voltios de concretó un valor de voltaje de 8.28 V, potencia de 0.67 W y una corriente de 81.40 mA.

Tabla 5.

Tablas de fallas en el alumbrado público.

Tipos de salida	Descripción
Normal	Cuando la luminaria está funcionando a un horario de 6PM a 6AM y presenta los valores mencionados anteriormente, genera este tipo de salida.
High-Use	Esta salida se da cuando su funcionalidad se encuentra en el horario de 6PM a 6AM, pero presenta un consumo de corriente superior a 300 mA
Low-Use	Esta salida se da cuando su funcionalidad se encuentra en el horario de 6PM a 6AM, pero presenta un consumo de corriente inferior a 250 mA
Day-On	Esta salida se da cuando su funcionalidad se encuentra en el horario de 6AM a 6PM, pero presenta un consumo de corriente superior a 1 mA
Night-Off	Esta salida se da cuando su funcionalidad se encuentra en el horario de 6PM a 6AM, pero presenta un consumo de corriente inferior a 1 mA

4.4.5.2 Middlewares. En esta parte se incluyen todos los archivos que ofrecen servicios y funciones comunes para las aplicaciones, en él independientemente del software, se pueden tomar acción en cualquier código, permitiendo desarrollar con mayor eficiencia actuando de una forma uniforme.

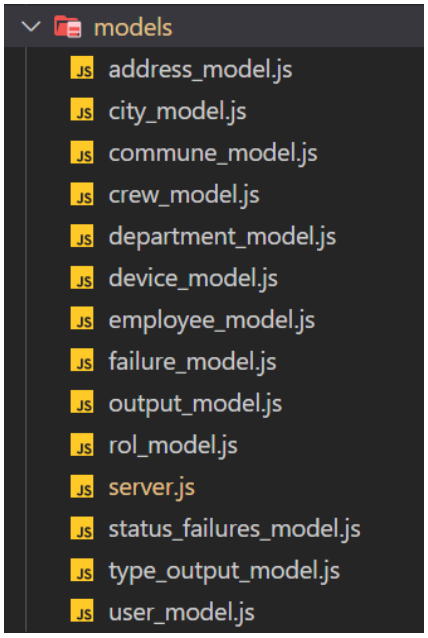
Los archivos de EMQX y también el de validación por JWT (Json Web Token) son middlewares que nosotros incorporamos para la trazabilidad de datos y confirmación de estos mismo con el fin de mejorar la distribución del código y con ello el entendimiento de él.

4.4.5.3 Modelos. Ya que estamos usando la librería de sequelize, la sección de modelos es la esencia para esta librería, siendo estos una representación abstracta de lo que son las tablas en la base de datos, en él también podemos realizar las respectivas relaciones entre tablas y poder

general e incorporar las claves primarias y foráneas que me permite tener la comunicación entre ellas. Además, también incluimos un modelo de servidor manejando esto por medio de clases, siendo pensado esto para una escalabilidad futura con el cual se pueda llegar a obtener diferentes instancias a la clase servidor y funcionar de una forma paralela.

Figura 56.

Distribución de los modelos



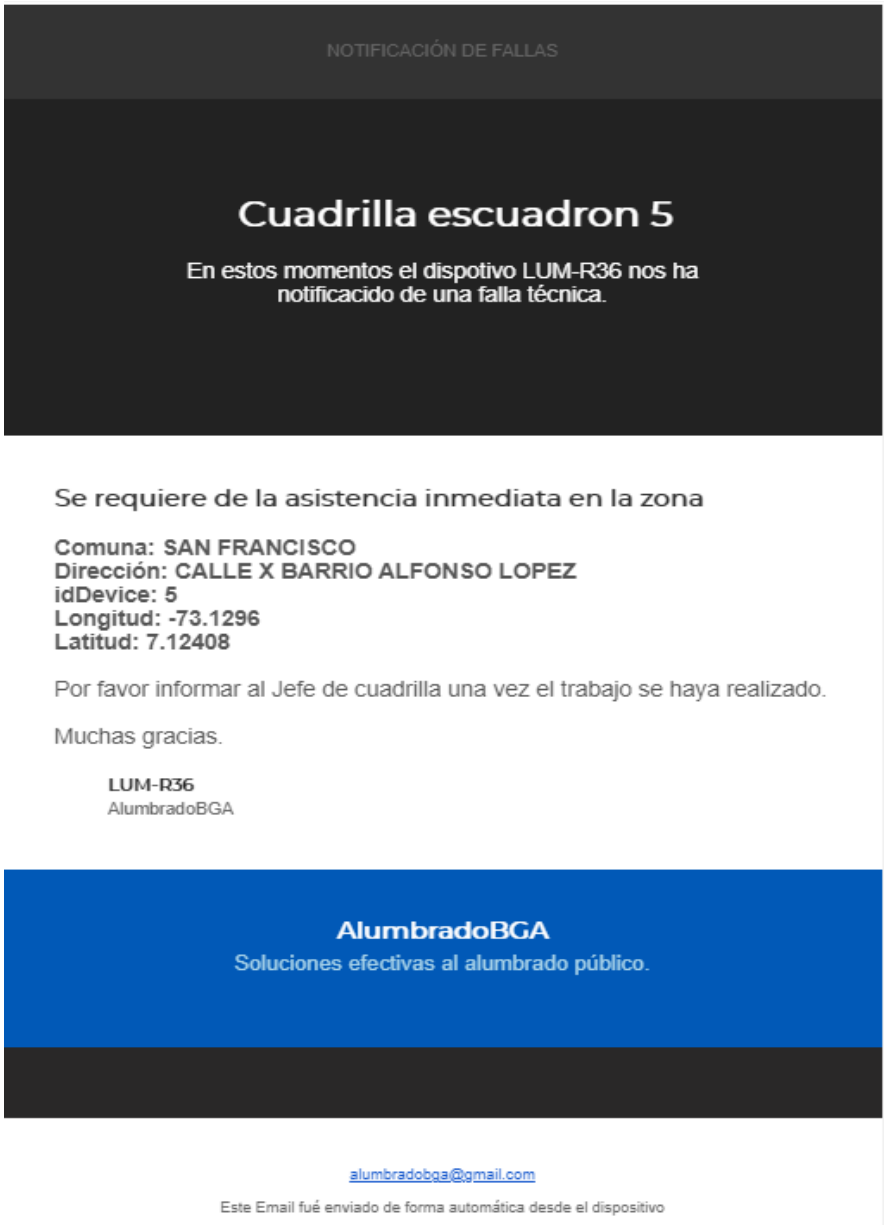
4.4.5.4 Templates. En la parte de templates se incorporan todos aquellos archivos que me generan un código html, para nuestro programa se utilizó en la generación de notificaciones al correo electrónico, donde se le informaba al usuario la información del dispositivo que presentaba la falla.

La librería que se utilizó para realizar él envió de estas notificaciones al correo electrónico fue Nodemailer, Para poder utilizar este servicio se deben tener registrados los correos en la

función de AWS (Amazon Simple Email Service) ya que la licencia gratuita de esta librería no me permite generar envíos a cualquier email.

Figura 57.

Mensaje de notificaciones



4.4.5.5 Rutas. Para la creación de las rutas usamos internamente dentro de las rutas maestras el concepto de router de la librería de express el cual me permite crear miniaplicaciones generando un manejo de recursos óptimos, el cual realizan consultas a la base de datos, mientras que a las rutas maestras donde se puede presenciar un cambio en la interfaz del usuario se utilizó la función de app para que su navegabilidad fuese más amena y rápida para el usuario

4.4.5.6 Helpers. Esta carpeta se creó con el fin de poder ayudar a incluir todos los archivos que de algún modo quedaban no se incluían en las demás secciones, es eta carpeta incluimos todas las consultas necesarias a la base de datos que se tuvieron que realizar para un buen desarrollo en el Frontend, estos datos se compartieron en formato JSON.

Estas consultas se realizaron desde la librería de sequelize ORM la cual está basa en promesas y modelos.

