eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ Trabajo Fin de Grado ▪

Ingeniería de Software

Plataforma web para integración de alarmas

Smart City

Aritz García Garmendia

2022

Dirección

Alfredo Goñi Sarriguren

Resumen

La empresa LEYCOLAN, ubicada en Usurbil, ofrece soluciones innovadoras en lo que respecta a la gestión de la iluminación. Se dedica a diseñar, fabricar e instalar tanto componentes de hardware como software para sistemas y redes de alumbrado que respondan a las necesidades básicas de instituciones y empresas en el plano económico, operativo y ambiental.

Para asegurar que dichas instalaciones funcionen correctamente durante el mayor tiempo posible, existe un sistema de alarmas en cada instalación que notifica mediante correo al propio instalador de la misma y a los responsables de mantenimiento en caso de fallo en alguna luminaria, nodo, sensor, etc. Tal y como está actualmente diseñado el sistema de notificaciones y de alarmas, resulta poco sencillo desempeñar las tareas de monitorización y reparación de los nodos o luminarias de las instalaciones , ya que hay varios tipos de las mismas y cada tipo funciona de manera distinta.

El objetivo principal de este proyecto es desarrollar una aplicación web que facilite la gestión de las alarmas enviadas por los diferentes tipos de instalaciones, y ofrezca la posibilidad de monitorizar múltiples infraestructuras en una misma plataforma de manera intuitiva y sencilla.

En esta memoria se detalla cómo se ha llevado a cabo el desarrollo de la aplicación, comenzando con una introducción. Se describen los antecedentes y las motivaciones del proyecto, seguido de la planificación del mismo. En los siguientes apartados se explica tanto el diseño como la implementación de la solución software y las tecnologías utilizadas. Finalmente se concluye con algunas de las pruebas realizadas, un vistazo al seguimiento y control del proyecto y algunas conclusiones y aspectos a mejorar.

Índice de contenidos

Lista de Figuras y Tablas

Figuras

Tablas

1

Introducción y Antecedentes

En este capítulo inicial se describe el contexto tanto del trabajo realizado como de la organización junto a la cual se ha desarrollado el proyecto para obtener una idea aproximada de las motivaciones del proyecto y las características de la aplicación como producto del desarrollo del mismo.

1.1. Introducción

Al inicio del periodo de prácticas que realicé durante este último verano en la empresa LEYCOLAN, los responsables de la organización me propusieron realizar un trabajo de fin de grado relacionado con todo lo trabajado durante esas semanas.

La plataforma web cuyas características se analizan con detalle en esta memoria, es el resultado de dicho proyecto. Para entender mejor de dónde surge la idea de desarrollar un proyecto como este, es necesario conocer, aunque sea a niveles básicos, en qué consiste el trabajo realizado por la organización, y cuales son las motivaciones que generan la necesidad de desarrollar esta aplicación.

1.1.1. Contexto de la organización

LEYCOLAN S.A.L. [1] es una empresa tecnológica ubicada en Usurbil, y se dedica al diseño, fabricación e instalación de componentes tanto hardware como software para instalaciones de alumbrado repartidas por diversas localidades de Gipuzkoa, llegando incluso a desarrollar proyectos en el extranjero, en países como Cuba.

En la organización trabajan con múltiples clases de instalaciones lumínicas, caracterizadas por poseer cada una de ellas un ordenador central, denominado *cabecera*, a través del cuál se lleva a cabo la gestión de los dispositivos conectados a la red de la instalación. Algunos de esos dispositivos pueden ser nodos inteligentes o lámparas, sensores e incluso cámaras de vídeo.

Tanto la regulación de dichos dispositivos como la monitorización del estado de los mismos se lleva a cabo gracias a las mencionadas cabeceras, que normalmente consisten en un dispositivo electrónico que contiene un sistema operativo linux o una serie de comandos mediante los cuales se puede llevar a cabo la gestión de toda la instalación conectada a estos ordenadores centrales, dependiendo del tipo de instalación.

Hasta el momento el proyecto se centra en gestionar dos de esos tipos de instalación, los cuales se describen a continuación. Estos tipos de instalaciones son de *banda ancha* y de *banda estrecha*, y su función final es prácticamente la misma, pero difieren en varios aspectos como pueden ser: la clase de cabecera que utilizan, las funcionalidades que ofrecen y la manera en la que se puede comunicar con estas cabeceras para integrar estas funciones con otras plataformas.

1.1.2. Tipos de instalaciones

Como bien se ha explicado en el punto anterior, la empresa LEYCOLAN trabaja con distintos tipos de instalación, y cada una tiene sus propias características, aunque todas tienen un sistema en común. Este sistema se trata de un sistema de alarmas, que funciona de la siguiente manera: cuando en una instalación alguno de los nodos o dispositivos presenta algún error, ya sea de conexión, potencia o de otro tipo, comunica mediante correo electrónico a los instaladores y responsables de la instalación informando de que algo no está funcionando correctamente.

Cada instalación gestiona este sistema de manera un tanto diferente y posee elementos hardware específicos y funcionalidades distintas según el tipo, por lo que conviene explicar algunos

conceptos clave en lo que respecta a cada clase de instalación. En este proyecto se han desarrollado funcionalidades teniendo en cuenta dos tipos: de banda ancha y de banda estrecha.

1.1.2.1. Instalaciones de banda ancha

Se caracteriza por tener como ordenador central un módulo de cabecera para control de instalaciones por tecnología *PLC*[1] en Banda *Ancha*[2] , y en este existen dos tipos de red: una red PLC y otra red Ethernet.

Fig.1.1. *Cabecera PLC de banda ancha.*

Dicha cabecera se compone de dos partes fundamentales:

- Módulo *BPLC Head*: actúa de pasarela entre la red PLC y la red Ethernet, permite la interconexión de los nodos conectados a la red eléctrica y los equipos *ServiLAN* o cualquier otro equipo de la red Ethernet.

- Módulo *ServiLAN*: contiene la parte importante del software del sistema. Su función es gestionar los dispositivos conectados a la red eléctrica a través de sus correspondientes nodos y cada uno de estos nodos tiene su propia dirección IP única dentro de la red de la cabecera. Es el encargado de controlar y gestionar el alumbrado en base a la configuración programada. Incorpora una aplicación sencilla y de uso intuitivo que permite la configuración, control y visualización del estado de la instalación y sus nodos.

1. Power Line Communications (PLC), es un término inglés que puede traducirse por comunicaciones mediante línea de potencia y que se refiere a diferentes tecnologías que utilizan las líneas de transmisión de energía eléctrica convencionales para transmitir señales con propósitos de comunicación.
2. En telecomunicaciones, se conoce como banda ancha a cualquier tipo de red con elevada capacidad para transportar información que incide en la velocidad de transmisión de esta.

Las instalaciones de este tipo consisten principalmente de nodos inteligentes conectados a cada uno de los dispositivos de iluminación, diseñados para monitorizar y controlar el punto de luz al que están conectados a través de comunicación *Power Line*.

Fig.1.2. *Nodos inteligentes de instalaciones de banda ancha.*

Aunque también pueden conectarse a la red de la cabecera dispositivos con distintas funcionalidades, como lo pueden ser los radares o sensores de movimiento como el que se muestra en la siguiente figura, que regulan la iluminación de las luminarias conectadas según las lecturas que este reciba.

Fig.1.3. *Sensor de microondas para detección de vehículos y personas a larga distancia.*

Para la comunicación e integración con plataformas externas se puede hacer uso de una API con comandos expresados en formato JSON que permite la ejecución de diversas funcionalidades para la gestión y el control de cada instalación. El funcionamiento de la API se detalla en el capítulo de Implementación.

1.1.2.2. Instalaciones de banda estrecha

Se caracteriza por tener como ordenador central un módulo de cabecera para control de las instalaciones por tecnología PLC en Banda *Estrecha*[3].

Fig.1.4. *Cabecera PLC de banda estrecha.*

Dicho módulo se instala dentro del cuadro eléctrico donde se derivan las líneas de alumbrado. Se comunica con los nodos conectados gracias a la tecnología *Power Line* (o PLC) para gestionar y almacenar en la nube la información y el estado de la instalación con protocolo *MODBUS*[4].

Este tipo de instalaciones también consisten en una serie de nodos inteligentes que se conectan a luminarias para regular y controlar su funcionamiento a través de la tecnología Power Line, pero son diferentes a los de banda ancha puesto que se comunican mediante conexiones de banda estrecha, como el nombre del tipo de la instalación indica.

Fig.1.5. *Nodos inteligentes de instalaciones de banda estrecha.*

Para la comunicación e integración con plataformas externas se puede hacer uso de una serie de APIs que permiten la ejecución de diversos comandos para la gestión y el control de cada instalación. El funcionamiento de estas APIs se detalla en el capítulo de Implementación.

---

3. Las conexiones de banda estrecha en el mundo de las conexiones a Internet hacen referencia a un tipo de conexión que utiliza un ancho de banda muy reducido.
4. Modbus es un protocolo de comunicaciones situado en los niveles 1, 2 y 7 del Modelo OSI, basado en la arquitectura maestro/esclavo (RTU) o cliente/servidor (TCP/IP), diseñado en 1979 por Modicon para su gama de controladores lógicos programables (PLCs). Convertido en un protocolo de comunicaciones estándar de facto en la industria, es el que goza de mayor disponibilidad para la conexión de dispositivos electrónicos industriales.

1.2. Motivación y objetivo principal

Una de las cuestiones principales a la hora de realizar una aplicación como la que se ha desarrollado a través de este proyecto gira en torno al sistema de alarmas mencionado anteriormente. Tal y como se encuentra actualmente la gestión de las alarmas de las diferentes instalaciones controladas por la empresa, resulta complejo llevar a cabo los procesos de monitorización y control. Esto se debe a que siempre que haya errores de algún tipo en una instalación, la cabecera de esta envía por correo cada día, varias veces al día, las alarmas correspondientes a los correos de todos los responsables o clientes de las mismas.

El hecho de recibir alarmas idénticas en más de una ocasión, y muchas alarmas de cada cabecera, dificulta bastante a los técnicos las tareas de reparación de los errores que puedan surgir. Para que un técnico o responsable se haga cargo del arreglo, puede hacerlo de dos maneras: acceder en remoto a la cabecera (mediante terminales como puTTy y similares) o, si no llega a deducir o arreglar el problema por mala conexión o cobertura, debe acudir presencialmente al lugar de la instalación y conectarse físicamente a la red para ver qué ocurre y tratar de solventar los problemas.

El proyecto que se me propuso llevar a cabo junto con la empresa consistiría en desarrollar una plataforma web, accesible mediante el navegador, que permita a los clientes de la organización monitorizar y gestionar las alarmas de sus instalaciones en un mismo lugar, incluso aunque un mismo cliente trabaje con instalaciones de diferente tipo.

La parte de la integración de las alarmas con la plataforma web se realizaría mediante una serie de APIs que posee cada una de las instalaciones, y se explican con detalle en la sección de Implementación. Como no se me impuso ninguna restricción en cuanto a las tecnologías a utilizar para el desarrollo de esta aplicación web, decidí que el proyecto iba a comenzar con una investigación acerca de las tecnologías más utilizadas en el ámbito profesional a día de hoy, y a partir de ahí diseñar e implementar lo que sería la solución software final.

El objetivo principal de este proyecto no sería otro que el de facilitar a los usuarios y clientes de LEYCOLAN el proceso de gestión, monitorización y control del estado de sus instalaciones y alarmas a través de una plataforma fácilmente accesible, intuitiva y sencilla de utilizar.

2

Planificación

Este capítulo describe la planificación inicial del trabajo de fin de grado. En ella se describen el alcance y los objetivos del proyecto, las tareas a desarrollar junto con algunas estimaciones de los periodos de desarrollo y un análisis de riesgos.

2.1. Alcance

El alcance incluye todo el trabajo necesario para producir y entregar una aplicación web que será utilizada por LEYCOLAN, empresa dedicada a diseñar e instalar componentes de hardware y software para el control de instalaciones eléctricas, tanto en exteriores como en interiores. Dichas redes son monitorizadas mediante una serie de pequeños equipos o cabeceras, encargadas de la gestión en remoto y monitorización de los diferentes tipos de instalaciones.

El proyecto se centrará en resolver uno de los problemas actuales de la empresa, y es que cuando un componente o funcionalidad de alguno de estos sistemas falla, la propia cabecera envía alarmas mediante correo a los clientes y técnicos de cada instalación. La cuestión es que la gestión y las notificaciones de las alarmas resulta altamente compleja debido a la implementación actual de dicho sistema de alertas.

Por ello el trabajo consistirá en desarrollar una aplicación web que aúne todas las alertas de las distintas instalaciones de cada usuario en un mismo lugar, haciendo uso de una serie de APIs que tienen comandos para recibir información de las instalaciones.

2.1.1. Objetivos

Objetivo 1. Desarrollar una aplicación web que cumpla con los requisitos del cliente, en este caso la empresa, y se ejecute en sus servidores con la mayor disponibilidad posible.

Objetivo 2. Ofrecer una experiencia agradable y sobre todo intuitiva para cualquier tipo de usuario que vaya a hacer uso de la aplicación.

Objetivo 3. Desarrollar y finalizar la plataforma web dentro del periodo de tiempo establecido en la propuesta del trabajo de fin de grado de la web de la universidad (*14/01/2022*).

Objetivo 4. Realizar la defensa del proyecto realizado ante un tribunal y obtener la mejor valoración posible.

2.1.2. Estructura de descomposición del trabajo (EDT)

Fig.2.1. *Estructura de descomposición de trabajo de este proyecto de fin de grado.*

El paquete de **Tecnologías y Herramientas** recogerá todas las tareas asociadas a la comparación y elección tanto de los lenguajes como entornos de desarrollo a utilizar para la realización del proyecto de la aplicación web.

La rama **Proyecto Software** del EDT recoge todos los paquetes cuyas tareas tienen como fin el desarrollo de la aplicación web. En el paquete de **Backend** se recopilan las tareas relacionadas con las funcionalidades del sistema, la lógica de negocio y la implementación de las clases y paquetes del proyecto software. El paquete de **Frontend** recoge todas aquellas tareas asociadas al diseño e implementación de las interfaces y vistas de la aplicación.

En la rama de **Despliegue Web** se incluyen los paquetes de **Base de Datos**, el cual agrupa todas las tareas de diseño y gestión de los datos de los que la aplicación hará uso, y el paquete de **Solución Web**, que contiene tareas asociadas con el despliegue en remoto de la propia aplicación.

Por último, la rama de **Gestión, Seguimiento y Control** recopila los siguientes paquetes: el paquete de **Documentación**, que aúna las tareas directamente relacionadas con la redacción de documentos necesarios para el desarrollo del proyecto, el paquete de **Control de Calidad**, con tareas dirigidas a la revisión y corrección del trabajo desarrollado, y el paquete de **Testeo**, que contiene tareas para la realización de las pruebas necesarias para controlar el comportamiento de la aplicación en todo tipo de situaciones.

2.2. Descripción de las tareas a realizar

Paquete de Tecnologías y Herramientas (TyH)
TyH.T1. Elección de lenguajes para las funcionalidades (*backend*) y gestión de datos del proyecto.
TyH.T2. Familiarización con lenguajes escogidos para el *backend* del proyecto.
TyH.T3. Elección de lenguajes para el diseño (*frontend*) del proyecto.
TyH.T4. Familiarización con lenguajes escogidos para el *frontend* del proyecto.
TyH.T5. Elección de entorno de desarrollo para el proyecto.
TyH.T6. Familiarización con el entorno de desarrollo para el proyecto.

Paquete de *Backend* (BE)
BE.T1. Diseño de la arquitectura del proyecto.
BE.T2. Implementación de funcionalidades de la aplicación.
BE.T3. Implementación de funcionalidades de Banda Ancha (integración API JSON y comandos Linux).
BE.T4. Implementación de funcionalidades de Banda Estrecha (integración ApiSPLIT y GeoLoc).
BE.T5. Implementación de funcionalidades con acceso a base de datos.

Paquete de *Frontend* (FE)
FE.T1. Diseño de la arquitectura del proyecto.
FE.T2. Implementación de las vistas o páginas de la aplicación.
FE.T3. Implementación del diseño de la aplicación.

Paquete de Base de Datos (BD)

BD.T1. Obtención del alojamiento para la base de datos en el servidor de la empresa.
BD.T2. Diseño de la arquitectura de datos que deberán almacenarse para el correcto funcionamiento de la aplicación.
BD.T3. Creación y diseño de la propia base de datos.
BD.T4. Inserción de datos de los diferentes usuarios y sus instalaciones.
BD.T5. Mantenimiento de la base de datos.

Paquete de Solución Web: despliegue y adaptación a ejecución en remoto **(SW)**
SW.T1. Obtención de espacio de alojamiento para la ejecución y despliegue de la aplicación.
SW.T2. Realización de cambios necesarios en el proyecto para su correcto despliegue en la nube.
SW.T3. Despliegue web de la aplicación para la realización de pruebas con usuarios reales.
SW.T4. Recogida de información con respecto al feedback recibido.