4.4.6 Desarrollo del Frontend

Para el desarrollo del frontend se trataron unos puntos claves con el fin de proporcionar un funcionamiento adecuado y seguro.

4.4.6.1. Autenticación y rutas. La autenticación se realizó almacenando el token en el local storage y creando 2 routers, uno privado y uno público, para acceder al público no se debe estar autenticado pues si lo esta no será redirigida al router privado y viceversa. Una vez autenticado se habilitan los componentes y rutas según el rol, de manera que solo los usuarios que

tienen permitido realizar ciertas acciones o ver información clasificada puedan hacerlo y brindar la seguridad que los datos no serán expuestos.

Cada vez que se recarga la aplicación se ejecuta un proceso de validación de token, de esta manera, si el token se venció el usuario será redirigido a la página del login y sus datos almacenados tanto en local storage como en el redux store serán eliminados.

4.4.6.2. Componentes UI. Para la creación de la interfaz de usuario, se hizo uso de SASS el cual es un metalenguaje de Hojas de Estilo en Cascada (CSS). Es un lenguaje de script que es traducido a CSS, facilita el uso de variables para almacenar los atributos de los temas y propone una estructura cómoda y fácil de entender para organizar los estilos de manera modular.

Además, se empleó la librería Material UI con el fin de reutilizar componentes y simplificar el código, creando una interfaz intuitiva y agradable.

Figura 58.

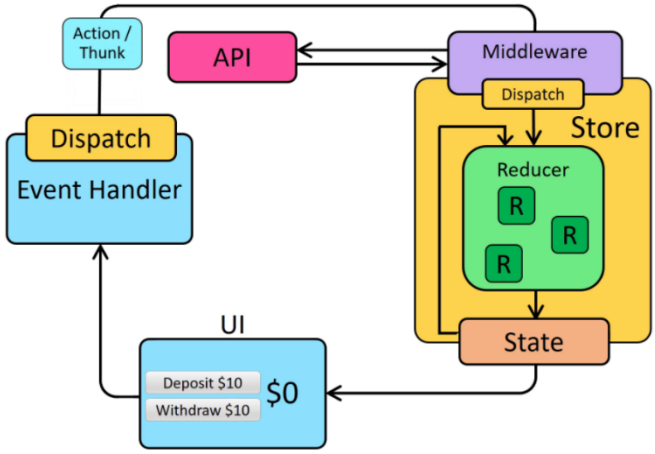
Estructura de estilos

4.4.6.3. Redux. Se usó la librería Redux la cual pretende que el manejo del estado a lo largo de la aplicación sea predecible. Para ello impone restricciones a la hora de actualizar el estado. Estas son:

- El estado está almacenado en una única fuente de verdad llamada store
- El contenedor que almacena el estado no tiene métodos para modificar el estado directamente, por lo que está protegido contra escritura, solo se pueden modificar a través de acciones
- para especificar cómo ha de cambiar el estado de la aplicación, se utilizan funciones puras llamadas Reducers.

Figura 59.

Arquitectura Redux



Nota: Tomado de Redux Fundamentals, Part 6: Async Logic and Data Fetching. https://redux.js.org/tutorials/fundamentals/part-6-async-logic

Redux funciona con los elementos principales que se mencionan a continuación

4.4.6.3.1 Acciones. Las acciones son portadores de la información que indica cómo se quiere modificar el estado. Se envían desde la aplicación hasta el Store.

4.4.6.3.2 Reducers. Un reducer es una función que especifica cómo se cambia el estado en función de las acciones recibidas. Es una función pura que devuelve un nuevo estado de una manera definida para un estado y una acción. Obtiene un nombre único porque comparte la misma estructura que ellos.

4.4.6.3.3 Store. El store es un objeto que almacena el estado de la aplicación. Emplea el patrón de diseño observer para comunicarse con el componente que renderiza el valor requerido en la interfaz de forma reactiva.

4.4.6.3.4 Thunk Middleware. Redux Thunk es un middleware que le permite invocar creadores de acciones que devuelven una función en vez de un objeto de acción. Esa función recibe el método de envío del store, que luego se utiliza para enviar acciones síncronas regulares dentro del cuerpo de la función una vez que se completaron las operaciones asíncronas. (Alligator.io, 2020)

4.4.6.3.5 Selectors. Permiten devolver partes del objeto guardado en el store de la aplicación dado un estado.

4.4.6.4. Leaflet. Para generar el mapa donde se ubican los dispositivos y las comunas se usó una librería open source de Javascript llamada Leaflet que permite la generación de mapas interactivos que utiliza HTML, CSS y JS

4.4.6.5. MQTT y datos en tiempo real. Para la conexión al Broker MQTT se implementó la librería MQTT la cual permite conectarse al broker y suscribirse a los tópicos deseados. A continuación, se muestra la configuración de conexión al broker.

Figura 60.

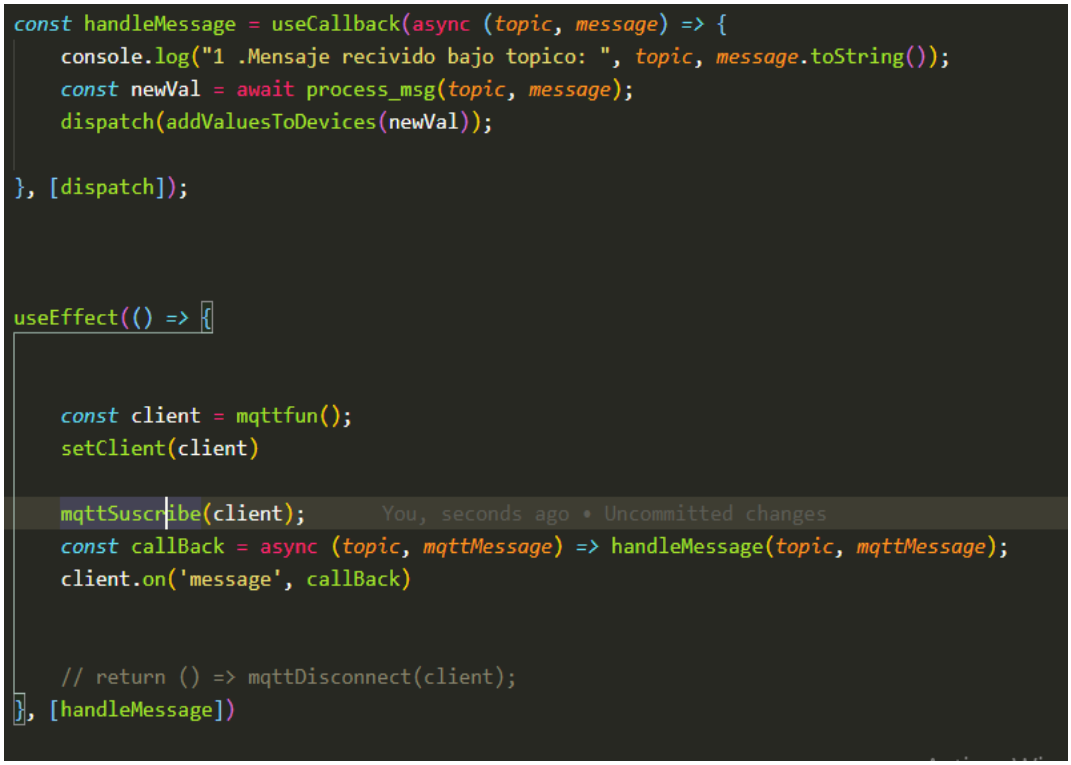
Configuración de conexión al Broker MQTT

```
const clientId = "alumbradoMar_" + Math.random().toString(16).substr(2, 8);
const host = "ws://alumbradobga.ga:8093/mqtt";
const options = {       You, a month ago • se añadio el broker para obtener valo
    keepalive: 60,
    clientId: clientId,
    username: "web_client",
    password: "private",
    protocolId: "MQTT",
    protocolVersion: 4,
    clean: true,
    reconnectPeriod: 1000,
    connectTimeout: 3 * 1000,
    will: {
        topic: "WillMsg",
        payload: "Connection Closed abnormally..!",
        qos: 0,
        retain: false,
    },
};
```

Esta conexión se realiza al momento de renderizar el elemento que encargado de mostrar los valores. Para lograr el renderzado constantemente se empleó un useEffect para la conexón incicial y un useCallback, que se llama cada vez que se resve un mensaje nuevo al tópico que se está suscrito. A su vez el callback hace dispatch a una acción que guarda los valores en el store, el componente utiliza un selector para obtener estos valores y renderizarse con el nuevo valor.

Figura 61.

useEffect y useCallback



```
const handleMessage = useCallback(async (topic, message) => {
    console.log("1 .Mensaje recivido bajo topico: ", topic, message.toString());
    const newVal = await process_msg(topic, message);
    dispatch(addValuesToDevices(newVal));

}, [dispatch]);



useEffect(() => {


    const client = mqttfun();
    setClient(client)

    mqttSuscribe(client);       You, seconds ago • Uncommitted changes
    const callBack = async (topic, mqttMessage) => handleMessage(topic, mqttMessage);
    client.on('message', callBack)


    // return () => mqttDisconnect(client);
}, [handleMessage])
```

4.4.6.6. Dashboard e informes. En la creación del dashboard se usó la librería ApexCharts la cual permite una alta personalización en las gráficas y flexibilidad para mostrar los datos. Se plantearon diferentes tipos de gráficas y un informe generado en Excel con un consolidado de los datos de registros y fallos de cada dispositivo dado un rango de tiempo, esto con el fin de que los administradores puedan generar sus propios informes y sacar estadísticas. A continuación, se muestran las gráficas que trae el dashboard del sistema.

Figura 62.

Grafica del consumo total

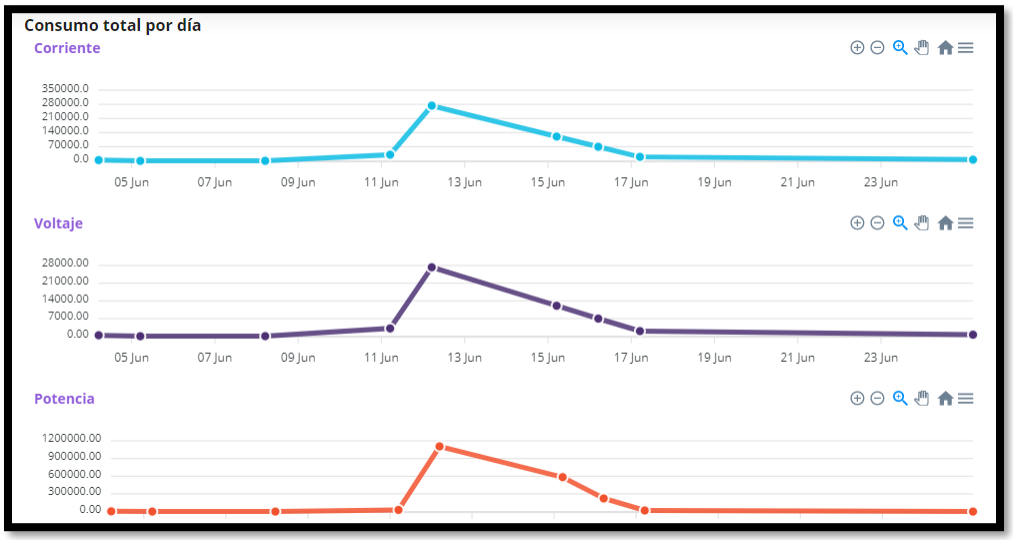


Figura 63.

Gráficas de fallos por sectores

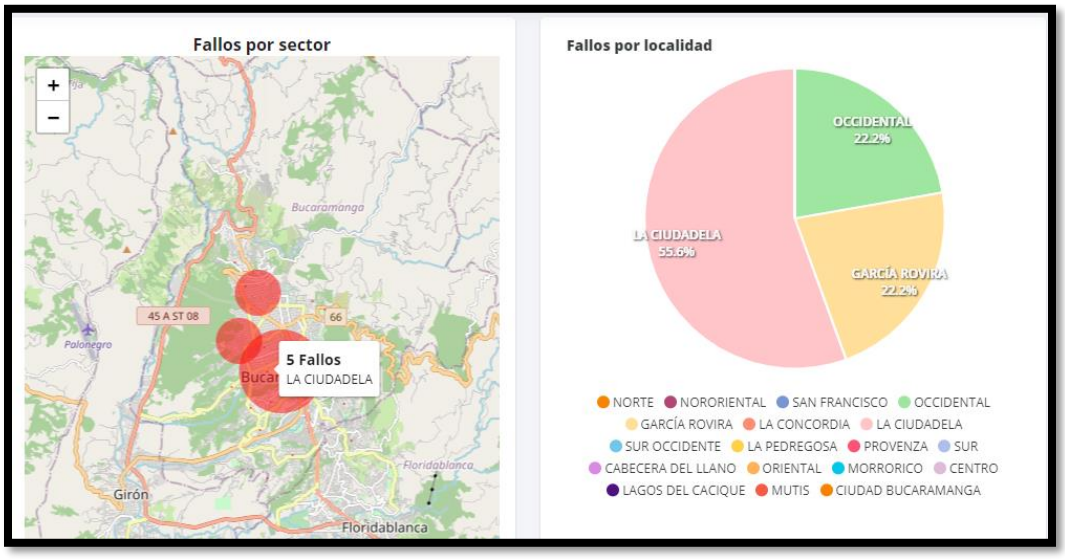


Figura 64.

Gráficas de tiempo de respuesta y tipos de registros

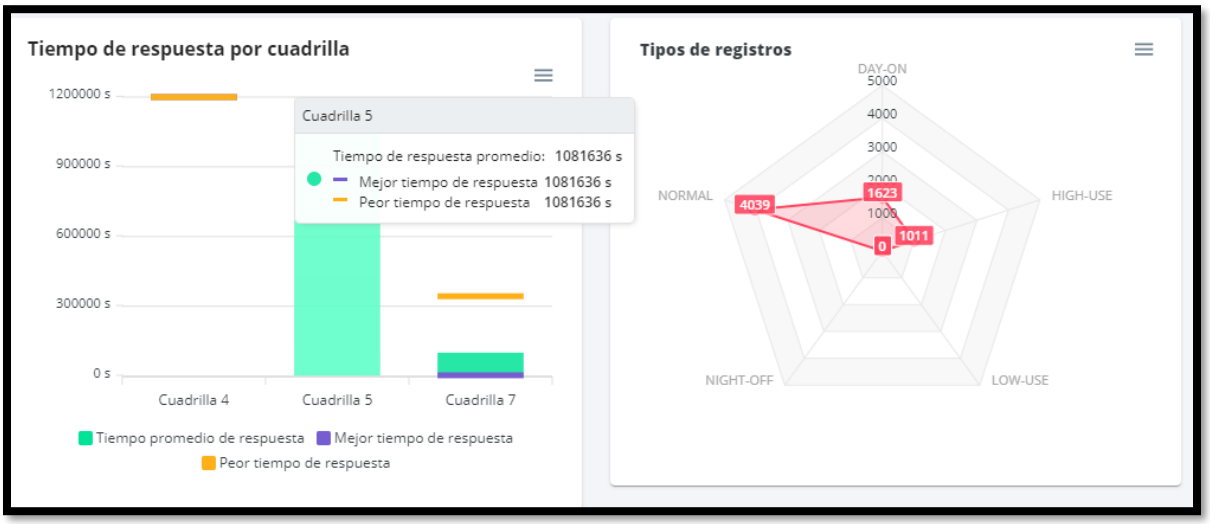


Figura 65.