Paquete de Documentación: memoria y otros documentos necesarios **(DOC)**
DOC.T1. Realización de la planificación del proyecto.
DOC.T2. Redacción de la memoria del proyecto.
DOC.T3. Informes de las posibles recogidas de requisitos con el cliente o la empresa.
DOC.T4. Realización del informe Seguimiento y Control.
DOC.T5. Redacción de documentos varios que puedan surgir durante el proyecto.

Paquete de Control de Calidad: revisiones, refactorización de código, etc **(CC)**
CC.T1. Revisión y análisis de las herramientas elegidas.
CC.T2. Revisión y refactorización o limpieza de código en el *backend* del proyecto.
CC.T3. Revisión y refactorización o limpieza de código de *frontend* del proyecto.
CC.T4. Revisión y corrección o actualización de la base de datos.
CC.T5. Revisión y corrección de la documentación.
CC.T6. Revisión y corrección del despliegue web.
CC.T7. Revisión y corrección de los diferentes tests adaptados a cada paquete o aspecto de la aplicación.

Paquete de Testeo: tests en local y para la web **(T)**
T.T1. Tests de código para el *backend*.
T.T2. Tests de código para el *frontend*.
T.T3. Tests y pruebas en la base de datos.
T.T4. Tests y pruebas de la aplicación web en remoto.

2.2.1 Periodo de realización de tareas

OCTUBRE / Paquetes	S3	S4
TyH		
BE		
FE		
BD		
W		
DOC		
CC		
T		

Tabla 2.1. *Calendario de planificación de trabajo correspondiente al mes de octubre.*

NOVIEMBRE / Paquetes	S1	S2	S3	S4
TyH				
BE				
FE				
BD				
W				
DOC				
CC				
T				

Tabla 2.2. *Calendario de planificación de trabajo correspondiente al mes de noviembre.*

DICIEMBRE / Paquetes	S1	S2	S3	S4
TyH				
BE				
FE				
BD				
W				
DOC				
CC				
T				

Tabla 2.3. *Calendario de planificación de trabajo correspondiente al mes de diciembre.*

ENERO / Paquetes	S1	S2
TyH		
BE		
FE		
BD		
W		
DOC		
CC		
T		

Tabla 2.4. *Calendario de planificación de trabajo correspondiente al mes de enero.*

2.2.2 Estimación de tiempos de realización de tareas por paquetes

- **Paquete de Tecnologías y Herramientas (TH)** → 10h.
- **Paquete de *Backend* (BE)** → 90h.
- **Paquete de *Frontend* (FE)** → 80h.
- **Paquete de Base de Datos (BD)** → 20h.
- **Paquete de Solución Web (SW)** → 10h.
- **Paquete de Documentación (DOC)** → 40h.
- **Paquete de Control de Calidad (CC)** → 30h.
- **Paquete de Testeo (T)** → 30h.

Estimación de horas totales: 310h.

2.3. Análisis de riesgos

R1. Existe la posibilidad de que los lenguajes de programación o el entorno de desarrollo escogidos para realizar el proyecto den problemas en cuanto a errores inesperados o mala estimación del tiempo aproximado de realización de diversas tareas debido a que son herramientas poco conocidas en relación a la experiencia previa o no utilizadas anteriormente. Para ello se incluirán varias tareas de selección de dichas herramientas y se realizará una comparativa de ventajas y desventajas para tratar de elegir aquellas que se prevé que facilitarán al máximo el desarrollo.

R2. Al tratarse de una aplicación que posteriormente será implementada en una empresa, puede que el cliente decida incluir o modificar requisitos tanto funcionales como de diseño de la aplicación en mitad del proyecto. Para evitar tener que deshacer el trabajo realizado o modificar notablemente la planificación y el desarrollo de la aplicación, se llevarán a cabo múltiples reuniones y/o comunicaciones vía mail periódicas con los responsables del trabajo en la empresa.

R3. La base de datos estará alojada en los servidores de la empresa, por lo que un fallo técnico o inesperado en la misma podría retrasar las tareas relacionadas con el paquete de Base de Datos y cualquier implementación software asociada al tratamiento de dichos datos. Esto podría evitarse copiando la base de datos y exportándola para poder seguir trabajando y realizando pruebas en base a esta sin depender de la disponibilidad de los servidores de la empresa, o bien realizando todas las pruebas pertinentes en local y no depender de las conexiones externas.

3

Análisis de requisitos

En este capítulo se detallan los roles de los usuarios dentro de la aplicación y los requisitos que la misma tendrá que cumplir para satisfacer las necesidades de la empresa, tras varias comunicaciones tanto electrónicas como presenciales con los responsables de esta.

Para desempeñar el desarrollo del proyecto con éxito y cumplir además con las expectativas y satisfacer las necesidades de la empresa, la cual hará uso de la aplicación en un entorno laboral real, se ha llevado a cabo un proceso de análisis de requisitos de la aplicación para tener una visión global del producto a realizar.

Dicho proceso ha consistido principalmente en dos partes; siendo la primera una serie de comunicaciones por correo para la primera recogida de datos y la concreción de una fecha para una reunión presencial, y la segunda fase consistía precisamente en llevar a cabo dicha reunión para la conclusión de la captura de requisitos.

En resumen, en este apartado se detallan los requisitos funcionales y no funcionales extrapolados de las diferentes comunicaciones electrónicas y la reunión llevada a cabo con los responsables de la empresa.

3.1. Roles de usuario

El sistema dispondrá de dos tipos de usuarios, claramente diferenciados en cuanto a funcionalidades disponibles y privilegios. Un usuario sin registrar no podrá hacer uso de la aplicación.

3.1.1. Cliente

El usuario estándar. Tiene acceso a algunas funcionalidades con respecto a sus instalaciones y la gestión de sus alarmas y correos.

3.1.2. Administrador

El usuario con privilegios de administración tiene acceso a funcionalidades con respecto a todas las instalaciones y podrá gestionar el envío de las alarmas y los correos. También dispondrá de funcionalidades de gestión de datos: un usuario de este tipo puede añadir y eliminar tanto usuarios como instalaciones de la base de datos.

3.2. Requisitos

Tras varias comunicaciones escritas, mediante correo electrónico, y una reunión presencial con el responsable del proyecto por parte de la empresa, y teniendo en cuenta las tareas a realizar descritas en la propuesta del trabajo de fin de grado, los requisitos principales extraídos son los siguientes:

3.2.1. Requisitos funcionales

- **R1.** La aplicación debe poder comunicarse con las diferentes plataformas de gestión de alumbrado y energía.
- **R2.** La aplicación debe procesar adecuadamente la información recibida de las cabeceras y generar las alarmas o mensajes correspondientes.

- **R3.** La plataforma debe supervisar el funcionamiento de las instalaciones, aunando a través de una única plataforma las diferentes clases de alarmas generadas.
- **R4.** La comunicación con las diversas plataformas se llevará a cabo, siempre que sea posible, a través de los comandos de las APIs ya implementadas.
- **R5.** La aplicación ofrecerá distintas funcionalidades según el tipo de usuario:
    - Administrador.
    - Cliente o usuario común.
- **R6.** El usuario de tipo *administrador* podrá definir qué usuarios tienen acceso a la plataforma web.
    - Gestión de usuarios.
- **R7.** El usuario de tipo *administrador* podrá dar de alta las distintas instalaciones y vincularlas a un cliente.
    - Gestión de instalaciones.
    - Gestión de base de datos.
- **R8.** Solamente aquellos usuarios con privilegios de *administrador* podrán hacer cambios en la base de datos, salvo excepciones (*R14*).
- **R9.** El cliente o usuario común tendrá la posibilidad de configurar las alarmas de las instalaciones que tiene asignadas.
    - Tipos de alarmas que desea o no desea recibir.
    - Hora del día a la que quiere recibir alarmas.
    - Consultar el registro o historial de las alarmas generadas por sus instalaciones.
- **R10.** La aplicación debe almacenar un histórico o registro de las alarmas que se han ido generando.
- **R11.** Cada usuario registrado debe recibir mediante correo, en caso de existir, las alarmas que se generen en sus instalaciones.
- **R12.** Para acceder a la plataforma de la aplicación, se hará uso de un código o nombre de usuario y una contraseña asociada a cada uno.
- **R13.** A cada cliente registrado se le proporcionará una contraseña pseudoaleatoria para el inicio de sesión dentro de la aplicación.
- **R14.** Los clientes dispondrán de una funcionalidad que les permita modificar su contraseña.
- **R15.** Solamente se podrá hacer uso de la aplicación en modo administrador si se accede a esta localmente.
- **R16.** Los tipos de alarmas serán diferentes para cada tipo de instalación, debido a las diferencias tanto en características como en funcionalidades y APIs.
- **R17.** La aplicación tendrá una función de monitorización de cabeceras, que se ejecutará cada cierto tiempo al día, para comprobar el estado de las mismas; con un máximo de tres intentos de conexión por cabecera. En caso de no poder conectarse con alguna o detectar algún problema, se notificará mediante correo al cliente al que pertenece la instalación.
- **R18.** Para las cabeceras de instalaciones de banda estrecha, el monitoreo de su estado será diferente (se realizará únicamente de madrugada), ya que los comandos de la API aumentan mucho el consumo de la misma y demasiadas peticiones en poco tiempo pueden afectar negativamente al funcionamiento y a su integridad.

3.2.2. Requisitos no-funcionales

- **RNF1.** La aplicación debe ser desarrollada utilizando software de uso libre.
- **RNF2.** La plataforma web deberá ser capaz de ejecutarse y funcionar correctamente sobre un sistema operativo LINUX.
- **RNF3.** La aplicación estará alojada en el servidor de la empresa.
- **RNF4.** Las funciones que ofrece la plataforma deben estar disponibles a tiempo completo (24 horas del día, 365 días al año).
- **RNF5.** La información almacenada en la base de datos de la que haga uso la aplicación debe estar protegida o encriptada.
- **RNF6.** Para la paleta de colores del diseño de las interfaces, se ha decido hacer uso de la plantilla de colores que se incluyen en el logo de la empresa, además del negro y blanco, al igual que varias aplicaciones web de las que ya dispone la empresa (*Fig.3.1*).
- **RNF7.** La fuente para el texto de la aplicación será *Futura*, al igual que varias aplicaciones web de las que ya dispone la empresa.
- **RNF8.** No se ha impuesto ninguna restricción o requisito para la creación y diseño de la base de datos de la que el programa hará uso.

Fig.3.1. *Logo de la empresa LEYCOLAN S.A.L. para aplicaciones web.*

En la siguiente figura se puede observar el modelo de casos de uso correspondiente a las funcionalidades a las que tiene acceso tanto cualquier usuario como un usuario con privilegios de administrador.

Fig.3.2. *Modelo de casos de uso correspondiente al análisis de requisitos de la aplicación a desarrollar.*

4

Tecnologías

En este capítulo se describen las tecnologías y herramientas software utilizadas para el desarrollo de la aplicación, comenzando por una breve investigación para determinar cuáles se han seleccionado para el proyecto y por qué, ya que en la propuesta del trabajo no se imponía ninguna restricción referente a las tecnologías a utilizar.

4.1. Estudio de tecnologías

En la propuesta del proyecto, la empresa no puso ninguna restricción en lo que a herramientas de trabajo se refiere, pudiendo ser la aplicación desarrollada mediante cualquier entorno y lenguaje de programación. Una de las ideas principales a la hora de decidir qué tecnologías se van a utilizar ha sido la de aprender sobre nuevas tecnologías y herramientas que más se utilicen hoy en día en los ecosistemas laborales.

Para escoger las herramientas más adecuadas para desarrollar esta aplicación web, se ha llevado a cabo una breve investigación acerca de diferentes lenguajes, tanto para las funcionalidades como para el diseño del sistema, y también comparando diversos frameworks para decidir cuáles serían los más adecuados para este trabajo. Hay que tener en cuenta que el objetivo final de este proyecto es el de diseñar e implementar una aplicación web que más adelante será adoptada para el uso habitual de la empresa.

Primero se detallan varias ventajas y desventajas de algunos lenguajes de programación para las funcionalidades y el diseño de la aplicación, teniendo en cuenta que se utilizarán para un **desarrollo web**. Después se repite el mismo proceso para los marcos de trabajo y entornos de desarrollo, y finalmente se toma la decisión final de las herramientas escogidas, incluyendo algunas conclusiones y otras tecnologías que también son útiles para, por ejemplo, gestionar adecuadamente el control de versiones de la aplicación.

4.1.1. Lenguajes

A continuación se muestran varios de los lenguajes planteados a utilizar para el desarrollo de la aplicación, comenzando con varios aspectos tanto positivos como negativos de aquellos que se usan para implementar las funcionalidades del sistema [2]. Posteriormente se detallan las ventajas y limitaciones de los lenguajes que conformarán el diseño [3].

Java

Características / Ventajas	*Limitaciones / Desventajas*
- Permite desarrollar aplicaciones con una alta escalabilidad. - Facilidad para entender el lenguaje y su sintaxis. - Aunque consuma bastante CPU, soporta infinidad de librerías de software libre. - Opciones avanzadas de seguridad.	- Java se ejecuta mejor en sistemas más potentes, por lo que puede ser un problema a la hora de integrar interoperabilidad al sistema. - Carece de herramientas y objetos modernos en cuanto a desarrollo de GUIs.

Tabla 4.1. *Tabla de ventajas y desventajas del lenguaje Java.*

PHP

Características / Ventajas	*Limitaciones / Desventajas*
- Fácil de usar e integrable en múltiples tipos de aplicaciones. - Lenguaje open-source con gran variedad de librerías disponibles. - Automatización para autenticaciones, mapeo de URLs y gestión de sesiones. - Altamente versátil y con seguridad integrada.	- Cada vez menos utilizado y menos popular en general. - Lenguaje limitado en comparación con tecnologías modernas de desarrollo backend. - Al ser open-source, puede darse el caso de crear código con bugs y mal optimizado.

Tabla 4.2. *Tabla de ventajas y desventajas del lenguaje PHP.*

Python

Características / Ventajas	*Limitaciones / Desventajas*
- Lenguaje muy cercano al "lenguaje humano" y por tanto fácil de entender. - Disponibilidad de múltiples librerías para facilitar las tareas implementadas. - Características IoT. - Eficiente en cuanto a coste.	- El acceso a bases de datos no está tan bien desarrollado como en otros lenguajes. - Requiere de mucho testeo y *debugging*. - Altamente dependiente de software de terceros.

Tabla 4.3. *Tabla de ventajas y desventajas del lenguaje Python.*

JavaScript

Características / Ventajas [4]	*Limitaciones / Desventajas*
- Ofrece rapidez al ejecutarse directamente en el navegador del cliente y no requiere compilación previa. - Lenguaje simple y popular, fácil de usar tanto en frontend como backend (gracias a tecnologías como *Node.js*). - Ofrece alta interoperabilidad y reduce la dependencia con el lado del servidor. - Interfaces avanzadas con funcionalidades extendidas para las webs. - Es actualizado con frecuencia y muy versátil, pudiendo desarrollar una aplicación entera, con acceso a datos, etc puramente en JavaScript	- Como JavaScript se ejecuta en el lado del cliente, el código puede manipularse con fines maliciosos. - Diferentes navegadores podrían interpretar el código de manera diferente, aunque hoy en día estas diferencias son mínimas.

Tabla 4.4. *Tabla de ventajas y desventajas del lenguaje JavaScript.*

HTML

Características / Ventajas [5]	*Limitaciones / Desventajas*
- Fácil de aprender y utilizar gracias a una sintaxis limpia y simple. - Es un estándar de la industria, utilizado por prácticamente todos los navegadores, además de ser independiente de la plataforma. - Capacidad de incluir fácilmente archivos multimedia o enlaces a sitios web tanto internos como externos. - Gran variedad de plantillas de uso libre para no tener que escribir todo el código.	- Necesita de servicios de terceros para garantizar seguridad. - Ideal para necesidades web básicas, pero carece de capacidades avanzadas. - Por defecto las páginas en HTML son estáticas, por lo que hay que echar mano de otros lenguajes o herramientas para crear código dinámico. - Mantener el código puede llegar a ser tedioso si la aplicación o web se conforma de muchas páginas HTML.

Tabla 4.5. *Tabla de ventajas y desventajas del lenguaje HTML.*

CSS

Características / Ventajas [6]	*Limitaciones / Desventajas*
- Ahorro considerable de tiempo al cargar las páginas o al realizar cambios de diseño en la aplicación. - Consistencia en el aspecto de la página. - Ofrece alta compatibilidad y una amplia variedad de diseño para diferentes dispositivos.	- Puede haber problemas de compatibilidad entre los diferentes navegadores. - Vulnerable al tratarse de código escrito en texto plano.

Tabla 4.6. *Tabla de ventajas y desventajas del lenguaje CSS.*

4.1.2. Frameworks

A la hora de implementar código, sea cual sea el lenguaje, también es importante escoger correctamente un marco de trabajo que ayude a agilizar el desarrollo del mismo, y para ello se detallan a continuación diversas ventajas y desventajas de algunos de los frameworks que podrían ser interesantes para realizar este proyecto [7][8].