Estadísticas de dispositivos

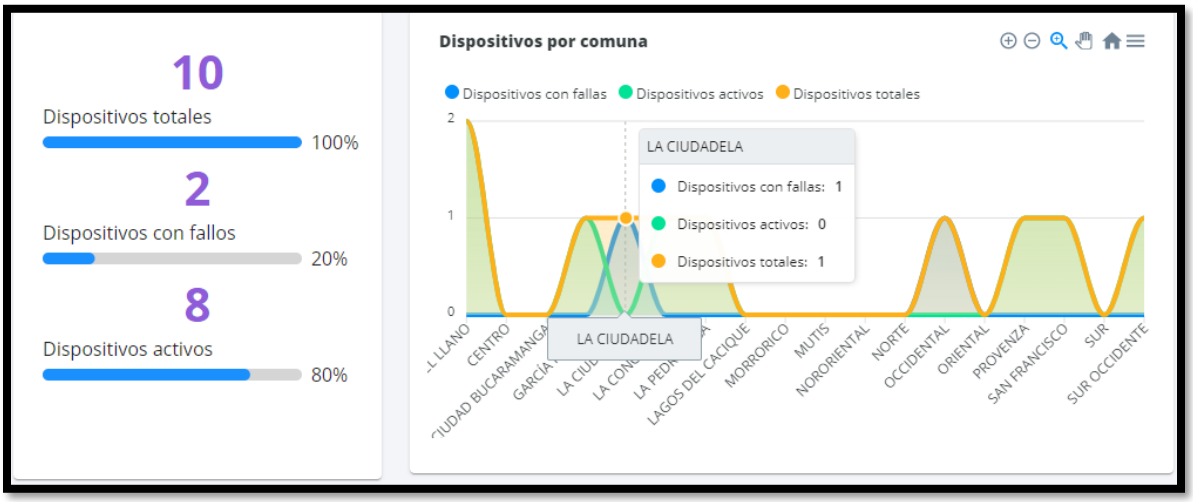


Figura 66.

Clasificación de errores por día

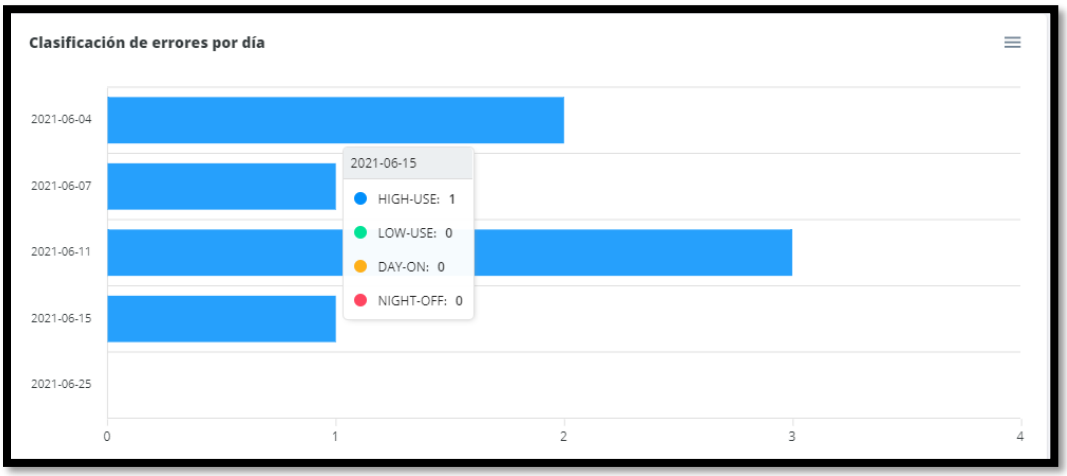
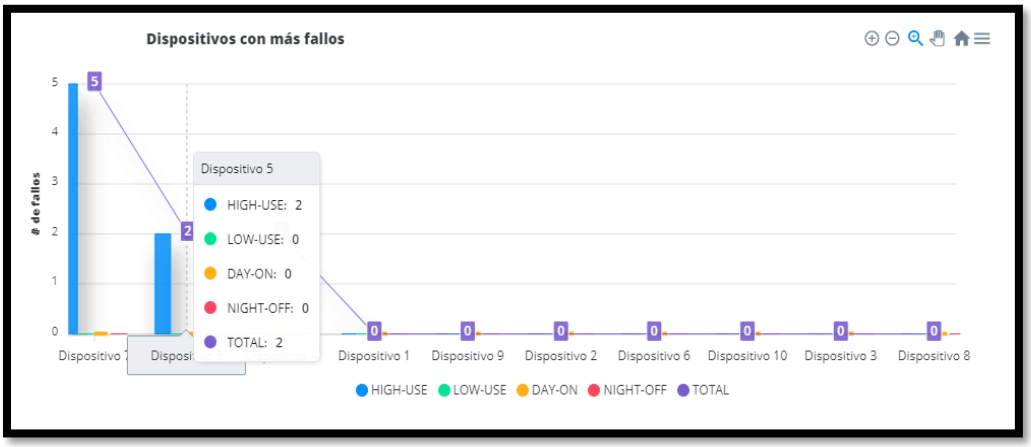


Figura 67.

Gráfica de top de dispositivos que más fallan



4.4.7 Desarrollo Arduino

Para poder cargar el código en la ESP32 se necesitó de poder descargar la placa de “ESP32 Dev Module” con ella y además de descargar las librerías necesarias como lo fue el del sensor

INA219, esta librería se llama <Adafruit_INA219.h>, así como la librería para la conexión por Wifi que lleva el nombre de <WiFi.h> y una librería más, la cual me permite obtener una conexión por medio del protocolo MQTT con el ESP32, esta librería es <PubSubClient.h>, con estas librerías y las respectivas líneas de código se puede cargar el proyecto a la placa del ESP32 por medio de un cable micro USB. Se debe destacar que el ID del dispositivo debe estar en el código Arduino, y este debe de coincidir con la que se encuentra en la base de datos, es decir que hay que tener presente a cuál dispositivo se le va a cargar el código para que no cargue con un ID erróneo.

Figura 68.

Porción de código en Arduino

```
client.loop();

float busVoltage_V = 0;
float voltage_V = 0;
float shuntvoltage = 0;
float current_mA = 0;
float power_mW = 0;

const int idDevice = 4;

long now = millis();
if(now-lastMsg >500){
  lastMsg = now;

  busVoltage_V = ina219.getBusVoltage_V();
  shuntvoltage = ina219.getShuntVoltage_mV();
  current_mA = ina219.getCurrent_mA();
  power_mW = (ina219.getPower_mW())/1000;
  voltage_V = busVoltage_V + (shuntvoltage / 1000);

  String to_send = String(shuntvoltage) + "," + String(current_mA) + "," +

  to_send.toCharArray(msg, 40);
  Serial.print("Publicamos mensaje -> ");
  Serial.println(msg);
  client.publish("values", msg);
  delay(20000);
}
```

4.5 Paso a producción y pruebas

Para el paso a producción fue necesario utilizar el software de FileZilla para incorporar nuestro proyecto en el servidor de AWS, primeramente generamos un compilado del proyecto en react con el comando de npm run build, esto me genera una carpeta build con todos los archivos adentro, estos los copiamos y lo trasladamos a la carpeta de public_html del proyecto de node JS, hay que tener en cuenta que la variable REACT_APP_API_URL que se encuentra en el archivo .env.production de react debe estar apuntando al valor de http://alumbradobga.ga:3000/api. Dentro de la instancia de AWS nos dirigimos a la carpeta de /home/admin/web/alumbradobga.ga/public_html, y allí copiamos todo nuestro proyecto con la ayuda de FileZilla, y después de que emqx este corriendo y pm2 también se esté ejecutando nos dirigimos a nuestro navegador web para observar que incorporando la dirección de alumbradobga.ga todo esté funcionando correctamente. Una vez adentro podemos navegar libremente por la página web observando que todo el proceso se realizó con éxito.

5. Resultados

Para los resultados nos basamos en las seis fases propuestas en la realización del proyecto, con ello apostamos en la recopilación, análisis, comprensión y puesta en ejecución de toda la información obtenida, los resultados obtenidos según las actividades planteadas fueron:

Tabla 6.

Fase 1 capacitación tecnológica y levantamiento de información

Fase 1: Capacitación tecnológica y levantamiento de información	
Capacitación de Bases teóricas y estado del arte.	Cumplida
Análisis de las tecnologías que se pueden utilizar para la realización del proyecto.	Cumplida

Tabla 7.

Fase 2: análisis y definición de arquitectura

Fase 2: Análisis y definición de arquitectura	
Redacción de requerimientos de la plataforma.	Cumplida
Redacción de requerimientos de la arquitectura de IoT.	Cumplida

Tabla 8.

Fase 3: diseño de la plataforma

Fase 3: Diseño de la plataforma	
Mockup de Base de datos y aplicación web.	Cumplida
Arquitectura de infraestructura tecnológica del sistema.	Cumplida
Circuitos de conectividad de los dispositivos.	Cumplida

Figura 69.

Conexión Físico ESP

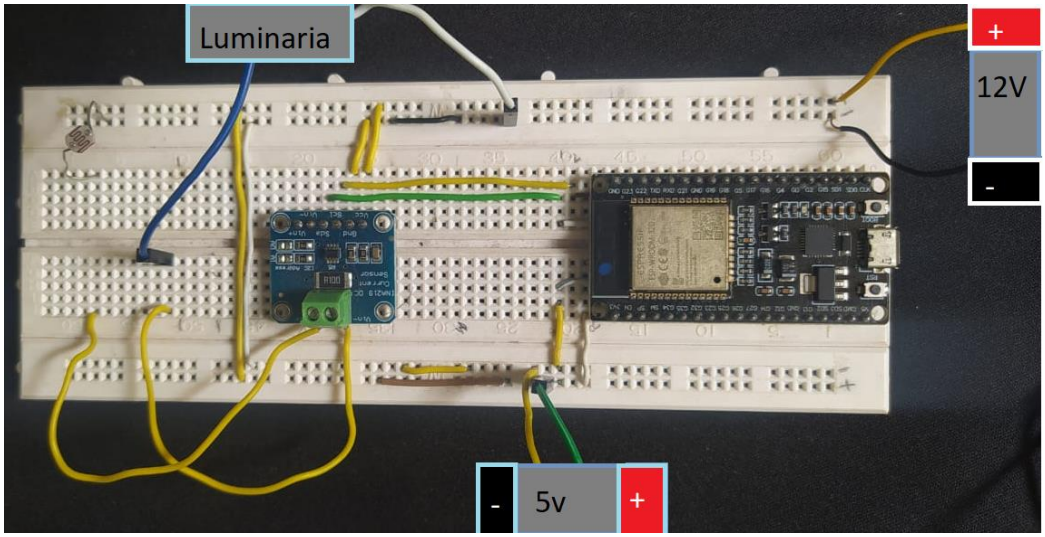


Tabla 9.

Fase 4: desarrollo del prototipo

Fase 4: Desarrollo del prototipo	
Prototipo funcional del sistema.	Cumplida

Figura 70.

Prototipo Funcional del sistema

Figura 71.

Prototipo funcional sistema 2

Tabla 10.

Fase 5: evaluación y pruebas del prototipo

Fase 5: Evaluación y pruebas del prototipo	
Informe de resultados y ajustes para realizar al prototipo.	Cumplida

Tabla 11.

Fase 6: entrega del prototipo final

Fase 6: Entrega del prototipo final	
Prototipo final.	Cumplida

5.1 Plan de pruebas

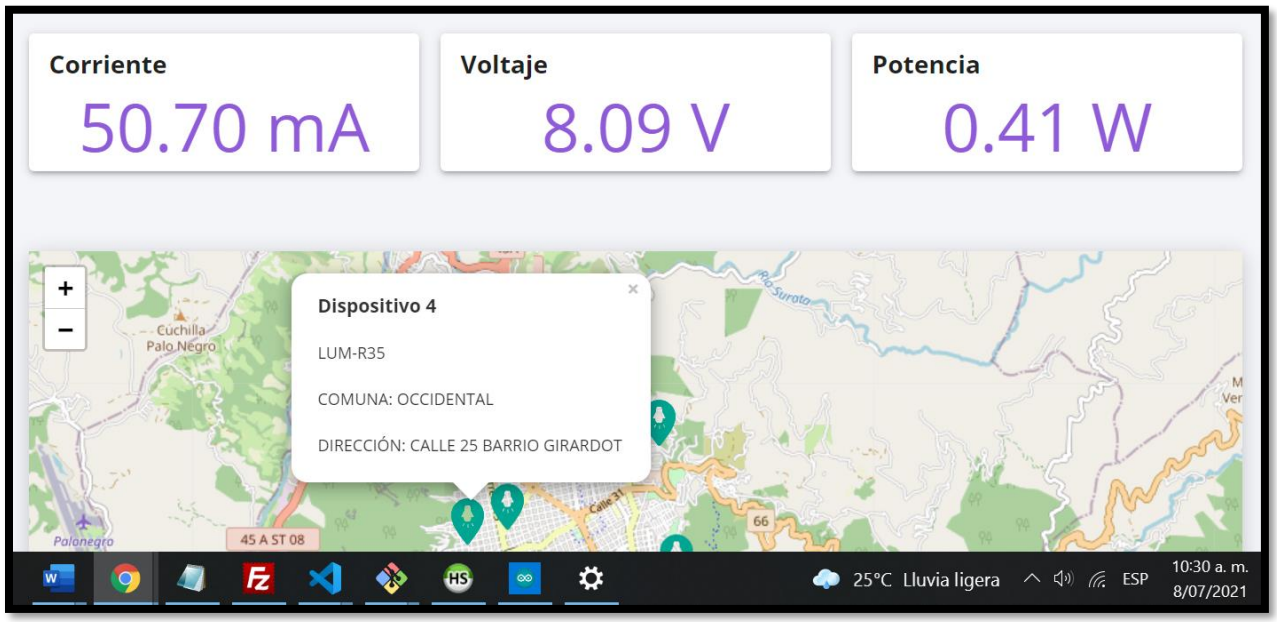
5.1.1 DAY ON

En este plan de pruebas vamos a realizar una validación de su funcionamiento generando un error del cual será el funcionamiento de la luminaria en horas cuando la luz del sol está presente en la zona, esto quiere decir que la fotorresistencia está generando problemas en su funcionalidad, el cual está permitiendo el paso de corriente.

En este ejemplo podemos ver que la luminaria se encuentra consumiendo una corriente de 50.70 mA en un horario de las 10:36 a.m por ende significa que presenta una falla.

Figura 72.

Valores en tiempo real del dispositivo Day On



Al correo se observa que le llega una notificación de la falla con su respectiva información con base a ubicación del dispositivo.

Figura 73.

Notificación de falla DayOn

Y en la dashboard se presencia que hay una falla pendiente por arreglar, con la información del nombre del dispositivo, el tipo de falla, el reporte con fecha y hora y la dirección donde se encuentra

Figura 74.

Reporte de las fallas pendientes Day On

Con ello una vez el daño se encuentre arreglado, el líder de cuadrilla puede oprimir en el cuadro verde para validar que ya se está funcionando de manera correcta

5.1.2 NIGHT OFF

Para las fallas que se presentan por que la luminaria se encuentre sin funcionamiento en la noche, se da porque la corriente que pasa por el sensor es menor a 1 mA, para que ocurra esta falla debe de estar en el horario de 6 de la tarde a 6 de la mañana.

En el siguiente ejemplo podemos presenciar que los valores de corriente que arroja son menores al umbral de funcionamiento, además de que se encuentra en un horario de 11:58 de la noche, generando así un reporte de fallo por luminaria apagada.

Figura 75.