Los marcos de trabajo barajados son los siguientes:

Node.js

Características / Ventajas [9]	*Limitaciones / Desventajas*
- Está basado en JavaScript, por lo que es fácil de aprender, y útil para desarrollar apps multiplataforma. - Solución viable a la hora de desarrollar aplicaciones escalables. - Open-source y con una comunidad muy activa, respaldada por desarrolladores de JavaScript. - Es eficiente en cuanto a coste y permite escribir código JavaScript tanto para frontend como para backend. - Reutilización de código y múltiples herramientas de uso libre.	- Incapaz de llevar a cabo operaciones computacionalmente pesadas. - API está en constante cambio, por tanto diversas funciones quedan obsoletas. - Herramientas de baja calidad al tratarse de software libre y ciertamente reciente.

Tabla 4.7. *Tabla de ventajas y desventajas de Node.js.*

Angular

Características / Ventajas	*Limitaciones / Desventajas*
- Está soportado por Google, por lo que goza de fiabilidad y constantes actualizaciones. - Gran variedad de integraciones de terceros, dando lugar a múltiples herramientas a disposición del programador. - Diseñado para ser completamente personalizable e implementable a otros proyectos. - La inyección de dependencias es una herramienta muy potente si se sabe utilizar bien. - Centrado más en frontend o en la parte del cliente que en el backend.	- Accesibilidad pobre para rastreadores de motores de búsqueda y opciones SEO limitadas. - Presenta complejidad alta a la hora de aprender a utilizarlo. - Mayor potencial para proyectos grandes que para proyectos medianos o pequeños.

Tabla 4.8. *Tabla de ventajas y desventajas de Angular.*

React

Características / Ventajas [10]	*Limitaciones / Desventajas*
- Herramienta open source y fácil de aprender a utilizar, sobre todo si se tienen conocimientos de JavaScript. - Utiliza DOM virtual, que resulta más eficiente y mejora el rendimiento de la interfaz de usuario. - Tiene componentes reutilizables, ahorrando tiempo y creando código más preciso. - Código más estable y controlable gracias al flujo unidireccional de datos (los cambios en los subelementos no afectan al elemento principal). - Rápido desarrollo con constantes actualizaciones.	- La sintaxis JSX [11] puede ser compleja y difícil de comprender, aunque no afecte al rendimiento de la aplicación. - Al estar constantemente en evolución, los nuevos cambios y características de ReactJS pueden llegar a ser confusas y complicadas de asimilar. - Las actualizaciones constantes ocasionan que haya insuficiente documentación y puede ser un inconveniente para los desarrolladores que hagan uso de esta tecnología.

Tabla 4.9. *Tabla de ventajas y desventajas de React.*

4.1.3 Entornos de desarrollo

Eclipse

Características / Ventajas [12]	*Limitaciones / Desventajas*
- Tiene a su disposición infinidad de plugins para gran variedad de proyectos, además de buenas herramientas de debugging. - Soporta múltiples lenguajes de programación, lenguajes de modelado como UML y es ideal para proyectos grandes. - Experiencia previa al utilizar este entorno.	- Eclipse tiende a ofrecer un rendimiento más lento debido a la cantidad de plugins que puede llegar a tener. - Utiliza mucho espacio en memoria al utilizarse en proyectos de gran tamaño.

Tabla 4.10. *Tabla de ventajas y desventajas del entorno de desarrollo Eclipse.*

Visual Studio Code

Características / Ventajas [13][14]	*Limitaciones / Desventajas*
- Entorno multiplataforma muy accesible, compatible con múltiples lenguajes y alta extensibilidad. - *Intelli-Sense*: detecta código incompleto y las variables se declaran automáticamente. - Proporciona herramientas para tecnologías web (HTML, CSS, JSON…).	- La instalación de los diferentes plugins para que el sistema sea estable y funcione correctamente puede no ser fácil. - Configurar la interfaz puede resultar complejo. - En ocasiones el rendimiento puede ser lento.

Tabla 4.11. *Tabla de ventajas y desventajas del entorno de desarrollo Visual Studio Code.*

4.1.4. Decisión final

Una vez vistas las características de estos lenguajes, marcos de trabajo y entornos de desarrollo, pese a no haber grandes diferencias ni elecciones obvias en cada uno de los apartados, y teniendo en cuenta que la idea de utilizar nuevas tecnologías ha tenido relevancia a la hora de escoger, estas han sido las herramientas seleccionadas para el desarrollo del proyecto:

Lenguajes

Para el diseño y la implementación de interfaces:

- *HTML*
- *CSS*
- *JavaScript*

Para las funcionalidades del sistema:

- *JavaScript*

Frameworks

Teniendo en cuenta las características detalladas, que el lenguaje principal es *JavaScript*, y que algunos de los frameworks más usados actualmente [15] en frontend y backend son **React** y **Node.js** respectivamente, estos son los que se van a utilizar para el desarrollo de la aplicación.

Entorno de desarrollo

Visual Studio Code

- Entorno de desarrollo que, además de ser gratuito, ofrece plugins y extensiones para la integración tanto de *React* como de *Node.js*.

En el siguiente apartado se describen otras tecnologías y herramientas que se han utilizado para el desarrollo final de la aplicación.

Otras tecnologías

APIs de cabeceras: para la integración de algunas funcionalidades de los dos tipos de instalaciones se ha hecho uso de una serie de APIs.

- *API banda ancha*: API web que funciona a través de instrucciones que permiten configurar y gestionar la instalación. Las instrucciones son comandos POST en HTTPS, y los datos gestionados por estos comandos tienen estructura de lenguaje JSON.
- *APIs banda estrecha*: APIs proporcionadas por la empresa *Solar Power*[1] a las cuales se accede mediante http o https, y se utilizan las URLs de los servicios que se quieren utilizar. Existen dos servicios independientes:
    - *APISplit*: permite al usuario final manejar sus instalaciones, bien consultando información en tiempo real, enviando configuraciones o consultando históricos de funcionamiento.

- *GeoLoc*: permite ubicar los nodos de la instalación y permite mandar órdenes a luminarias concretas para comprobar en campo la correcta instalación y funcionamiento de cada punto de luz.

Hooks de React: son una característica relativamente reciente de React [16]. Estos *hooks* permiten usar dinámicamente estados, variables y otras características de React sin necesidad de implementar una clase.
Los hooks utilizados son los siguientes:

- *useState:* devuelve un valor con estado y una función para actualizarlo (*setState*).
- *useEffect:* parte de código que se ejecuta después de cada renderizado completado, aunque también se puede manipular para ejecutarlo solo cuando ciertos valores han sido modificados.
- *useContext:* la manera que se ha utilizado a través de React para simular el comportamiento de variables globales, para la autenticación o la renderización de según qué componentes dependiendo de ciertos factores, por ejemplo.

Algunos ejemplos de cómo se han utilizado estos elementos aparecen más adelante, en el capítulo de implementación.

Paquetes npm: NPM es un administrador de paquetes que ayuda a organizar y compartir paquetes y módulos de Node.js. NPM puede descargar paquetes y buscar actualizaciones de los paquetes que ya han sido instalados. Su funcionamiento sería parecido al de los *imports* de Java, pero en lugar de añadir las librerías, se instalan mediante comandos a través de la consola del terminal.
Algunos de los paquetes más relevantes utilizados han sido:

- *Axios:* utilizado para la comunicación entre la parte del cliente y la API del servidor de la aplicación, y para el intercambio de datos entre la parte del servidor y las APIs externas de las cabeceras.
- *Express:* utilizado para la creación de un servidor Express en el backend, y posee una gran cantidad métodos de programa de utilidad HTTP y middleware a su disposición. Gracias a ello la creación de una API sólida resulta rápida y sencilla.
- *Mysql:* utilizado para las consultas con la base de datos de la aplicación.
- *Bcrypt:* utilizado para el cifrado de contraseñas.
- *Cron:* utilizado para la programación de la ejecución de diversas funcionalidades a horas concretas del día. Esencial a la hora de realizar las pruebas de funcionamiento de la aplicación final.

Base de datos

- *MySQL:* lenguaje utilizado para la base de datos, puesto que es con el que más se ha trabajado a lo largo del grado y resulta sencillo de manejar.
- MySQLWorkbench: aplicación utilizada para la creación y diseño de la base de datos y la puesta en marcha de un servidor SQL para poder trabajar desde la aplicación con los datos almacenados en esta.

---

1. Solar Power es una empresa proveedora de equipos de energía solar ubicada en Madrid, España.

5

Diseño

En este capítulo se describe el diseño que se ha decidido emplear para la base de datos de la aplicación, incluyendo las tablas y las relaciones entre las mismas. También se analiza el diseño general que posee el proyecto de la aplicación.

5.1. Diseño de la base de datos

La base de datos utilizada para el almacenamiento de datos de la aplicación tiene un diseño sencillo pero contiene la suficiente información como para poder desempeñar correctamente las funciones para las que la aplicación ha sido desarrollada. El esquema se ha realizado teniendo en mente la posible adición de nuevos tipos de instalaciones y alarmas no incluidos en esta primera versión, facilitando el proceso en caso de necesidad de incluir alguno de los mencionados.

5.1.1. Diagrama de tablas

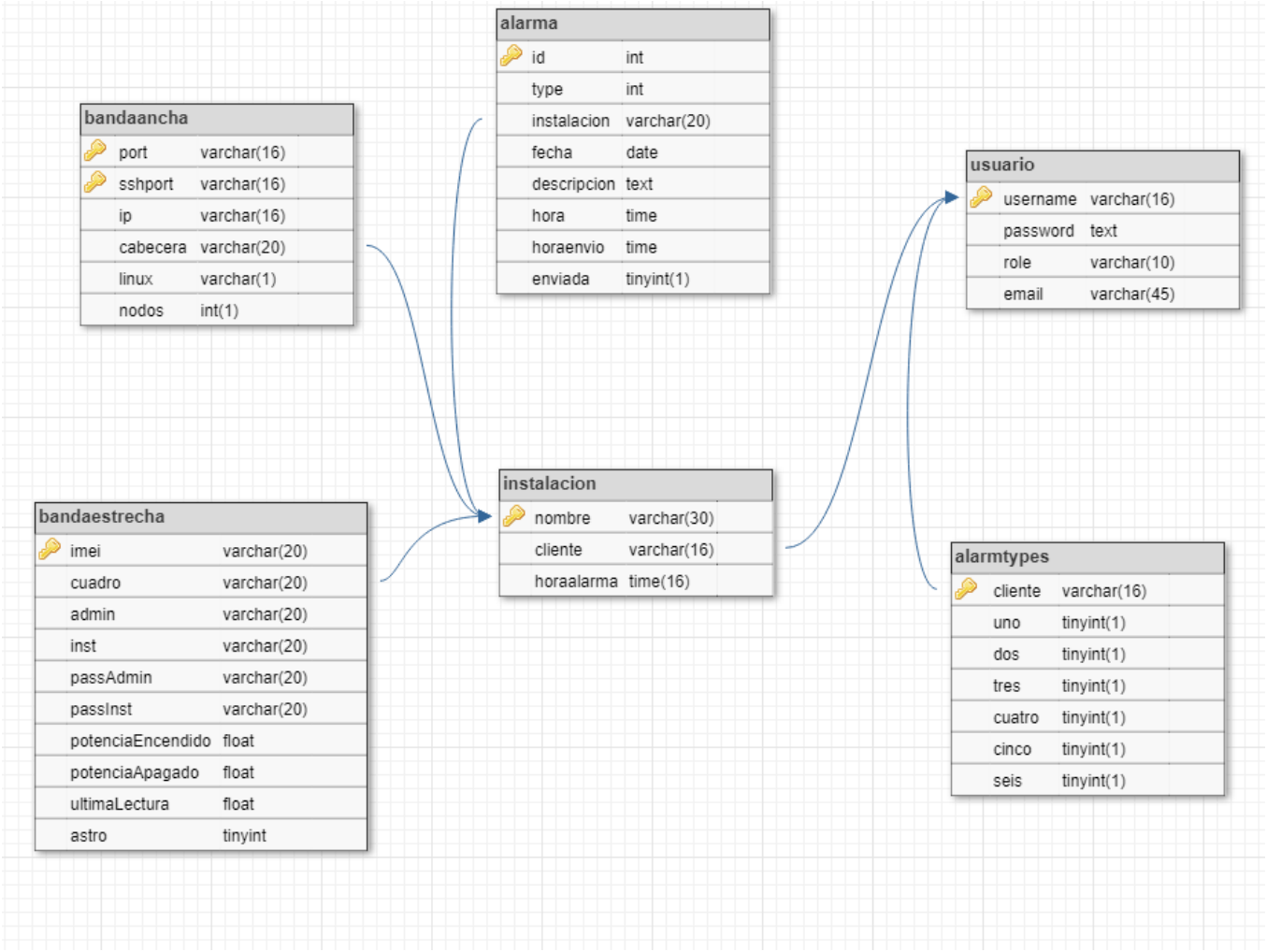


Fig.5.1. *Diagrama de las tablas que componen la base de datos.*

Ver diagrama UML incluido en el anexo A para obtener una visión más clara de las relaciones entre los datos almacenados en la base de datos.

5.1.2. Descripción de clases

Usuario: representa un usuario de la plataforma, el cual accede a la misma con su nombre de usuario y una contraseña. Existen varios roles de usuario, y cada uno posee un email en el que recibirá las alarmas que tenga configuradas.

Atributos:

- *username*: clave primaria. Se utiliza tanto para identificar de manera única a cada cliente como para que este acceda a la plataforma junto con una contraseña.
- *password*: contraseña que utiliza el cliente para acceder a la plataforma, se almacena cifrada por razones de seguridad.
- *role*: rol del usuario, generalmente habrá únicamente dos tipos: 'cliente' y 'admin'. Este último goza de mayor privilegio y funcionalidades añadidas que ningún cliente debería poder utilizar en circunstancias normales.
- *email*: dirección de correo electrónico del usuario a la que se enviarán las alarmas que se puedan generar en cada una de sus instalaciones.

Instalación: representa una instalación de alumbrado. Contiene el nombre de la localización en la que se encuentra, el nombre del cliente responsable de la misma y la hora de envío de las alarmas que esta genere. El resto de datos necesarios para utilizar funcionalidades de un tipo u otro de instalación se encuentra en diversas tablas relacionadas con esta, como *BandaAncha y BandaEstrecha*. Como se ha mencionado anteriormente, se ha decidido seguir este diseño en pos de posibilitar añadir nuevos tipos de instalación en un futuro, cada una con sus propios atributos exclusivos.

Atributos:

- *nombre:* clave primaria. Identificador de cada instalación.
- *cliente:* clave extranjera (Usuario). Nombre del cliente al que pertenece una instalación.
- *horaalarma:* hora a la que se realizará el envío de las alarmas de cada instalación. Este dato es configurable por el cliente mediante la plataforma web.

BandaAncha: representa las instalaciones de banda ancha. Se identifican mediante dos puertos; el puerto del servidor web que contiene la API de comandos de la instalación, y un puerto para la comunicación con el servidor ssh. Utilizan una ip pública para la comunicación externa, y para acceder al servidor ssh es necesario introducir un usuario específico, representado en la base de datos mediante un carácter.

Atributos:

- *port:* clave primaria compuesta. Puerto en el que se encuentra en ejecución el servidor web de la cabecera de la instalación. Necesario para la utilización de los comandos de la API JSON de la misma.

- *sshport:* clave primaria compuesta. Puerto en el que se encuentra el servidor ssh de la cabecera. Necesario para comprobar el estado y la conexión de la instalación.
- *ip:* dirección ip pública de la cabecera de la instalación. Necesaria para el envío de datos mediante la API de comandos de la propia instalación.
- *cabecera:* clave extranjera (Instalación) que representa el nombre de la instalación.
- *linux:* representa mediante un carácter el usuario para la conexión remota con el puerto ssh de la cabecera.
- *nodos:* número de nodos que tiene la instalación conectados. Se utiliza este valor para la comprobación del nivel de urgencia de la alarma que se genere.

BandaEstrecha: representa las instalaciones de banda estrecha, identificadas mediante un número de serie o *IMEI*. Para el uso de los comandos de la API de estas instalaciones se necesitan las claves de instalador o de administrador. Las lecturas de la potencia entregada por estas instalaciones es esencial para el monitoreo de las alarmas de banda estrecha.

Atributos:

- *imei:* clave primaria. Número de serie de la cabecera de la instalación de banda estrecha, utilizado para su identificación y el envío de comandos.
- *cuadro:* clave extranjera (Instalación) que representa el nombre de la instalación.
- *admin:* nombre de usuario de tipo administrador de la instalación, utilizado para ciertos comandos de la API. (Este dato es ajeno al nombre de usuario usado para acceder a la aplicación y solo se utiliza en los comandos de la API de banda estrecha).
- *inst:* nombre de usuario de tipo instalador de la instalación, utilizado para algunos comandos de la API. (Este dato es ajeno al nombre de usuario usado para acceder a la aplicación y solo se utiliza en los comandos de la API de banda estrecha).
- *passAdmin:* contraseña para poder ejecutar ciertos comandos de la API, junto con el nombre de usuario del mismo tipo (*admin*).
- *passInst:* contraseña para poder ejecutar ciertos comandos de la API, junto con el nombre de usuario del mismo tipo (*inst*).
- *potenciaEncendido:* valor de potencia leído (en vatios, W) durante el encendido más reciente de la instalación. Este valor se actualiza cada vez que la aplicación detecte el encendido de la instalación y realice la lectura de potencia.
- *potenciaApagado:* valor de potencia leído (en vatios, W) durante el apagado más reciente de la instalación. Este valor se actualiza cada vez que la aplicación detecte el apagado y realice la lectura de potencia.
- *ultimaLectura:* último valor de potencia leído. Este valor se actualizará cada vez que se realice una lectura de potencia de la instalación desde la aplicación.

- *astro:* valor que representa si la instalación está encendida o no, ya que en remoto no existe otra manera de saber con certeza si una instalación de banda estrecha se encuentra encendida o apagada (0 = apagada, 1 = encendida).

Alarma: representa las distintas alarmas que se generan en las instalaciones. Se identifican mediante un id autoincremental. Se almacena el nivel de urgencia y la descripción de la alarma junto con la instalación que la ha generado y la fecha y hora en las que ha sucedido. Las alarmas se enviarán a los clientes de las instalaciones que las hayan generado, a la hora configurada.

Atributos:

- *id:* clave primaria. Identifica las diferentes alarmas dentro de la base de datos.
- *type:* nivel de urgencia o tipo de la alarmas. Actualmente existen seis niveles de alarmas:
  - 1: menos del 10% de los nodos de la instalación presentan algún error (al menos uno).
  - 2: entre el 10% y el 25% de los nodos de la instalación presentan algún error.
  - 3: más del 25% de los nodos de la instalación presentan algún error.
  - 4: error de conexión con el servidor web o API de la cabecera de banda ancha.
  - 5: error de conexión con la cabecera de la instalación.
  - 6: error de potencia en encendido o apagado de la instalación de banda estrecha.
- *instalación:* clave extranjera (Instalación) que representa la instalación en la que se ha generado la alarma.
- *fecha:* fecha en la que se ha generado la alarma *(aaaa-mm-dd)*.
- *descripción:* texto que describe el problema que ha provocado la alarma. Indica cuántos nodos presentan errores o si hay algún error de conexión o de lectura de potencia.
- *hora:* hora a la que se ha generado la alarma *(hh:mm:ss)*.
- *horaenvio:* hora a la que se ha programado el envío de la alarma *(hh:mm:ss).*
- *enviada:* valor booleano que indica si la alarma ha sido enviada por correo al cliente correspondiente, para evitar reenviar alarmas o enviar alarmas de la misma instalación y mismo tipo múltiples veces (0 = no enviada, 1 = enviada).

AlarmTypes: representa los niveles de alarma que cada usuario configura para recibir. Contiene el nombre del cliente junto con los niveles de alarma de los cuales desea o no desea ser notificado en caso de alarma.
Atributos:

- *cliente*: clave primaria y extranjera (Usuario). Representa al usuario al que pertenece la configuración de niveles de alarma.
- *uno:* valor booleano que representa la habilitación del envío de alarmas de tipo o nivel 1. (0 = deshabilitado, 1 = habilitado).
- *dos:* valor booleano que representa la habilitación del envío de alarmas de tipo o nivel 2. (0 = deshabilitado, 1 = habilitado).
- *tres:* valor booleano que representa la habilitación del envío de alarmas de tipo o nivel 3. (0 = deshabilitado, 1 = habilitado).
- *cuatro:* valor booleano que representa la habilitación del envío de alarmas de tipo o nivel 4. (0 = deshabilitado, 1 = habilitado).
- *cinco:* valor booleano que representa la habilitación del envío de alarmas de tipo o nivel 5. (0 = deshabilitado, 1 = habilitado).
- *seis:* valor booleano que representa la habilitación del envío de alarmas de tipo o nivel 6. (0 = deshabilitado, 1 = habilitado).

5.2. Diagramas de secuencia

Para entender mejor el diseño y funcionamiento de la aplicación, a continuación se muestran diagramas de secuencia de algunas de las funcionalidades, mostrando cómo funciona la aplicación en distintos casos, según necesite acceder solamente a la capa de la base de datos o precise de la comunicación con una API externa.

Inicio de sesión

El usuario accede mediante la aplicación a la página de inicio de sesión e introduce sus datos. Al tratar de acceder, el lado cliente de la aplicación envía el nombre y contraseña introducidos por el usuario al lado del servidor. Se accede a la capa de la base de datos para buscar el nombre de usuario. Después de cotejar los datos, si existe el usuario, el servidor cifra y compara la contraseña introducida por el usuario con la almacenada para ver si coinciden, y en ese caso genera y responde con una variable de sesión y una cookie que permite al usuario acceder a la aplicación, redireccionándolo a la página de inicio (a la de administrador o cliente, según el caso).

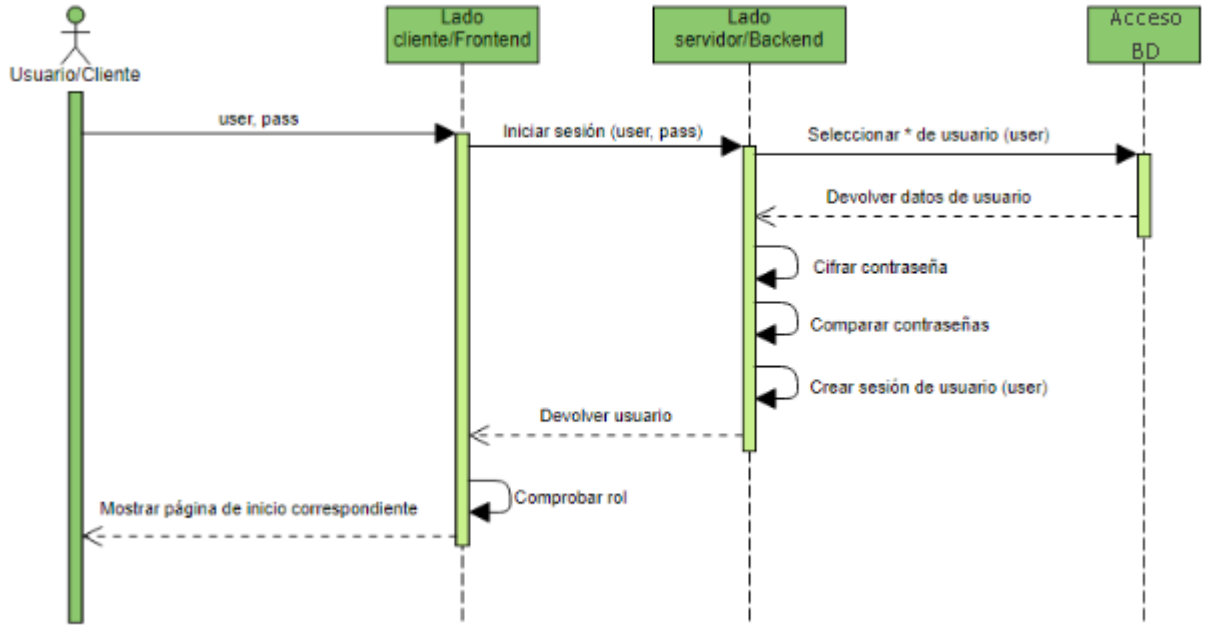


Fig.5.2. *Diagrama de secuencia del caso de uso correspondiente a iniciar sesión.*

Envío de alarmas

El servidor obtiene la hora actual para posteriormente compararla con la hora de envío de un usuario y en caso de coincidir, enviarle un correo con sus alarmas. Obtiene todos los usuarios de la base de datos, y por cada uno de ellos consulta la hora de envío almacenada y realiza la comparación de las horas y los envíos de correo (en caso de coincidir ambas horas).

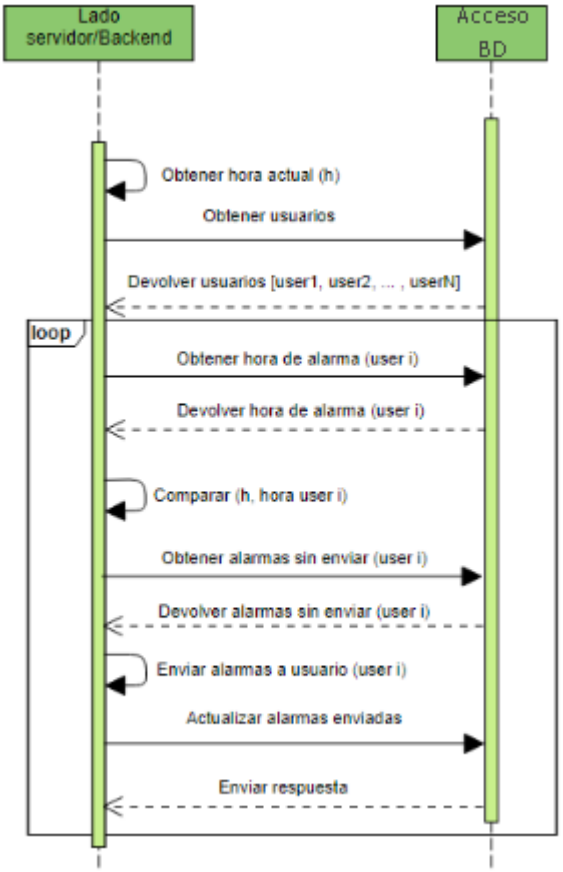


Fig.5.3. *Diagrama de secuencia del caso de uso correspondiente al envío de alarmas por correo.*

Obtención de alarmas de banda ancha

Existe la misma funcionalidad (aunque ligeramente diferente) tanto en el lado del servidor como en el lado del cliente, siendo este último ejecutable por el mismo usuario en la propia interfaz. En este caso se va analizar la que se encuentra en el lado del cliente, ya que accede a más capas de la aplicación.

El usuario ejecuta la función de comprobación de alarmas de una instalación concreta desde la interfaz. La aplicación genera una alerta de confirmación, que si el usuario acepta, se comunica con el lado del servidor con los datos de la instalación seleccionada. Posteriormente se realiza una comunicación servidor-API externa. La API de banda ancha responde, el servidor reenvía la respuesta al lado del cliente y este formatea las alarmas a un formato claramente legible y entendible. Finalmente se muestra por pantalla una alerta de información con las alarmas que existan en la instalación seleccionada.

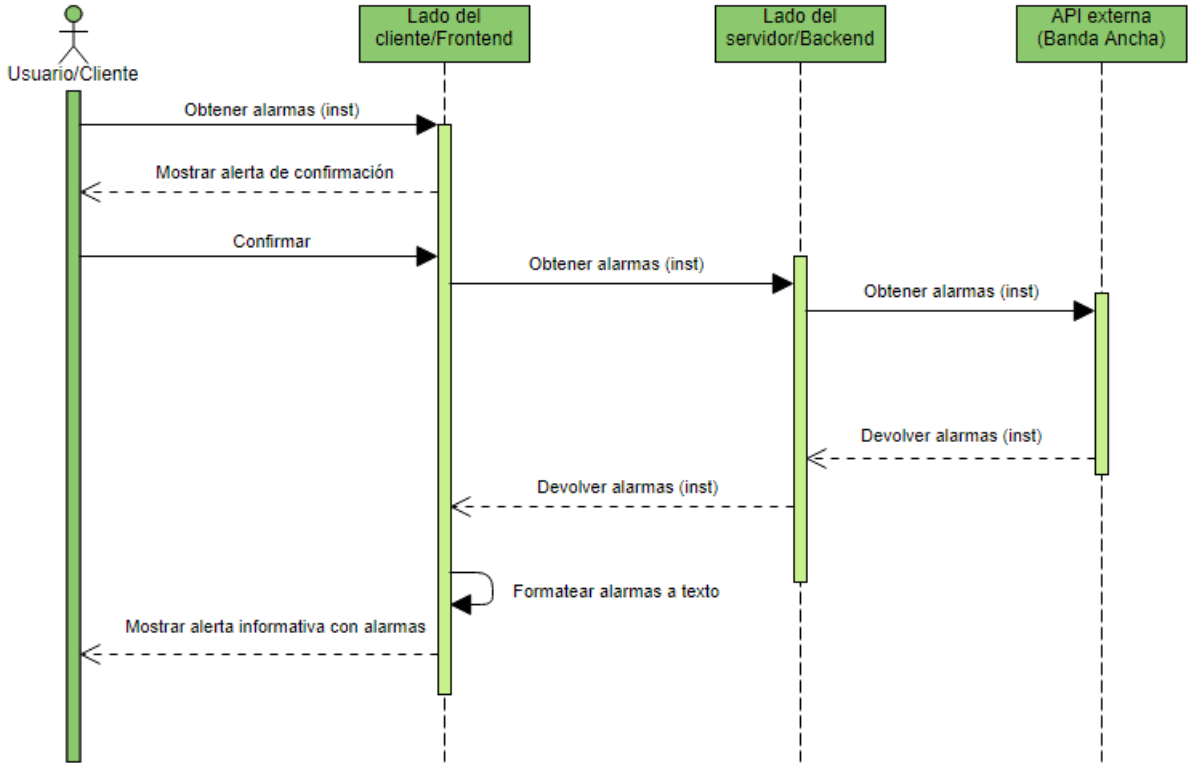


Fig.5.4. *Diagrama de secuencia del caso de uso correspondiente a la obtención de alarmas (banda ancha).*

En cuanto al resto de funcionalidades y casos de uso de la aplicación, la gran mayoría siguen alguno de los esquemas detallados en este apartado, por lo que no es necesario analizar todos y cada uno de los diagramas ya que todos serían muy similares.

En el anexo B se encuentra una imagen que describe la arquitectura de la aplicación de manera gráfica.

6

Implementación

En este capítulo se describe la arquitectura del proyecto y se explica cada una de las partes en las que está dividida la aplicación, haciendo énfasis en aquellas partes de código que resulten esenciales o interesantes para la ejecución del programa.

6.1. Arquitectura del proyecto

El proyecto está principalmente dividido en tres partes o capas: el lado del cliente, el lado del servidor y la base de datos, cuyo diseño y características se han repasado en el capítulo anterior. Aunque se podría considerar que también posee una cuarta capa si se incluyen las APIs externas de las instalaciones como parte de la propia aplicación.

La estructura inicial del proyecto se genera al crear la aplicación de React mediante el comando de consola *'npm create-react-app'*.

La organización de ficheros y directorios final, tras trabajar y desarrollar la aplicación, es la que se muestra a continuación (los nombres han sido modificados para facilitar el entendimiento de la estructura o de los contenidos de los directorios):

- /root: directorio principal del proyecto.
    - /cliente: directorio que contiene los archivos que conforman el lado del cliente.
        - /node_modules: generada automáticamente, contiene las dependencias de los paquetes npm utilizados.
        - /public: generada automáticamente, contiene el logo de la aplicación y el fichero *index.html* sobre el que se renderizan las páginas .js.
        - /src: directorio que contiene las páginas y los componentes.
            - /components: directorio que contiene los componentes utilizados en las distintas páginas.
            - /pages: directorio que contiene las páginas .js que componen las interfaces o ventanas de la aplicación que el usuario final ve.
            - /image: directorio con imágenes e iconos.
            - App.js: archivo principal de la aplicación que contiene y con el cual se renderizan todas las páginas y componentes.
        - package.json: este archivo es el corazón de cualquier proyecto Node. Registra metadatos importantes sobre un proyecto que son necesarios antes de publicar en NPM, y también define atributos funcionales de un proyecto que npm utiliza para instalar dependencias, ejecutar scripts e identificar el punto de entrada a nuestro paquete.
    - /servidor: directorio que contiene los archivos que conforman el lado del servidor.
        - /node_modules.
        - index.js: archivo que contiene todo el código y las funciones del servidor. Se ejecuta en un puerto diferente al del lado del cliente y debe estar activo para que la aplicación pueda funcionar.
        - package.json.

Existen algunos directorios y archivos que se generan automáticamente, y otros archivos de estilo (.css) que no se muestran en la estructura anterior ya que no poseen tanta relevancia para entender el funcionamiento o estructuración del programa.

6.2. Implementación

6.2.1. Cliente

La parte del cliente o *frontend* es con la que el usuario final interactúa y la que se comunica con el lado del servidor para el intercambio de datos y la ejecución de diversas funcionalidades. Se accede a esta mediante el navegador y está dividida en varias páginas. Dichas páginas consisten en ficheros .js que renderizan dinámicamente código HTML sobre un fichero HTML base llamado *index.html*.

El funcionamiento de React es un tanto peculiar en este aspecto, ya que hace uso de una función llamada *render* para, en lugar de mostrar el código del fichero *index.html*, se muestre el código HTML renderizado por los archivos .js.

Dos de los directorios más importantes son el directorio /pages y la carpeta /components, que contienen las páginas mencionadas y algunos componentes utilizados para las interfaces, respectivamente.