Valores en tiempo real del dispositivo Night Off

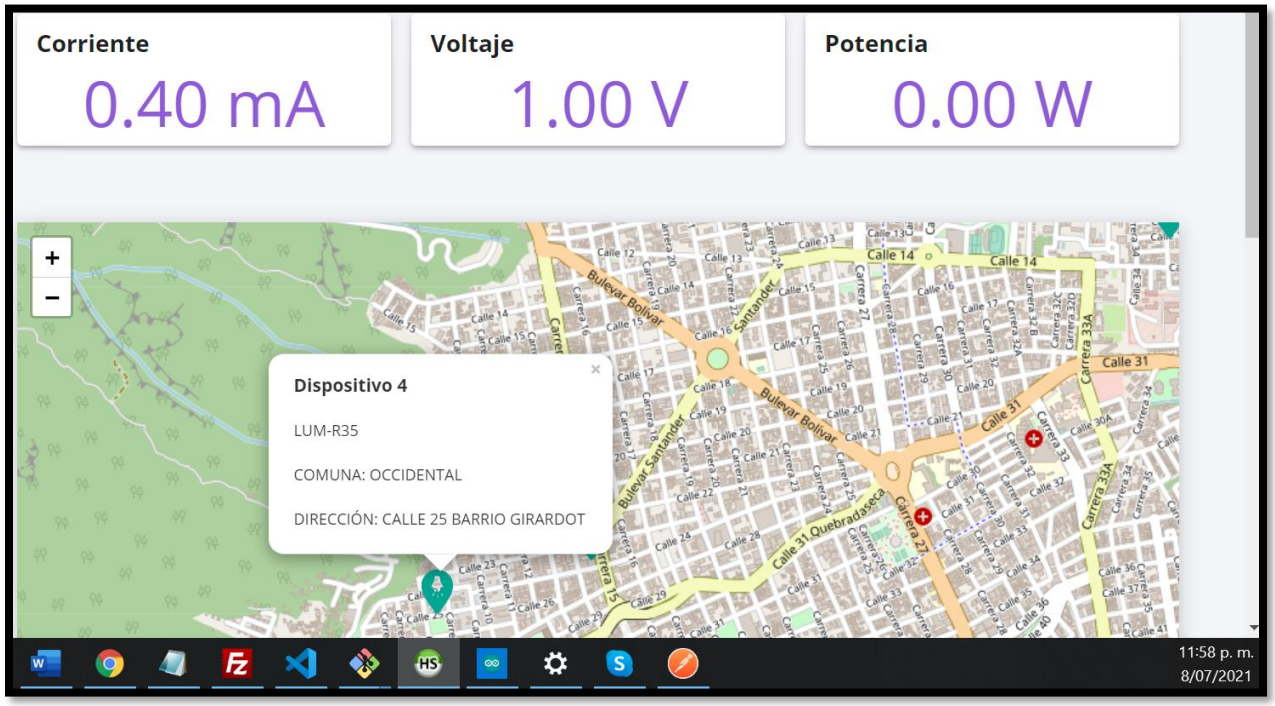


Figura 76.

Reporte de las fallas pendientes Night Off

5.1.3 HIGH USE

En el reporte de falla por consumo elevado de corriente, simulamos un alto consumo de corriente como se puede observar en la figura 77, del cual un consumo por encima de 300 mA, me generará un elevado consumo de corriente, esta prueba se dio con dos luminarias conectadas en paralelo, para ello conectamos un dispositivo de más para que consumiera más corriente y con ello se pudiese reportar esta falla, para que ocurra esta notificación se debe estar en el horario de 6 de la tarde a 6 de la mañana.

Figura 77.

Valores en tiempo real del dispositivo High Use

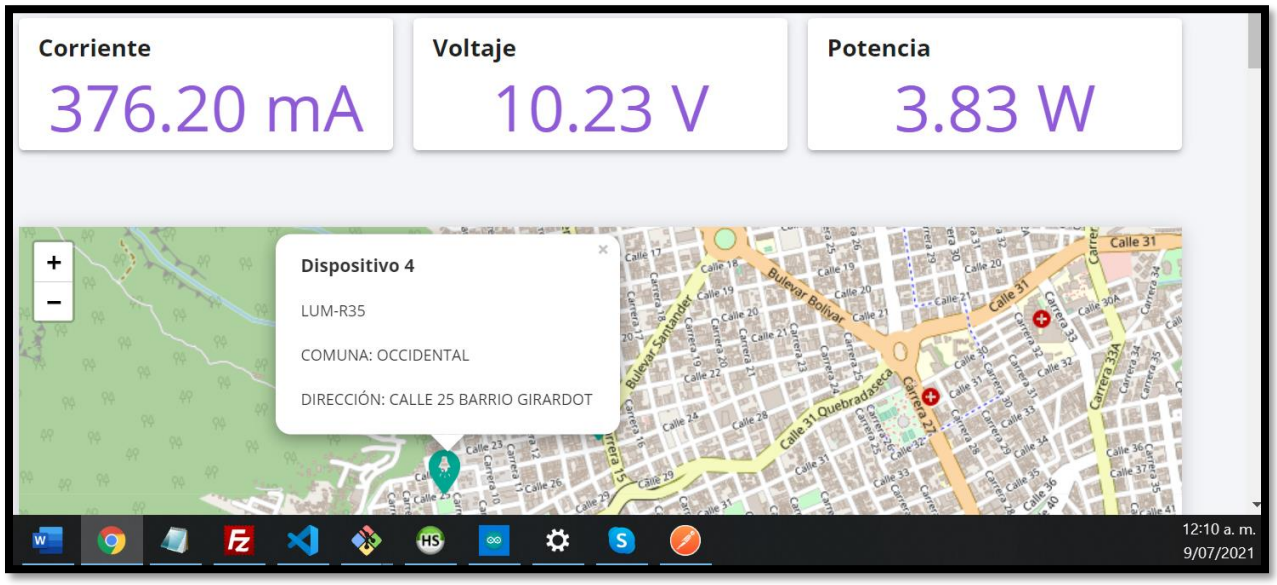


Figura 78.

Reporte de las fallas pendientes High Use

Figura 79.

Maqueta con falla por High Use

5.1.4 LOW USE

En el último caso de reporte de fallas, se da por un consumo inferior a 250 mA, para ello apagamos una luminaria, dejando solo encendida una como se observa en la figura 82. Al igual que el anterior reporte, este se debe presentar en el horario de 6 de la tarde a 6 de la mañana.

Figura 80.

Valores en tiempo real del dispositivo Low Use

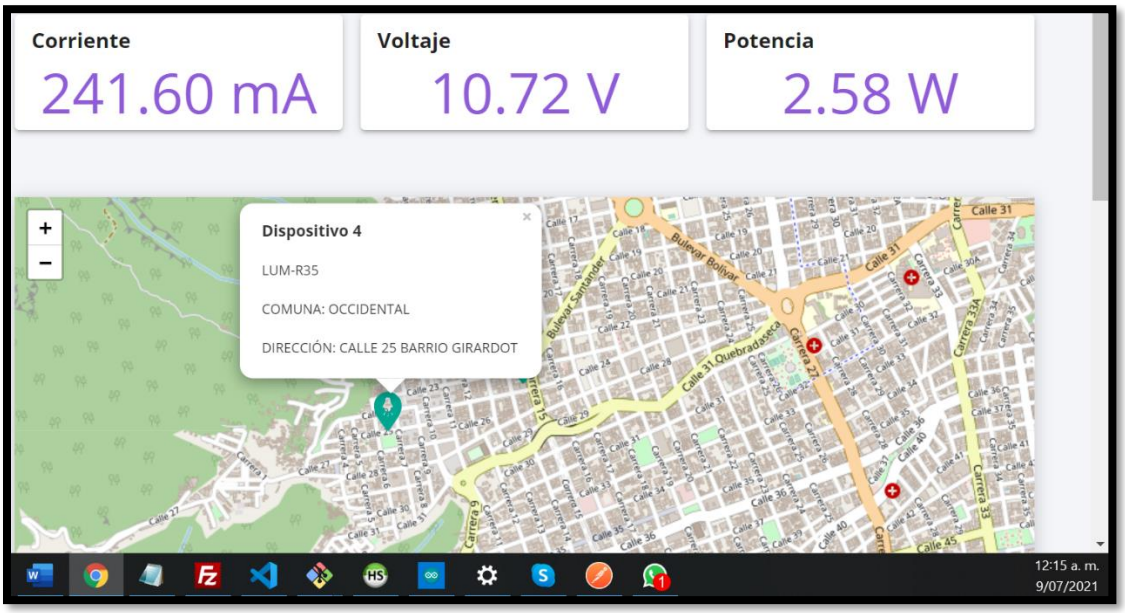


Figura 81.