Páginas

App.js

No es una página como tal, ni se accede a este fichero directamente mediante el navegador, pero resulta esencial para la ejecución de la aplicación, ya que contiene todas las rutas a las diferentes páginas, así como los componentes necesarios para el correcto funcionamiento del sistema. El renderizado llevado a cabo en todas las páginas .js es similar, y solamente cambia la estructura y los elementos HTML, en función de la página que se esté mostrando en cada momento.

```
export default function App() {
  return (
  <Auth>
    <Headers>
      <div>
      <Router>
        <Layout>
            <Routes>
              <Route exact path="/" element ={<LoginUi />}/>
              <Route exact path="/home" element ={<Home />}/>
              <Route exact path="/admin" element ={<AdminHome />}/>
              <Route exact path="/instalaciones" element ={<Instalaciones />}/>
              <Route exact path="/instalacionesAdmin" element ={<InstalacionesAdmin />}/>
              <Route exact path="/gestion" element ={<Gestion />}/>
              <Route exact path="/alarmas" element ={<Alarms />}/>
              <Route exact path="/alarmasAdmin" element ={<AlarmsAdmin />}/>
              <Route exact path="/perfil" element = {<Perfil/>}/>
              <Route exact path="/perfilAdmin" element = {<PerfilAdmin/>}/>
            </Routes>
        </Layout>
      </Router>
      </div>
    </Headers>
  </Auth>
  );
}
```

Fig.6.1. *Extracto del código que se renderiza en App.js, con las rutas a las páginas de la aplicación.*

Para navegar entre las diferentes páginas de la aplicación, se utilizan las rutas (*routes*) del paquete npm *react-router-dom*. Todos los paquetes npm que se utilizan en las distintas partes del programa se instalan mediante el comando de consola *npm install {nombre_paquete}*.

Para poder utilizar tanto las funcionalidades de los paquetes, como los componentes de nuestro proyecto, es necesario importarlos en el archivo o página que los utilice (similar a como se importaría una librería en otros lenguajes como Java).

```
import React from "react";
import {BrowserRouter as Router, Routes, Route, } from "react-router-dom";
import { Layout } from "./frontend/components/Layout";

import LoginUi from "./frontend/pages/LoginUi";
import Home from "./frontend/pages/Home";
import AdminHome from "./frontend/pages/AdminHome";
import Gestion from "./frontend/pages/Gestion";
import Instalaciones from "./frontend/pages/Instalaciones";
import InstalacionesAdmin from "./frontend/pages/InstalacionesAdmin";
import Alarms from "./frontend/pages/Alarms";
import AlarmsAdmin from "./frontend/pages/AlarmsAdmin";
import Perfil from "./frontend/pages/Perfil";
import PerfilAdmin from "./frontend/pages/PerfilAdmin";

import Headers from "./frontend/components/Headers";
import Auth from "./frontend/components/Auth";
```

Fig.6.2. *Extracto de código de App.js. Muestra como se importan diferentes elementos.*

index.js

Es el fichero que se ejecuta al lanzar la aplicación, y renderiza el archivo App.js. Gracias a ello se puede interactuar con las diferentes interfaces que este contiene.

```
ReactDOM.render(
    <App />,
  document.getElementById('root')
);
```

Fig.6.3. *Extracto de código de index.js. Muestra como se renderiza el elemento principal, App.js.*

LoginUi.js

Página inicial que se muestra al usuario cuando accede a la aplicación y no ha iniciado sesión. Contiene dos campos de texto para introducir tanto el nombre de usuario como su contraseña, y dos botones; uno para entrar y otro para limpiar ambos campos.

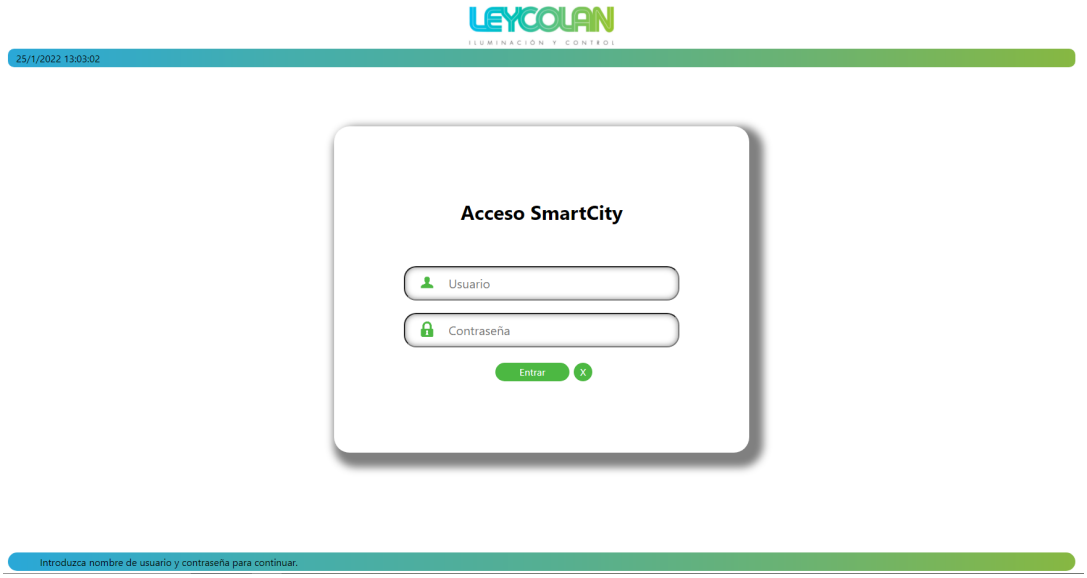


Fig.6.4. *Página de inicio de sesión.*

Si los datos introducidos son erróneos, o falta algún campo por rellenar, aparece un pequeño mensaje en color rojo indicando esto.

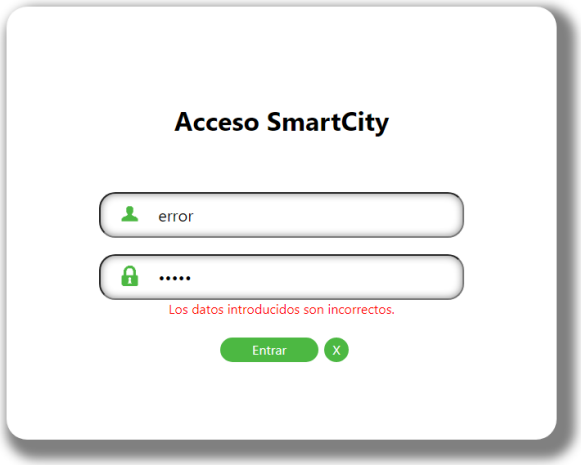


Fig.6.5. *Mensaje de error tras introducir datos incorrectos.*

Fig.6.6. *Mensaje de error por dejar campos sin rellenar.*

Home.js y AdminHome.js

Es la primera página que se muestra tras iniciar sesión correctamente. Muestra un cuadro de texto explicativo que detalla brevemente al usuario las posibilidades que ofrece la aplicación y un *slider* o carrusel con accesos a las diferentes páginas.

Fig.6.7. *Pantalla de inicio del cliente (sin mostrar el slider completo).*

La única diferencia entre acceder a esta página como cliente y como administrador es un acceso adicional en el caso de este último, a la página de gestión de datos.

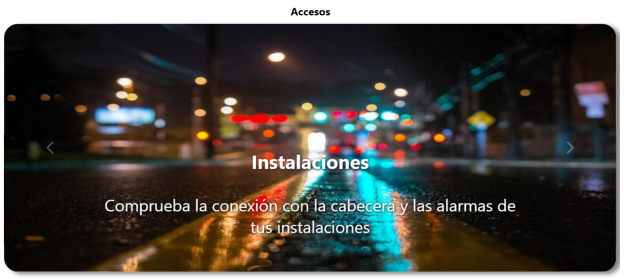


Fig.6.8. *Acceso a Instalaciones del slider.*

Fig.6.9. *Acceso a Alarmas del slider.*

Fig.6.10. *Acceso a Gestión del slider de administrador.*

Instalaciones.js e InstalacionesAdmin.js

Aquí se muestran todas las instalaciones que pertenecen al usuario que ha iniciado sesión. También ofrece la posibilidad de comprobar la conexión y las alarmas en instalaciones de banda ancha, y leer la potencia entregada (en vatios) y las alarmas en instalaciones de banda estrecha. La diferencia entre clientes y administradores en este caso es que para los administradores aparecerán todas las instalaciones de la base de datos, mientras que los clientes solo tendrán accesos a las suyas.

Fig.6.11. *Página de instalaciones del usuario "Legorreta".*

En caso de no disponer de instalaciones de alguno de los tipos, esta parte de la página aparecerá como deshabilitada, ya que no tendría sentido habilitar los botones con las funciones anteriores si no existen instalaciones sobre las cuales ejecutarlas.

Fig.6.12. *Funcionalidades de instalaciones deshabilitadas.*

Cada vez que un usuario trate de ejecutar cualquier función, se le mostrará una alerta de confirmación que puede aceptar o cancelar. Tras aceptar y finalizar la ejecución de la

funcionalidad se le mostrará una alerta informativa con el resultado de la operación, tal y como se muestra en las siguientes capturas. Para generar estas alertas se ha utilizado el paquete npm *sweetalerts2* en lugar de utilizar las alertas por defecto del navegador.

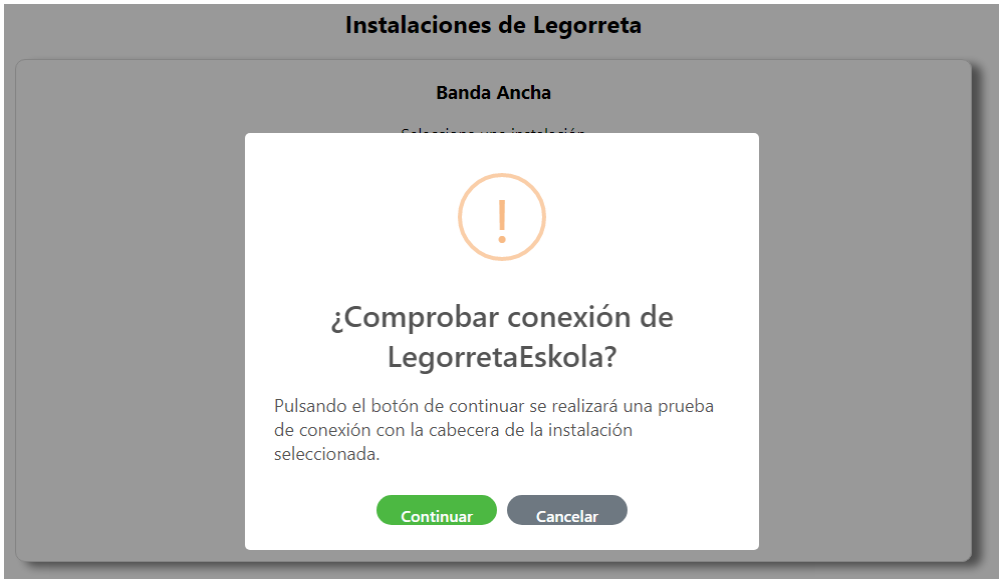


Fig.6.13. *Alerta de confirmación de una operación de prueba de conexión.*

Para la implementación de muchas de las funciones presentes en la aplicación, dadas las características de javascript, en lugar de declarar una función como se haría habitualmente, se permite declarar una variable a la que se le pasa como valor el código de la propia función, al igual que se muestra en la siguiente figura.

```
const confirmPing = () =>{
  swal.fire({
      title: '¿Comprobar conexión de '+inst+'?',
      text: 'Pulsando el botón de continuar se realizará
      icon: 'warning',
      showCancelButton: true,
      focusConfirm: false,
      confirmButtonText: 'Continuar',
      cancelButtonText: 'Cancelar',
  }).then(res=>{
      if(res.isConfirmed){
          document.body.style.cursor='wait';
          ping();
      }
  });
```

Fig.6.14. *Extracto de código que se ejecuta para mostrar la alerta de la figura 6.13.*

El icono mostrado en las alertas cambia en función de si la funcionalidad ejecutada ha tenido éxito, ha habido algún error de conexión o simplemente informa al usuario.

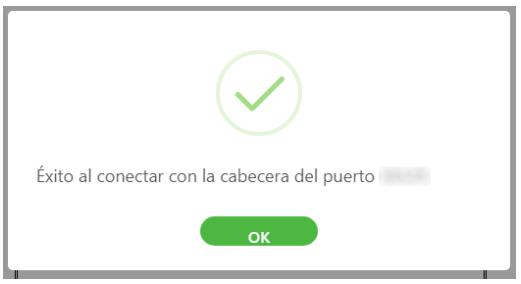


Fig.6.15. *Alerta informativa al conectar con éxito con una cabecera.*

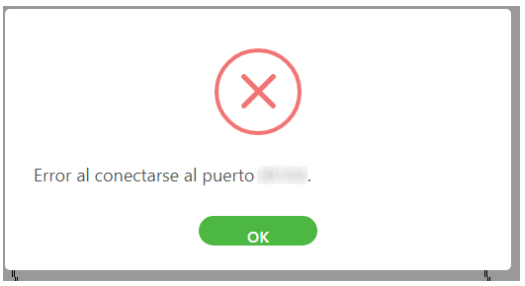


Fig.6.16. *Alerta informativa de error de conexión con una cabecera.*

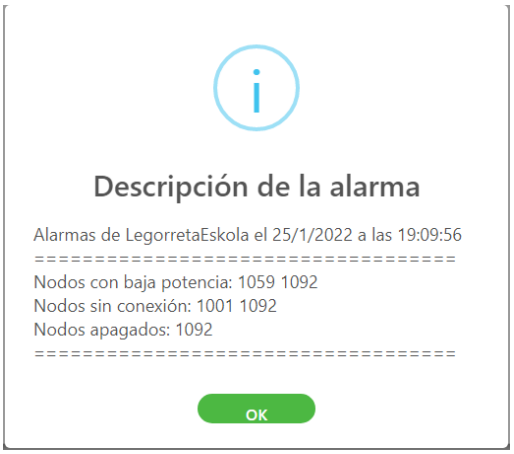


Fig.6.17. *Alerta informativa mostrando las alarmas de una instalación de banda ancha.*

Fig.6.18. *Alerta informativa mostrando la potencia entregada por una instalación de banda estrecha.*

Fig.6.19. *Alerta informativa mostrando las alarmas de una instalación de banda estrecha.*

Para mostrar las instalaciones y ejecutar la función con la instalación seleccionada correspondiente, se hace uso de los hooks de React *useState, useEffect* y *useContext*. Estos elementos se utilizan en prácticamente todas las páginas de la aplicación, aunque de manera ligeramente diferente teniendo en cuenta las necesidades y los datos utilizados en cada una de ellas.

UseState se utiliza para almacenar la información de la instalación seleccionada en la lista desplegable mediante el uso de estados, para poder actualizar la información que se muestra y las variables que posteriormente se utilizan para las diferentes funcionalidades.

```
const [instBE, setInstBE]=useState('');
const [imei, setImei]=useState('');
const [admin, setAdmin]=useState('');
const [adminPass, setAdminPass]=useState('');
const [instalador, setInstalador]=useState('');
const [instPass, setInstPass]=useState('');
const [instalacionesBE, setInstalacionesBE] = useState('');
```

Fig.6.20. *Extracto de código que muestra la declaración de los estados en React.*

En el siguiente extracto de código se muestra un ejemplo de cómo se puede modificar el valor de los estados de React, en este caso haciendo uso de los cambios en el valor seleccionado de la lista desplegable de instalaciones.

```
<p><select disabled={instalaciones.length===0}  onChange={(e)=>{
    setInst(instalaciones[e.target.selectedIndex].cabecera);
    setPort(instalaciones[e.target.selectedIndex].port);
    setSsh(instalaciones[e.target.selectedIndex].sshport);
    setSshuser(instalaciones[e.target.selectedIndex].linux);
}}>
```

Fig.6.21. *Extracto de código que muestra cómo se modifican algunos estados mostrados en Fig.6.20.*

UseEffect se utiliza para obtener todas las instalaciones del usuario que está utilizando la aplicación una vez se carga la página.

```
useEffect(()=>{
    if(logged!==""){
        Axios.post("http://localhost:3001/api/getInst",{
            username: logged
        }).then((response)=>{
            if(response.data.length>0){
            setInstalaciones(response.data);
            setInst(response.data[0].nombre);
            setPort(response.data[0].port);
            setSsh(response.data[0].sshport);
            setSshuser(response.data[0].linux);
        }
        });
    }
  }, [logged]);
```

Fig.6.22. *Extracto de código que muestra un useEffect que modifica los useStates y hace uso de una función de la api del lado del servidor.*

UseContext se utiliza para obtener información del usuario logueado a través de uno de los componentes, emulando el uso de variables globales que usan otros lenguajes.

```
const [logged, setLogged] = useContext(User);
```

Fig.6.23. *Extracto de código en el que se declara una variable a partir de un componente Context.*

Para la ejecución de muchas de las funcionalidades de la aplicación, el lado del cliente hace uso del paquete npm *Axios*, enviando una petición GET, POST… al url de la api del lado del servidor en la que se encuentra dicha función (ver fig.6.22).

Alarms.js y AlarmsAdmin.js

Página que contiene un calendario (paquete npm *react-calendar*) para cada tipo de instalación. Seleccionando una fecha en cada uno de estos calendarios muestra, para la fecha marcada, las alarmas existentes de dicho día en caso de haberse generado. Se puede verla descripción de la alarmas pulsando el botón correspondiente, y se muestra una alerta informativa, similar a las ya descritas (ver fig.6.17).

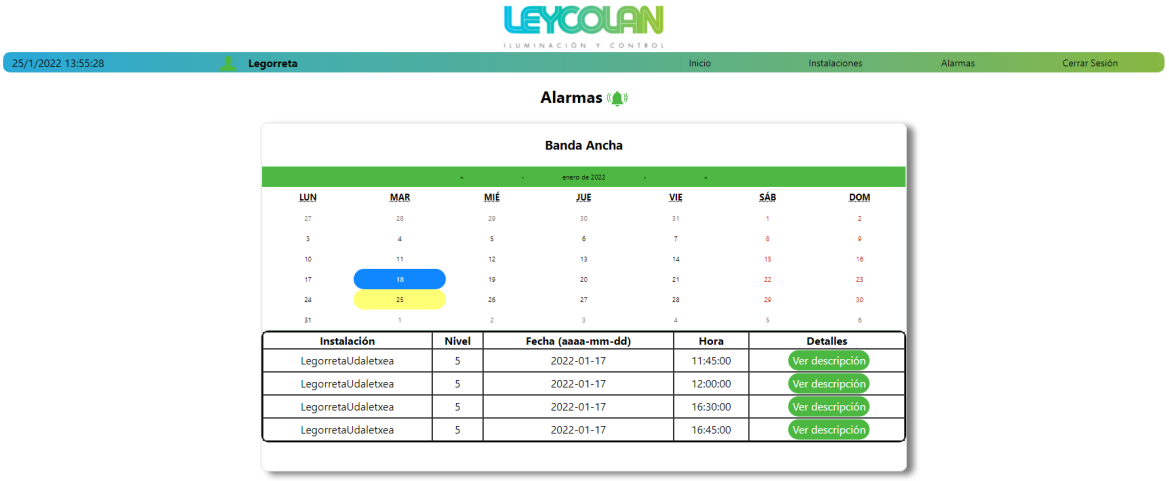


Fig.6.24. *Página de alarmas.*

Similarmente a la página de las instalaciones, el administrador tendrá acceso a todas las alarmas y el cliente solamente a las correspondientes a sus instalaciones, y aparecerá un mensaje indicando que no hay alarmas si se selecciona un día para el cual no exista registro de ellas en la base de datos.

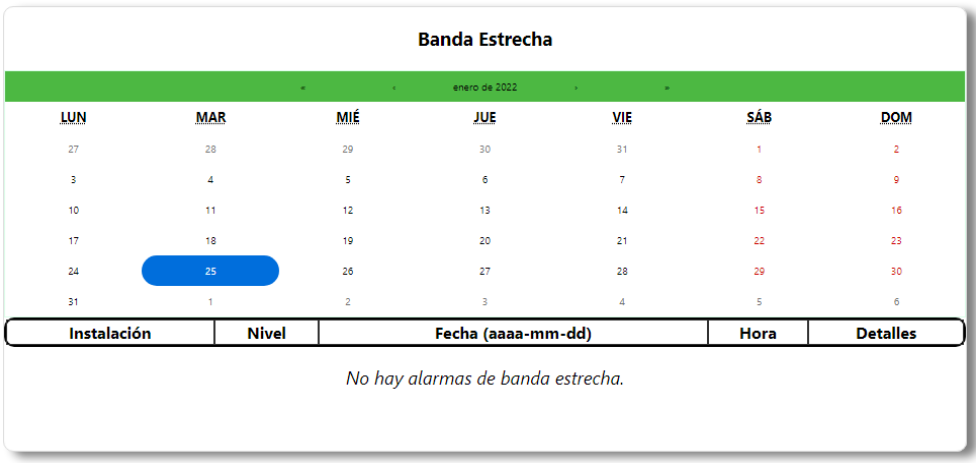


Fig.6.25. *Mensaje mostrado al no encontrar alarmas en una fecha concreta.*

El funcionamiento de los hooks y funcionalidades presentes en esta página es similar a las detalladas en apartados anteriores.

Perfil.js y PerfilAdmin.js

Página que muestra la información del usuario y posibilita la configuración de los niveles de alarmas que desea recibir y la hora a la que quiere que se le envíen los correos con dichas alarmas. Cuando se acceda a esta página, los *checkboxes* de los niveles para los que el cliente ha deshabilitado el envío aparecerán sin seleccionar, el resto estarán seleccionados.

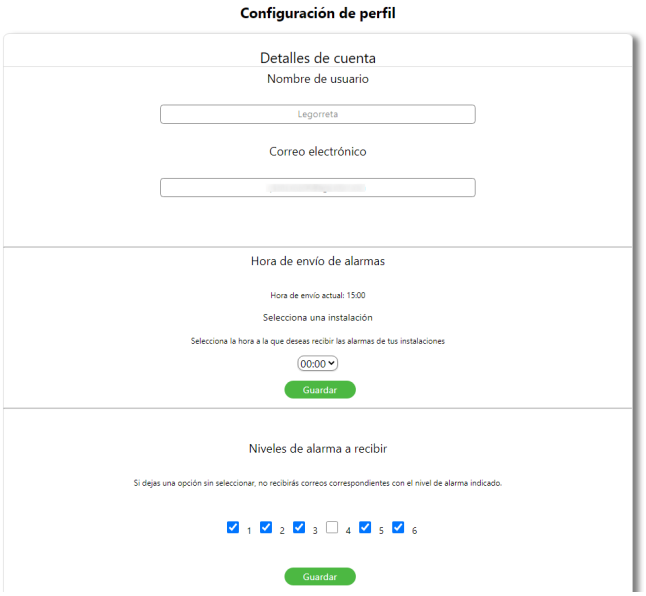


Fig.6.26. *Detalles de cliente y algunas funciones de la página de perfil.*

También ofrece al usuario la funcionalidad de cambio de contraseña mediante un formulario en el que debe introducir la contraseña actual, la nueva y repetir esta última en un tercer campo. Al igual que en el inicio de sesión, todas las funcionalidades muestran un breve mensaje indicando si ha habido error o no (ver fig.6.5).

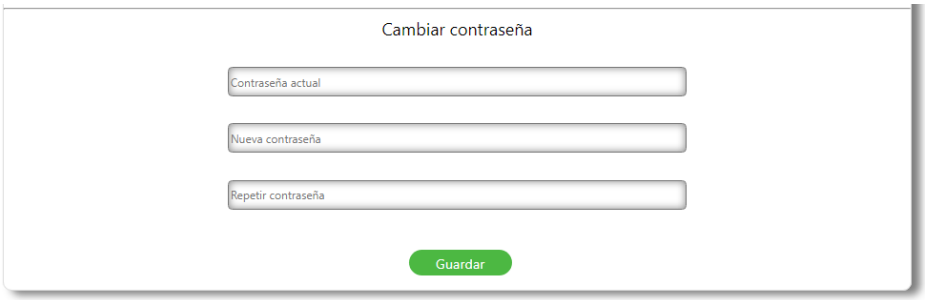


Fig.6.27. *Formulario de cambio de contraseña.*

El administrador no podrá configurar la hora de envío de las alarmas, ya que el envío de correos para clientes y administradores está gestionado de diferente manera.

El funcionamiento de los hooks y funcionalidades presentes en esta página es similar a las detalladas en apartados anteriores.

Gestion.js

Página de uso exclusivo para usuarios administradores. A través de esta se pueden insertar y eliminar tanto usuarios como instalaciones en la base de datos.

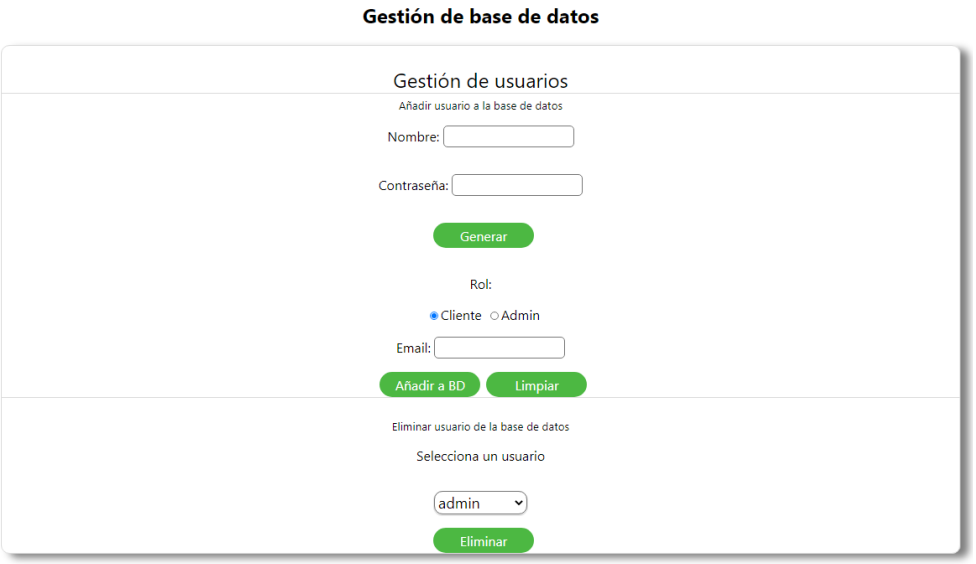


Fig.6.28. *Página de gestión, gestión de usuarios.*

Los elementos mostrados para la inserción de una instalación varían en función de la opción elegida.

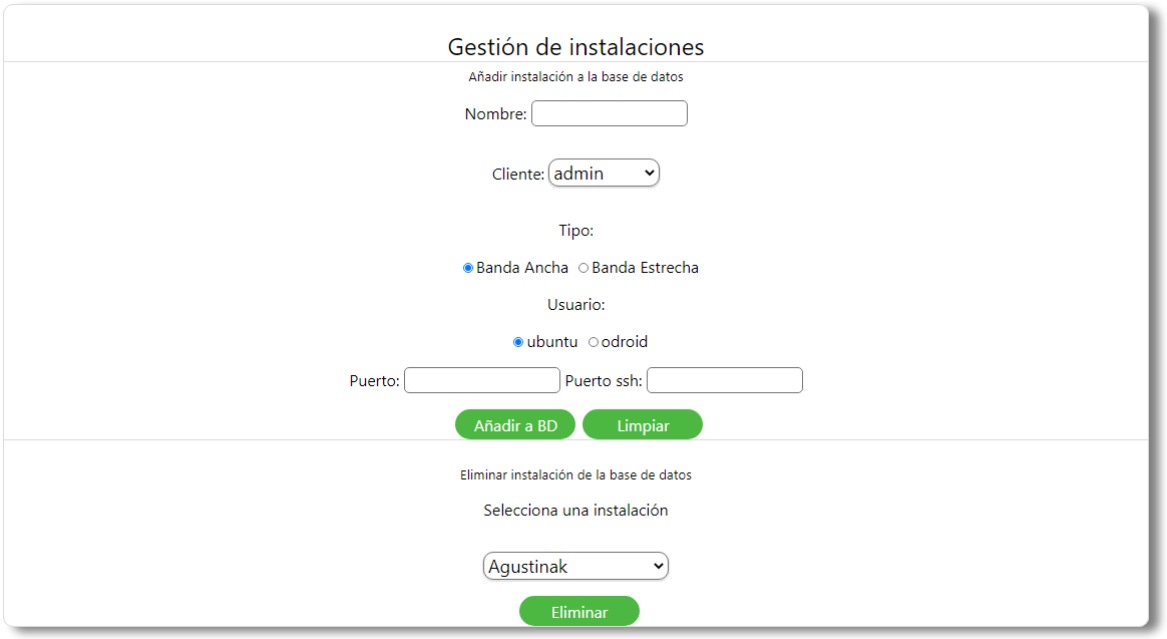


Fig.6.29. *Página de gestión, gestión de instalaciones (banda ancha).*

Fig.6.30. *Página de gestión, gestión de instalaciones (banda estrecha).*

Esto se consigue renderizando unos elementos u otros en función de la opción seleccionada, junto con los useStates de React:

```
{tipo==="ancha" ?
<div>
<p>Usuario:</p>
<div>
<input className="radioBtn" type="radio" id="u" value="u" name="usuario" checked={user === "u"} onChange={(e)=>setUser("u")}/>ubuntu
<input className="radioBtn" type="radio" value="o" name="usuario" checked={user === "o"} onChange={(e)=>setUser("o")}/>odroid
</div>
<p>
  Puerto: <input className="input" type="text" id="puerto" name="puerto" onChange={(e)=>setPort(e.target.value)}/>
   Puerto ssh: <input className="input" type="text" id="ssh" name="ssh" onChange={(e)=>setSshPort(e.target.value)}/>
</p>
</div> : null}
```

Fig.6.31. *Extracto de código que se renderiza cuando el valor del useState 'tipo' es uno concreto.*

Al igual que en las demás páginas, toda función que se ejecute muestra un pequeño mensaje de feedback para informar al administrador del resultado de las inserciones y eliminaciones realizadas en la base de datos (ver fig.6.5).

Al insertar un usuario en la base de datos, envía un correo al email añadido indicando el nombre de usuario y contraseña. Posee también una funcionalidad que utiliza un componente que genera un cadena de caracteres pseudoaleatoria para la contraseña.

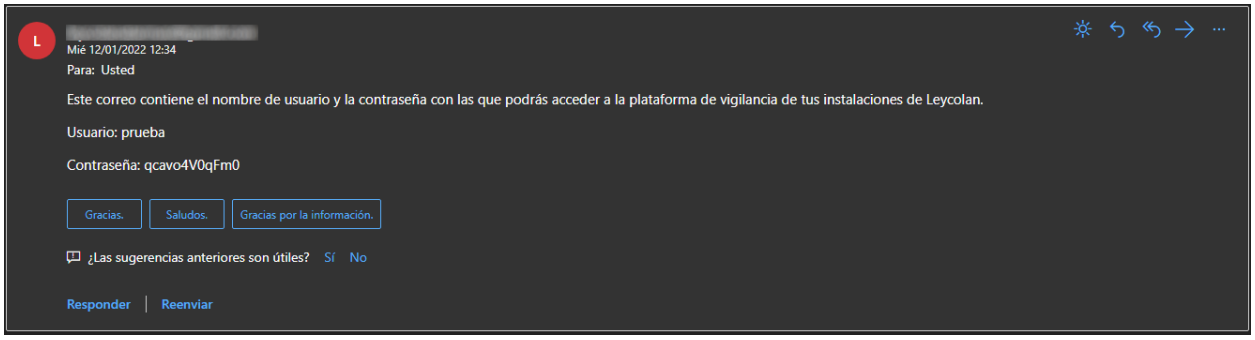


Fig.6.32. *Correo enviado a un usuario tras su registro en la base de datos.*

El funcionamiento de los hooks y funcionalidades presentes en esta página es similar a las detalladas en apartados anteriores.

Componentes

Además de las páginas detalladas en el apartado anterior, existe una serie de componentes, que también consisten en diversos ficheros .js, que enriquecen aspectos de la plataforma. Los componentes implementados son los siguientes:

Layout.js

Componente que contiene el encabezado y el pie de página de la interfaz de usuario. Contiene accesos a las distintas páginas de la aplicación, así como la opción de cerrar sesión y un componente reloj indicando la fecha y hora actuales. Las páginas de los accesos del encabezado se renderizan gracias al componente *link* del paquete *router-react-dom* y dichos accesos cambian dependiendo del estado del usuario: sin iniciar sesión, cliente o administrador. El siguiente fragmento de código muestra el código que renderiza el layout para un administrador, con enlaces (*link*) a las páginas. El elemento *props.children* que aparece sirve para, una vez renderizado el layout en la aplicación (ver fig.6.1), renderizar las páginas correspondientes entre el encabezado y el footer.

```
{state.adminHeader===true?
  <div>
    <div className="logoDiv"><img className="headerLogo" alt="Leycolan_logo" src={logo}/></div>
    <div className="header">
       {<Clock/>}
        <Link to="/perfilAdmin">
          <button id="btn" className="userButton" onClick={scrollUp}>
            <img  className="userIcon" src={userIcon} alt="user"/>
            <label  className="btnText">{logged}</label>
          </button>
        </Link>
        <Link to="/alarmasAdmin">
          <button id="btn" className="hButton2" onClick={scrollUp}>
            Alarmas
          </button>
        </Link>
        <Link to="/instalacionesAdmin">
          <button id="btn" className="hButton3" onClick={scrollUp}>
            Instalaciones
          </button>
        </Link>
        <Link to="/admin">
          <button id="btn" className="hButton4" onClick={scrollUp}>
            Inicio
          </button>
        </Link>
        <Link to="/gestion">
          <button id="btn" className="hButton1" onClick={scrollUp}>
            Gestión
          </button>
        </Link>
        <button id="btn" className="hButton" onClick={logout}>Cerrar Sesión</button>
    </div>

    {props.children}
    <div className="footer"><p className="footerText">ADMINISTRADOR</p></div>
  </div>
: null}
```

Fig.6.33. *Extracto de código que muestra como se renderiza el layout de un administrador.*

Fig.6.34. *Encabezados dependiendo del estado de la sesión de usuario: sin autenticar (arriba), cliente (medio) y administrador (abajo).*

Al cerrar sesión, se accede a la parte del servidor que contiene esta función para destruir la sesión, se modifica el estado de la cabecera (*headers*) y se redirecciona al usuario a la página de inicio de sesión.