Reporte de las fallas pendientes Low Use

Figura 82.

Maqueta con falla de Low Use

5.1.5 Recuperar contraseña

Para el caso en el que el usuario olvide su contraseña deberá comunicarse con el administrador de la plataforma, este es el único que tiene la opción de cambiar contraseña en el módulo de gestión de usuarios. Para ello comprobamos que el login funciona y avisa de las credenciales incorrectas, luego iniciamos sesión con un usuario administrador y este cambia la contraseña de otro usuario, después iniciamos sesión con el usuario al que le cambiaron la contraseña y probamos que todo funciona bien.

Figura 83.

Login con credenciales incorrectas

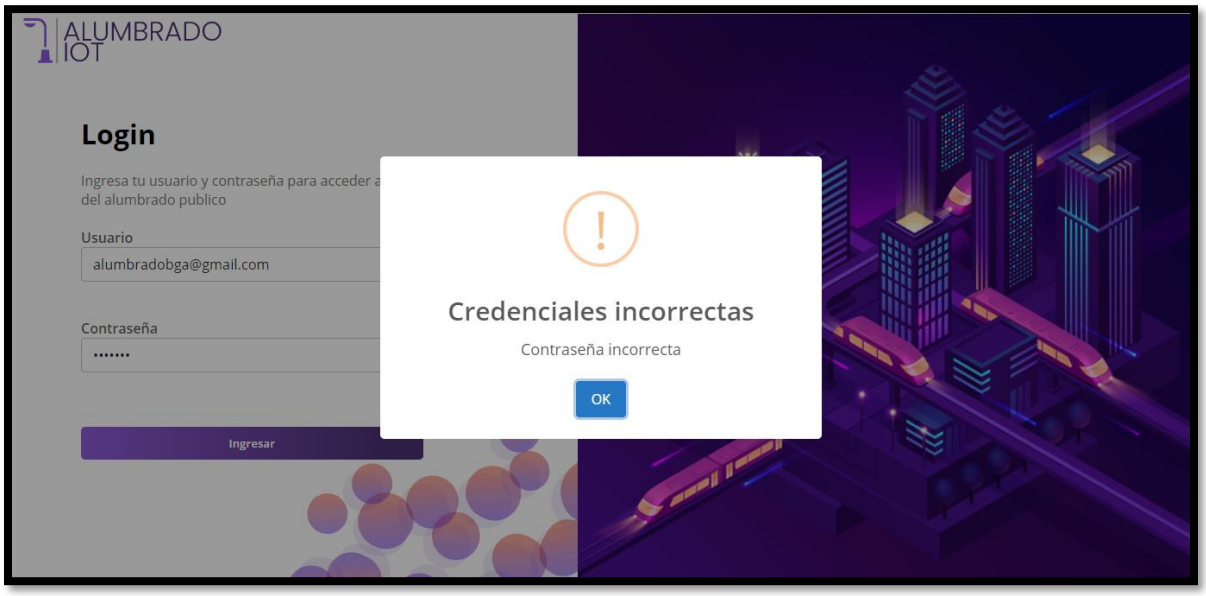


Figura 84.

Cambio de contraseña

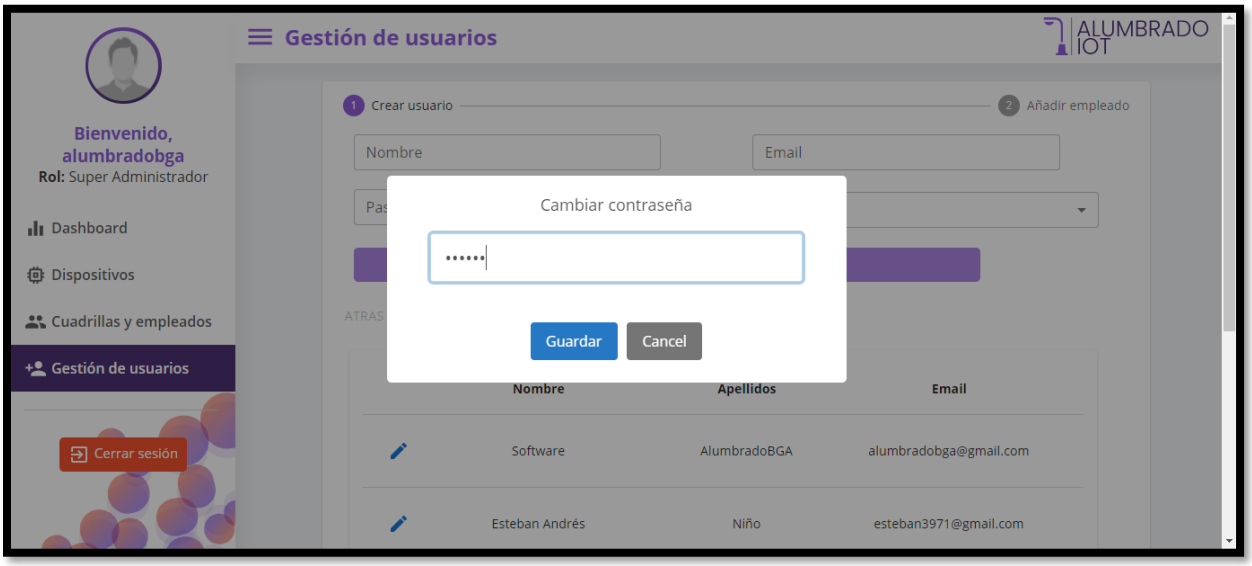
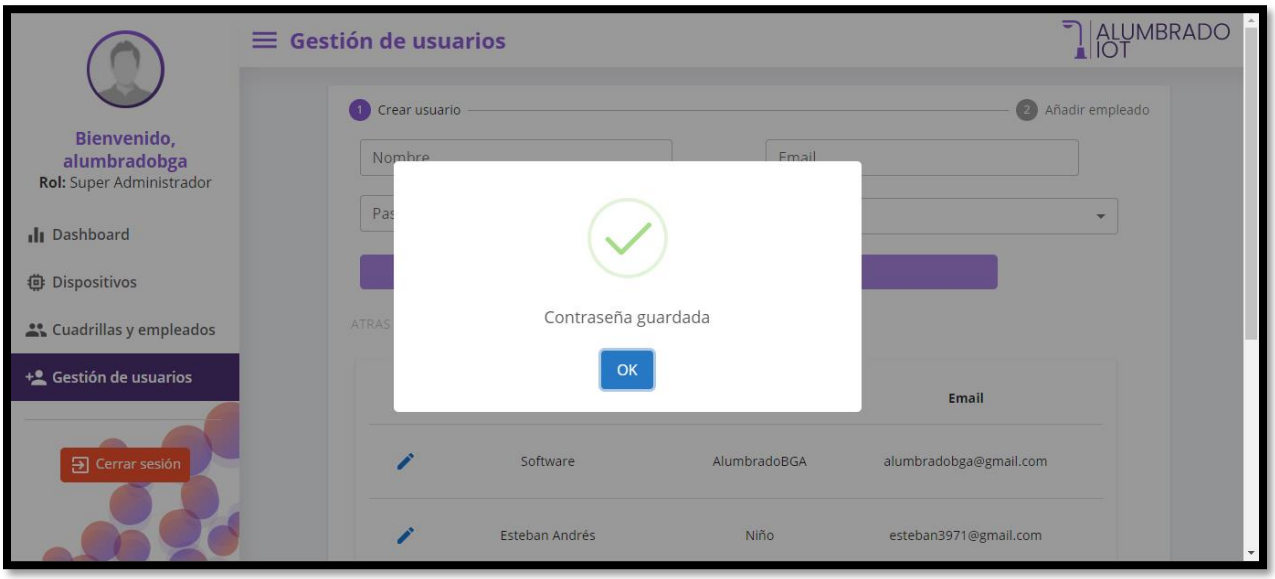


Figura 85.

Confirmación de guardado



6. Conclusiones

El proyecto nació con la finalidad de desarrollar una plataforma que permitiera el control y reporte de fallas en el alumbrado público con el objetivo de mejorar los tiempos de respuesta en el arreglo de averías que presentan las luminarias pues estas son causantes de inseguridad, accidentes y altos costos de consumo.

Se observa que se cumplieron los objetivos propuestos en un principio logrando diseñar y desarrollar un prototipo funcional para la problemática del control de fallos en el alumbrado público. Esto se logró gracias al planteamientos iniciales de una metodología de trabajo y el seguimiento de cada una de sus fases.

Se identificaron los diferentes conceptos que el proyecto requería para su realización y se hizo la respectiva investigación de estos, en esta fase se pudieron definir la variedad de tecnologías que existían para suplir cada requerimiento del proyecto y tener una visión más clara en cuanto a las necesidades de este.

De las tecnologías estudiadas se eligieron las que permitieran un desarrollo más eficiente y lograran cubrir los requerimientos de la plataforma de manera óptima. La fase de capacitación fue clave para el momento de elegir la infraestructura tecnológica a utilizar, además de la comparación de diferentes tecnologías con el fin de lograr un mejor análisis de sus ventajas y desventajas frente a otras, tomando como un criterio importante el tiempo de desarrollo, pues dominar una tecnología requiere de varias horas de estudio y dedicación, por lo que se optó por utilizar herramientas que fueran familiares para los autores.

En la parte del diseño fueron claves los conocimientos adquiridos a lo largo del programa de ingeniería de sistemas. Se hizo uso de múltiples conceptos explicados en asignaturas como ingeniería del software, bases de datos, sistemas de información, sistemas digitales por nombrar algunas, pues no es suficiente con aprender una tecnología, se necesita planificar el desarrollo detalladamente y seguir una estructura bien definida para garantizar un software de alta calidad con buenas prácticas implementadas.

En la etapa de desarrollo y evaluación del prototipo se fueron verificando los requerimientos definidos y desarrollando pruebas en conjuntos para probar el correcto funcionamiento en diferentes escenarios, esto con el fin de registrar posibles fallos e implementar acciones correctivas inmediatamente. Por medio de las pruebas realizadas se concluyó que la plataforma satisface los objetivos definidos al inicio del proyecto de brindar una solución para el control y reporte de fallas en el alumbrado público.

La construcción de una maqueta escala del sistema fue de vital importancia para el desarrollo de las pruebas y verificación de requerimientos del prototipo. Además de aportar una visión más realista de cómo sería implementar la plataforma en un entorno real y que factores se deben tener en cuenta para su correcto funcionamiento.

7. Recomendaciones

Se pudo observar que él envió de notificaciones puede ser más practico si se realiza por medio de un mensaje de texto al celular, ya que esto evita que el usuario tenga que estar conectado a internet en todo momento, con ello también existe la necesidad de que los dispositivos estén conectados a una red WiFi, las 24 horas del día los siete días de la semana para que pueda estar funcionando bien, existen diferentes opciones para migrar a otro tipo de conectividad como es la red Lora o SigFox que me permiten una conectividad WAN privada de baja transmisión como lo es para envió de datos, actualmente estas redes no tienen suficiente cobertura en el territorio Colombiano pero me permiten conectividad sin necesidad de estar conectado a una red móvil tradicional, evitando una dependencia total a estos mismos. Por ello si se tienen los recursos necesarios el utilizar este tipo de comunicación es más eficiente para él envió de datos planos.

Cabe resaltar también que el cambio de contraseñas y configuraciones al perfil del usuario, todo está bajo la responsabilidad del administrador, es bueno que tareas como cambio de contraseña o recuperación de contraseña lo puedan realizar los mismos usuarios.

Se recomienda realizar un módulo para que el reporte de fallas se pueda visualizar en móvil para más comodidad del usuario, es decir por medio de un aplicativo móvil, no solo por un aplicativo web, además, también implementar la opción de cambiar contraseña por parte de cada usuario, así como el cambio de la imagen de perfil.

Referencias Bibliográficas

Alligator.io. (2020, 2 diciembre). Cómo funcionan las acciones asíncronas de Redux con Redux Thunk. Recuperado el 20 de Junio de 2021 https://www.digitalocean.com/community/tutorials/redux-redux-thunk-es

Amazon. (2021). SQL (relacional) en comparación con NoSQL (no relacional) https://aws.amazon.com/es/nosql/

Arquitecturas IoT. (2018, 11 noviembre). Aprendiendo Arduino. Recuperado el 21 de Julio de 2021 https://aprendiendoarduino.wordpress.com/tag/modelos-de-capas-iot/

Baquero, J. (2017). Single-Page Application, todo un website desde única página. Arsys.es. Recuperado el 29 de abril de 2019, https://www.arsys.es/blog/programacion/diseno-web/spa-unica-pagina/.

EPM. (2020, 14 febrero). EPM recibió sello de excelencia del MINTIC por su proyecto piloto de telegestión del alumbrado público en Medellín. https://www.epm.com.co/site/home/sala-de-prensa/noticias-y-novedades/epm-recibio-sello-de-excelencia-del-mintic-por-su-proyecto-piloto-de-telegestion-del-alumbrado-publico-en-medellin

ESSA. (2020, 17 febrero). Alumbrado público. Clientes. https://www.essa.com.co/site/clientes/empresas/normatividad-empresas/alumbrado-p250blico

EVA. (2018, 1 junio). Decreto 943 de 2018 - EVA - Función Pública. Gestor Normativo. https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=86680

Gerber, A. (2017, 4 octubre). Simplifique el desarrollo de sus soluciones de IoT con arquitecturas de IoT. IBM Developer. https://developer.ibm.com/es/articles/iot-lp201-iot-architectures/

González, A. J. (2017, junio). IoT: Dispositivos, tecnologías de transporte y aplicaciones. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64286/3/agonzalezgarcia0TFM0617memoria.pdf

González, A. M., & Pinilla, M. E. (2009). Gestión de mantenimiento para alumbrado público. https://docplayer.es/39395065-Gestion-de-mantenimiento-para-alumbrado-publico-gonzalez-gelvez-aldo-marco-pinilla-marquez-martin-emilio.html

Iluminet. (2019, 4 julio). BrightSites, el poste inteligente para una ciudad conectada. https://www.iluminet.com/brightsites-poste-inteligente-signify/

Ministerio de Minas y Energía & Unidad de Planeación Minero Energética. (2016, diciembre). Plan de Acción Indicativo de Eficiencia Energética –pai Proure 2017 - 2022. http://www1.upme.gov.co/DemandaEnergetica/MarcoNormatividad/PAI_PROURE_207-2022.pdf

National Institute of Standards and Technology. (2011, septiembre). The NIST Definition of Cloud Computing (N.o 800-145). https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Philips. (s. f.). CityTouch. Recuperado 17 de agosto de 2020, de https://www.lighting.philips.es/sistemas/sistemas-de-iluminacion/citytouch

Reyes, M. A. (2016, marzo). Diseño de la investigación: beneficios en la implementación del sistema de alumbrado público tipo led, para la población del municipio el tejar, Chimaltenango. http://biblioteca.usac.edu.gt/tesis/08/08_3351_IN.pdf

Rouse, M. (2017). REST (Representational State Transfer). Searchmicroservices.techtarget.com. Recuperado el 29 de abril de 2019, https://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer.

SAP. (s. f.). ¿Qué es Internet de las cosas y cómo mejora la tecnología? Recuperado 17 de agosto de 2020, de https://www.sap.com/latinamerica/trends/internet-of-things.html

Tvilight. (2020, 28 Julio). Market leader in intelligent lighting. Tvilight - Empowering Intelligence. https://www.tvilight.com/