Como la cabecera es el único elemento de todo el programa que se encuentra presente en todas las páginas, contiene un *useEffect* importante: controla el estado de la sesión de usuario, redireccionando a este mismo a la página de inicio si no esta logueado; este *useEffect* también evita que un cliente acceda a las funcionalidades de administración y viceversa. Además comprueba el estado del servidor, y en caso de que no esté operativo, muestra una alerta y cierra la sesión.

Fig.6.35. *Extracto de código que muestra el useEffect descrito.*

Clock.js

Componente que indica la fecha y hora actuales, incrustado en el encabezado de la aplicación.

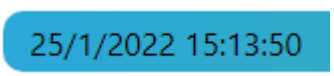


Fig.6.36. *Componente del reloj en el encabezado.*

RandomPass.js

Componente utilizado para generar una cadena de caracteres pseudoaleatoria para la contraseña de usuario.

```
export function random(length) {
    var result           = '';
    var characters       = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789@_';
    var charactersLength = characters.length;
    for ( var i = 0; i < length; i++ ) {
      result += characters.charAt(Math.floor(Math.random() *
 charactersLength));
   }
   return result;
}
```

Fig.6.37. *Extracto de código de la función random().*

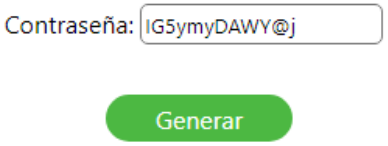


Fig.6.38. *Ejecución de la función en la página de gestión.*

Slider.js y SliderAdmin.js

Componente que implementa el carrusel con los accesos a las páginas en la página de inicio tanto de cliente como de administrador. Como ya se ha explicado, la diferencia entre ambos componentes es la existencia de una acceso adicional para los administradores a la página de gestión.

```
function Slider() {
  return (
    <div style={{ display: 'inline', width: 700, padding: 30 }}>
      <h2>Accesos</h2>
      <Carousel>
        <Carousel.Item interval={3000}>
            <Link to="/instalaciones">
                <img className="d-block w-100" src={inst} alt="Ir a instalaciones" onClick={scrollUp}/>
            </Link>
            <Carousel.Caption>
            <Link to="/instalaciones">
                <h3>Instalaciones</h3>
                <p>Comprueba la conexión con la cabecera y las alarmas de tus instalaciones</p>
            </Link>
            </Carousel.Caption>
        </Carousel.Item>
            <Carousel.Item interval={3000}>
                <Link to="/alarmas">
                    <img className="d-block w-100" src={alarmas} alt="Ir a alarmas" onClick={scrollUp}/>
                </Link>
            <Carousel.Caption>
            <Link to="/alarmas">
                <h3>Alarmas</h3>
                <p>Gestiona las alarmas de tus instalaciones</p>
            </Link>
            </Carousel.Caption>

            </Carousel.Item>
      </Carousel>
    </div>
  );
```

Fig.6.39. *Extracto de código que renderiza el slider.*

Auth.js y Headers.js

Componentes de contexto que emulan variables globales para obtener en todo momento el nombre del usuario en sesión y la cabecera correspondiente al rol de usuario en sesión. El funcionamiento de ambos es muy similar y solo cambian las variables declaradas. Se utilizan mediante el *hook* de React descrito en otros apartados: *useContext*.

```
import React, {useState} from "react";

const loggedUser = "";

export const User = React.createContext();

const Auth = ({children}) =>{
    const [logged, setLogged] = useState(loggedUser);

    return (
        <User.Provider value={[logged, setLogged]}>{children}</User.Provider>
    );
};
```

Fig.6.40. *Extracto de código de Auth.js*

6.2.2. Servidor

La parte del servidor es la que se encuentra en ejecución todo el tiempo, y responde a las peticiones del cliente. Consiste en un servidor express que se declara mediante el paquete npm *express*. Este paquete sirve para implementar, configurar y probar el servidor del backend y, junto con otros paquetes, habilitar el intercambio de información entre el frontend y el backend.

```
const express = require("express");
const app  = express();
```

Fig.6.41. *Declaración del servidor express.*

La configuración del servidor express se realiza con la función *use()*, añadiendo las variables necesarias de otros paquetes npm para el correcto funcionamiento de este.

```
app.use(cors({
    origin: ["http://localhost:3000", "http://[redacted]/"],
    methods: ["GET", "POST", "PUT", "DELETE"],
    credentials: true
}));
app.use(express.json());
app.use(cookieParser());
app.use(bodyParser.urlencoded({extended: true}));
app.use(session({
    name: "userId",
    key: "userId",
    secret: "cambiar",
    resave: false,
    saveUninitialized: false,
    cookie: {
        expires: 1000 * 60 * 60 *  24
    }
}));
```

Fig.6.42. *Configuración del servidor express.*

Para garantizar la comunicación cliente-servidor y con las APIs externas y el tratado de datos de las respuestas HTTP, se han utilizado los paquetes npm *axios* (peticiones REST[1] y comunicaciones frontend-backend y backend-APIs externas), *body-parser* (habilita el *middleware* para el intercambio de información entre el cliente y el servidor), *cors* (para solventar la política de mismo origen en el navegador y habilitar determinadas urls) y *cookieParser* (para la sesión de usuario).

La puesta en marcha del servidor se realiza con la función *listen()*, que tiene como parámetros el puerto en el que se va a alojar el servidor y el código que ejecutará una vez empiece a hacerlo.

```
app.listen(3001, ()=>{
    console.log("server running on port 3001...");
});
```

Fig.6.43. *Servidor ejecutándose en el puerto 3001.*

La parte del servidor es también la encargada de enviar los correos, supervisar la conexión con las cabeceras, comunicarse con las APIs de las cabeceras, modificar la base de datos, entre otras funcionalidades.

Contiene implementada una API compuesta de numerosas funciones destinadas a múltiples funcionalidades: gestión de datos, envío de correos, comunicación con las APIs externas de las instalaciones, programación de funciones a horas concretas, entre otras.

Todo el código que compone las funciones que completan la parte del servidor de la aplicación se encuentra en un fichero *index.js* que se mantiene activo todo el rato. Cada vez que el cliente ejecute funciones que necesiten del backend, el servidor ejecuta y devuelve al cliente la respuesta correspondiente a cada una de sus peticiones.

Dentro de todas estas funciones se pueden diferenciar tres tipos principales: funciones con acceso a la base de datos, funciones que se comunican con las APIs de las cabeceras de las instalaciones tanto de banda ancha como de banda estrecha y funciones que gestionan el envío de correos. Cada grupo de funciones necesita de un paquete npm clave.

---

1. REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP. Utiliza los mismos verbos GET, POST, PUT y DELETE.

Funciones con acceso a datos

Las funciones de gestión de datos se caracterizan por realizar consultas a la base de datos, ya sean de tipo INSERT, SELECT, UPDATE o DELETE, utilizando el paquete npm *mysql* para conectarse a la base de datos.

```
const mysql = require("mysql");
const db = mysql.createPool({
    host: "localhost",
    user: "[redacted]",
    password: "[redacted]",
    database: "smartcity"
});
```

Fig.6.44. *Configuración mysql para posteriores consultas.*

Algunos ejemplos de funciones con acceso a datos se muestran en las siguientes capturas:

```
app.post("/api/getInst",(req,res)=>{
    const user = req.body.username;
    const sqlSelect="SELECT * FROM instalacion JOIN bandaancha ON instalacion.nombre = bandaancha.cabecera WHERE cliente = ?";
    db.query(sqlSelect,user,(err,result)=>{
        res.send(result);
    })
});
```

Fig.6.45. *Función para obtener las instalaciones de banda ancha de un determinado cliente.*

```
const sqlInsert2 = "INSERT INTO alarmtypes VALUES (?,1,1,1,1,1,1)";
db.query(sqlInsert2, username,(err2,result2)=>{
    if(err2){
        console.log(err2);
        res.send(err2);
    }else{
        console.log(result2);
        res.send(result2)
    }
});
```

Fig.6.46. *Fragmento de código que, al insertarse un usuario, inserta un registro en la tabla 'alarmtypes' con la configuración de envío de alarmas por defecto.*

```
app.put('/api/updatePass',(req,res)=>{
    const user = req.body.username;
    const pass = req.body.password;
    bcrypt.hash(pass, saltRounds, (err,hash)=>{
        const sqlUpdate="UPDATE usuario SET password = ? WHERE username = ?";
        db.query(sqlUpdate,[hash,user],(err,result)=>{
            if(err) {
                console.log(err);
            }else{
                console.log(result);
                res.send(result);
            }
        });
    })
;})
```

Fig.6.47. *Función que actualiza la contraseña de un usuario.*

```
app.delete('/api/deleteInst/:i',(req,res)=>{
    const inst = req.params.i;
    const sqlDelete="DELETE FROM instalacion WHERE nombre = ?";
    db.query(sqlDelete,inst,(err,result)=>{
        if(err) {
            console.log(err);
            res.send(err);
        }else{
        console.log(result);
        res.send(result);
        }
    });
;})
```

Fig.6.48. *Función que elimina de la base de datos una instalación dado su nombre.*

Funciones con conexiones externas

Para la comunicación con las APIs de las diferentes instalaciones, se han utilizado principalmente los paquetes npm *axios* y *ssh2* (para la conexión con la red de las cabeceras de banda ancha).

```
app.post("/api/json",(req,res)=>{
    const port = req.body.puerto;
    var data = JSON.stringify({"request_command":50,"data":null});
    var config = {
        timeout: 1000 * 60,
        method: 'get',
        url: 'http://[redacted]:'+port+'/jsonapi/control',
        headers: {
            'Authorization': '[redacted]',
            'Content-Type': 'application/json'
        },
        data : data
    };
Axios(config)
.then(function (response) {
    res.send(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log("ERR");
  res.send("ERR");
});
});
```

Fig.6.49. *Función que obtiene alarmas de banda ancha mediante la API de JSON.*

```
function scanBE(admin, adminPass, imei, callback){

    Axios.get('https://telegestion.solarpowerinnovations.es/telegestion/Services/ApiSPLIT.php?user='+admin+
    '&password='+adminPass+'&imei='+imei+'&group=16&device=6&command=SCAN&start=0&quantity=240')
    .then((response)=>{
        callback(response);
    });
}
```

Fig.6.50. *Función que realiza el escaneo de nodos en una instalación de banda estrecha.*

```
app.post("/api/ping", (req,res)=>{
    const sshport = req.body.port;
    var u = "";
    if(req.body.user==='u'){
        u="[redacted]";
    }else{
        u="[redacted]";
    }
    console.log(sshport+" "+u)
    const { Client } = require('ssh2');
    const connection = new Client();
    connection.on('error', function(err){
        //console.log(err);
        res.send("ERR");
        connection.end();
    });
    connection.on('ready', function(){
        res.send("Éxito al conectar con la cabecera del puerto "+sshport+".");
        connection.end();
    });
    connection.connect({
        host: '[redacted]',
        port: sshport,
        user: u,
        password: '[redacted]',
        readyTimeout: 30*1000
    });
});
```

Fig.6.51. *Función que realiza la comprobación de conexión con cabeceras de banda ancha.*

Funciones de envío de correos

El envío de los correos de registro de usuarios y los de las alarmas se lleva a cabo gracias al paquete npm *nodemailer*, utilizando un correo creado expresamente para la aplicación. Ambas hacen uso de la función *sendEmail()* para el envío de sus respectivos mensajes.

```
const nodemailer = require('nodemailer');
function sendEmail(sendTo, subj, txt, callback){
    const transporter = nodemailer.createTransport({
        service: 'gmail',
        auth: {
          user: 'leycolanalarmas@gmail.com',
          pass: [redacted]
        }
      });
      const mailOptions = {
        from: 'leycolanalarmas@gmail.com',
        to: sendTo,
        subject: subj,
        html: txt
      };
      transporter.sendMail(mailOptions, function(error, info){
        if (error) {
          console.log(error);
          callback("ERR");
        } else {
          console.log('Email sent: ' + info.response);
          callback("SENT");
        }
      });
}
```

Fig.6.52. *Extracto de código que muestra la función que se utiliza para enviar correos.*

Funciones programadas

Para lograr que el servidor monitorice de manera continua el estado de las instalaciones y envíe correos y demás a lo largo del día, se necesita que las funciones que comprueben dicho estado se ejecuten periódicamente. Para ello resulta crucial el paquete npm *cron*.

Este paquete permite configurar la ejecución de código a unas horas del día, días de la semana, días del mes y meses concretos. Su funcionamiento es idéntico al *cron* de Linux.

```
cron.schedule('0 23 * * *', () => {
    insertAlarmsBA();
});
cron.schedule('0 5 * * *', () => {
    insertAlarmsBA();
});
```

Fig.6.53. *Cada día a las 23:00 y a las 5:00 se ejecuta la función que comprueba si hay alarmas de banda ancha y las inserta.*

```
cron.schedule('*/30 * * * *', () => {
    var h = new Date().toLocaleTimeString();
    console.log("Enviando alarmas ("+h+")...");
    sendAlarms();
});
```

Fig.6.54. *Envío de alarmas programado cada media hora.*

Algunas de las funciones que se ejecutan más frecuentemente son las lecturas de potencia de las instalaciones de banda estrecha y la comprobación de conexión o pings a las cabeceras de banda ancha.

```
conectando con cabeceras y comprobando potencias... 16:53:08
SorozabalA: 3.3W
GaztelutxoB: 3.35W
Sorgintzulo: 3.34W
GaztelutxoA: 2.8W
SorozabalB: 3.39W
```

Fig.6.55. *Ejemplo de la salida del terminal del servidor tras ejecutarse la lectura de las potencias (instalaciones apagadas).*

Callback

Una callback en JavaScript es una función que debe ser ejecutada después de que otra función haya finalizado su ejecución. Una definición más formal sería: cualquier función que se pasa como un argumento a otra función para que pueda ser ejecutada en esa otra función se llama como *callback*.

Debido al carácter asíncrono de javascript, ha sido necesario implementar funciones con callbacks para la sincronización de múltiples funciones que necesitaban la respuesta de otra anterior para funcionar correctamente.

Un ejemplo claro de uso de este tipo de funciones se encuentra en la función que comprueba la conexión de las cabeceras de banda ancha. Según los requisitos funcionales para la comprobación de la conexión con estas instalaciones, se debe realizar un máximo de tres intentos de conexión para determinar el estado de cada cabecera.

Para ello, se ha utilizado un callback en la función de conexión con la cabecera, para posteriormente poder realizar un intento detrás de otro. Si no se hubiese incorporado este sistema de sincronización, las tres peticiones se harían a la vez. Hasta que un *ping* no devuelva respuesta, no se realiza otro intento.

```
function pingInstBA(inst, callback){
    var ssh = inst.sshport;
    var sshuser = inst.linux;
    Axios.post("http://localhost:3001/api/ping",{
        port: ssh,
        user: sshuser
    }).then((response)=>{
        callback(response ? response : "");
    }).catch((error)=>{
        callback(error ? "ERR" : "");
    });
}
```

Fig.6.56. *Función que comprueba la conexión con una cabecera de banda ancha.*

```
function pingAllInstBA(callback){
    var date = new Date();
    var f = date.toISOString().split('T')[0];
    var h = new Date().toLocaleTimeString();
    getAllInstBA((inst)=> {
        console.log("\ncomprobando conexión con las instalaciones...");
        var alarms=[];
        var cont = inst.length;
        inst.forEach((e) => {
            console.log("conectando con cabecera de "+e.nombre+"...");
            var connected=false;
            pingInstBA(e, (res)=>{
                if(res.data!=="ERR"){
                    console.log(e.nombre+" conecta");
                    connected=true;
                    cont--;
                    flush5alarms(e.nombre);
                    return;
                }if(!connected){
                    pingInstBA(e,(re)=>{
                        if(re.data!=="ERR"){
                            console.log(e.nombre+" conecta");
                            connected=true;
                            cont--;
                            flush5alarms(e.nombre);
                            return;
                        }if(!connected){
                            pingInstBA(e, (r)=>{
                                if(r.data!=="ERR"){
                                    console.log(e.nombre+" conecta");
                                    connected=true;
                                    cont--;
                                    flush5alarms(e.nombre);
                                    return;
                                }
                                console.log(e.nombre+" NO conecta");
                                alarms.push({
                                    id: 0,
                                    type: 5,
                                    instalacion: e.nombre,
                                    fecha: f,
                                    descripcion: "No se puede conectar con la cabecera de la instalación",
                                    hora: h,
                                    horaenvio: '',
                                    enviada: false
                                });
                                if(alarms.length===cont){
                                    callback(alarms);
                                }
                            });
```

Fig.6.57. *Función que comprueba la conexión con todas las instalaciones de banda ancha.*

7

Pruebas

En este capítulo se describen las pruebas realizadas para comprobar y corregir el funcionamiento de la aplicación. Para el lado del cliente se han realizado casos de prueba con diferentes escenarios y para el lado del servidor se han utilizado funciones que se ejecutan periódicamente.

7.1. Casos de prueba

Para comprobar el funcionamiento del lado del cliente, se han realizado casos de pruebas de cada funcionalidad para la cual el usuario debe introducir datos o seleccionar alguna opción de la aplicación para obtener un resultado o salida.

7.1.1. Casos de prueba: cliente

Casos de prueba de inicio de sesión

Datos correctos

Entrada: nombre de usuario y contraseña correctos y que se encuentran en la base de datos.
Salida esperada: inserción exitosa y acceso a página de inicio.
Salida obtenida: salida esperada.

Datos incorrectos

Entrada: nombre de usuario y contraseña correctos pero que no se encuentran en la base de datos.
Salida esperada: mensaje indicando que los datos introducidos son incorrectos.
Salida obtenida: salida esperada.

Campos sin rellenar

Entrada: nombre de usuario introducido, contraseña vacía.
Salida esperada: mensaje indicando que faltan campos por rellenar.
Salida obtenida: salida esperada.

El resto de casos de prueba del cliente se encuentran en el anexo C.

7.1.2. Casos de prueba: administrador

Gestión

Inserción de usuario

- Inserción correcta
  Entrada: usuario no existente en base de datos, campos de contraseña y correo correctos.
  Salida esperada: mensaje indicando la inserción del usuario, correo enviado a la dirección indicada y base de datos actualizada con el nuevo registro.
  Salida obtenida: salida esperada

- Usuario ya existente
  Entrada: usuario a introducir existe en base de datos, campos de contraseña y correo correctos.
  Salida esperada: mensaje de error indicando que ya existe ese usuario, base de datos sin cambios.
  Salida obtenida: salida esperada

- Campos sin rellenar
  Entrada: todos los campos excepto el del email rellenos y correctos.
  Salida esperada: mensaje indicando que faltan campos por rellenar, base de datos sin cambios.
  Salida obtenida: salida esperada.

Inserción de instalación

- Inserción correcta (instalación de banda ancha)
  Entrada: nombre de instalación correcto, tipo de usuario seleccionado y puertos correctos.

Salida esperada: mensaje indicando que se ha insertado una nueva instalación de banda ancha, cambios en base de datos: tablas *instalación* y *bandaancha*.
Salida obtenida: salida esperada.

- Inserción correcta (instalación de banda estrecha)
  Entrada: nombre de instalación, imei, datos de administrador e instalador correctos.
  Salida esperada: mensaje indicando que se ha insertado una nueva instalación de banda estrecha, cambios en base de datos: tablas *instalación* y *bandaestrecha*.
  Salida obtenida: salida esperada.

- Instalación ya existente
  Entrada: todo correcto, pero el nombre de instalación a insertar ya existe en la base de datos.
  Salida esperada: mensaje de error indicando que la instalación ya existe, base de datos sin cambios.
  Salida obtenida: salida esperada.

- Campos sin rellenar
  Entrada: todos los campos rellenos y correctos excepto algunos.
  Salida esperada: mensaje indicando que faltan campos por rellenar, base de datos sin cambios.
  Salida obtenida: salida esperada.

7.2. Pruebas con instalaciones reales

Además de comprobar el funcionamiento del lado del cliente con los casos de prueba mostrados, también se ha llevado a cabo un serie de pruebas en la parte del servidor, pero estas últimas son un tanto diferentes a las anteriores, ya que no se tratan de casos de prueba al uso, sino de una serie de pruebas periódicas mediante la utilización de algunos paquetes npm.

7.2.1. Datos utilizados para las pruebas

Usuarios (rol)*

- admin (administrador)
- Felix (administrador)
- Aroa (administrador)
- Errenteria1 (cliente)
- Errenteria2 (cliente)
- Legorreta (cliente)

Instalaciones (usuario)*

- Banda ancha
  - AlabergaGoikoa (Errenteria1)
  - GaltzarabordaTopoa (Errenteria1)
  - Agustinak (Errenteria1)
  - Hirigune (Errenteria1)
  - TxirritaMaleo (Errenteria1)
  - Iztieta (Errenteria1)
  - Ondartxo (Errenteria1)

    - Esnabide (Errenteria2)
    - Ernio (Errenteria2)
    - AlabergaBehekoa (Errenteria2)
    - GaltzaParkea (Errenteria2)
    - ParkeKalea (Errenteria2)
    - Galtzaraborda52 (Errenteria2)
    - Igerilekua (Errenteria2)
    - Beraun (Errenteria2)
    - LegorretaEskola (Legorreta)
    - LegorretaParking (Legorreta)
    - LegorretaUdaletxea (Legorreta)

- Banda estrecha
    - Sorgintzulo (Errenteria1)
    - SorozabalA (Errenteria1)
    - SorozabalB (Errenteria1)
    - GaztelutxoA (Errenteria1)
    - GaztelutxoB (Errenteria1)

* No se muestran los valores de los correos electónicos, puertos, direcciones ip ni contraseñas por motivos de seguridad

7.2.2. Implementación de pruebas periódicas

Como ya se ha indicado en el apartado de implementación, en la sección del servidor, se ha podido llevar a cabo una serie de pruebas periódicas a horas concretas del día para comprobar el funcionamiento del mismo en un entorno más real, debido a que esta aplicación está pensada para que sirva de vigilante de las instalaciones.

Dichas pruebas se han llevado a cabo gracias a una de las funciones del paquete npm *cron, schedule()*. Esta función permite configurar la ejecución del código deseado en tiempos concretos pasados por parámetro (ver fig.6.53).

El parámetro en cuestión es una cadena de caracteres como la que se utiliza con el *cron* de linux para programar la ejecución de funciones o scripts [17]. Existen páginas web como crontab.guru que permiten probar con distintos valores para ver cuándo se ejecutaría el código incluido en la función *schedule()* mencionada.

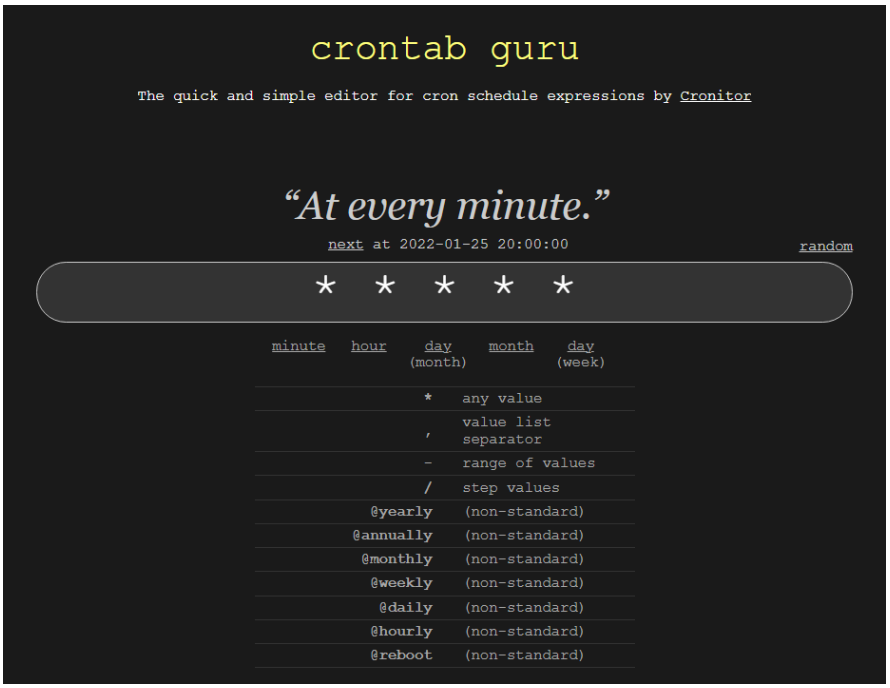


Fig.7.1. *Captura de pantalla de la web crontab.guru.*

Las funciones que han sido probadas y ejecutadas periódicamente utilizando este método se encuentran en el anexo C de esta memoria.

8

Seguimiento y Control

En este capítulo se analiza el desarrollo general del proyecto y si ha habido desviaciones o los riesgos previstos han tenido algún impacto. Se describen también las desviaciones con respecto a las estimaciones y periodos de desarrollo de las tareas.

8.1. Desarrollo general del proyecto

El proyecto ha sido finalizado con éxito y se han cumplido tanto los objetivos propuestos como los requisitos del cliente.

8.1.1 Desviaciones con respecto al plan inicial y riesgos

En general el desarrollo del proyecto ha ido más o menos según lo previsto, a pesar de que debido a algunos problemas iniciales en cuanto a comunicaciones con la empresa, el desarrollo de la aplicación comenzó algunas semanas más tarde de lo inicialmente planeado. Esto se debe a la alta carga de trabajo que estaba gestionando la empresa en el momento de iniciar el proyecto, dificultando la posibilidad de realizar una reunión inicial.

En cuanto a los riesgos analizados en un principio, el estudio de herramientas y tecnologías ha llevado más tiempo del estimado. Los cambios en algunos requisitos por peticiones del clientes no han supuesto casi dificultades debido a que semanalmente se han realizado reuniones de seguimiento para analizar el trabajo llevado a cabo y el trabajo a realizar hasta la siguiente reunión. Como el despliegue se ha realizado al finalizar todo el proyecto, el tema de las conexiones con la base de datos y demás no ha supuesto problemas debido a que la totalidad del desarrollo y las pruebas se han realizado en local.

8.2. Desviaciones de las estimaciones con respecto a la planificación

En términos generales, no ha habido ninguna desviación significativa en cuanto a la planificación inicial del proyecto, salvo en el caso del estudio de tecnologías y herramientas a utilizar para el desarrollo de la aplicación, que ha llevado bastante más tiempo del estimado.

Además de eso, cabe mencionar que debido a las dificultades iniciales para arrancar con el desarrollo software, se acordó con la empresa dar prioridad a las tareas relacionadas con el proyecto software, y dejar para el final todo lo relacionado con la documentación, como esta memoria.

Al ser el primer proyecto de este estilo que se ha llevado a cabo, y teniendo en cuenta que todo el trabajo lo ha hecho una sola persona, el periodo de desarrollo de las tareas se ha terminado entremezclando, ocasionando que se trabajen en tareas de diferentes paquetes al mismo tiempo, cuando en un principio la idea de trabajo era distinta.

8.2.1. Tabla comparativa

En la siguiente tabla se muestra de una manera más gráfica las comparaciones entre los tiempos estimados a invertir en cada paquete de trabajo al inicio del proyecto y los tiempos finalmente invertidos.

Paquetes de Trabajo	Estimación (h)	Tiempo real (h)	Desviación (h)
Tecnologías y Herramientas (TyH)	10	25	15
Backend (BE)	90	97	7
Frontend (FE)	80	82	2
Base de Datos (BD)	20	17	-3
Solución Web (SW)	10	6	-4
Documentación (DOC)	40	53	13
Control de Calidad (CC)	30	25	-5
Testeo (T)	30	28	-2
TOTAL	310	333	23

Tabla 8.1. *Tabla comparativa de estimaciones y tiempos finales.*

Se puede observar que a pesar de ser un proyecto de grandes dimensiones, las estimaciones de tiempos no han estado tan desacertadas. Cabe destacar que el tiempo de estudio de las tecnologías a incluir en el desarrollo ha sido mucho mayor del planificado al inicio.

Se ha invertido más tiempo del estimado en las tareas de desarrollo tanto del frontend como del backend del sistema y ha habido un desvío ligeramente más significativo en el desarrollo de la documentación.

En el resto de paquetes las estimaciones han estado acertadas en mayor o menor medida, y los tiempos planificados inicialmente se han acabado acercando bastante al resultado final.

8.2.2. Diagrama de Gantt

Por último, se pueden observar los tiempos de desarrollo del proyecto de manera gráfica, utilizando un diagrama de Gantt.

En él se pueden observar varios puntos comentados anteriormente:

- El estudio de tecnologías ha llevado más tiempo del esperado.
- El comienzo del desarrollo de la parte del proyecto software se tuvo que posponer debido al retraso de la reunión inicial, ocasionando una mayor carga de trabajo relacionada con estos paquetes en un marco de tiempo menor.
- El desarrollo de la memoria se comienza a realizar tras terminar la aplicación.

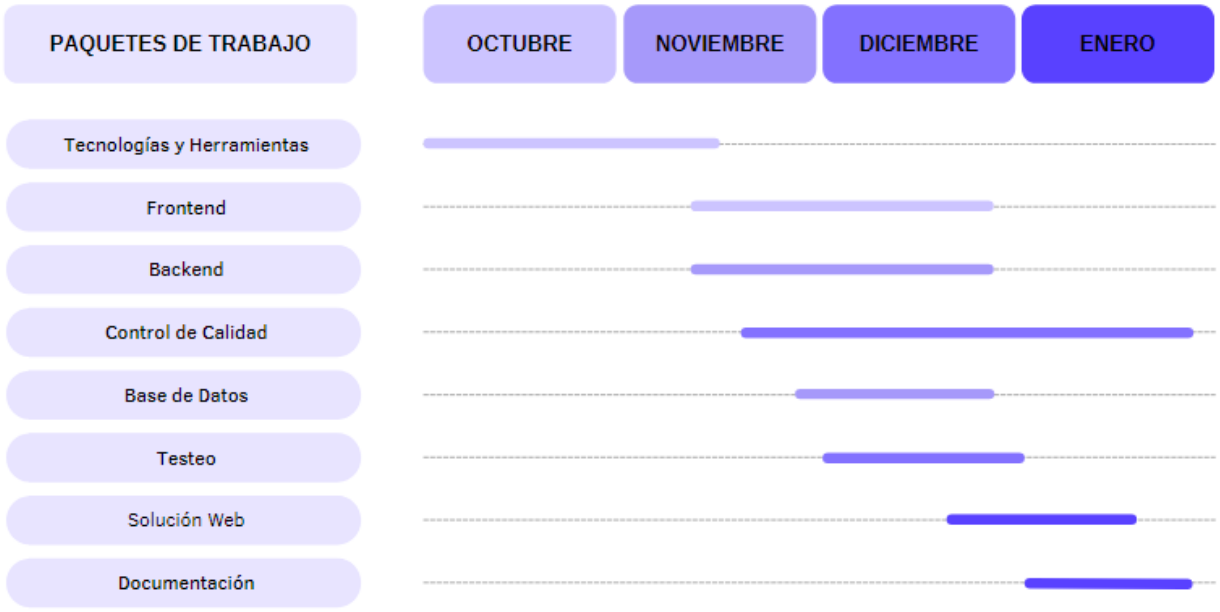


Fig.8.1. *Diagrama de Gantt del desarrollo de los paquetes de trabajo del proyecto.*

9

Conclusiones

En este último capítulo, se realizan algunas reflexiones con respecto al trabajo realizado durante este proyecto, se indican algunas mejoras con respecto al estado actual de la aplicación y se detallan varias líneas de trabajo para desarrollar en un futuro.

9.1. Posibles mejoras

Una de las mejoras que necesita la aplicación es la de refactorización de código, que al haber sido implementada con herramientas que nunca he utilizado o he usado muy poco, para cuando me he acostumbrado a trabajar con ellas, no había tiempo suficiente como para revisar y limpiar el código escrito hasta el momento a la vez que seguía con la aplicación. No tanto en la parte del cliente, pero la parte del servidor sí necesita un repaso medianamente urgente y una limpieza de código exhaustiva, pero necesaria.

Otro de los posibles cambios a tener en cuenta tiene que ver con la información mostrada al usuario final. Al tratarse de una aplicación cuyo cliente es una empresa dedicada a las telecomunicaciones en el ámbito de la ingeniería electrónica, no he tenido en cuenta que a veces la aplicación muestra información con pocos detalles que solo entienden aquellos puestos en el tema. Esto se da por ejemplo en la parte que se muestran las alarmas al cliente, en lugar de aparecer una etiqueta en el nivel de urgencia de las alarmas aparece simplemente un número. Pasa lo mismo para la página del perfil, que muestra solo los números en vez de algún texto que explique qué significa cada nivel de las alarmas.

Sin duda otro aspecto a mejorar es el diseño de la aplicación, que es bastante simple y se puede mejorar infinitamente, y más con las herramientas que ofrecen los frameworks de hoy en día que permiten hacer todo de manera mucho más rápida y sencilla.

9.2. Líneas de trabajo futuras

Alguna de las primeras ideas como parte de continuar con el desarrollo de la aplicación que ya he hablado con la empresa es la de integrar funcionalidades de nuevos tipos de instalación a la plataforma. Debido a que el tiempo para el proyecto no es muy extenso, solo ha dado tiempo a trabajar con dos tipos, pero existen otras instalaciones con las que ya trabajan que sería interesante ir añadiendo (instalaciones bluetooth, instalaciones solares, etc).

Además de integrar funcionalidades de nuevos tipo de instalaciones, se tratará de seguir añadiendo funcionalidades a los tipos de instalación ya existentes, para ampliar el abanico de gestiones que se pueden realizar con las instalaciones dentro de la propia aplicación.

9.3. Conclusiones del proyecto

Tras la finalización del periodo de desarrollo del proyecto, se ha cumplido con el objetivo inicial de desarrollar una plataforma web que cumpliese con los requisitos establecidos con la empresa. El cliente, o en este caso el responsable del proyecto en la empresa me ha llegado a comunicar en varias ocasiones que le gusta y le parece interesante la manera en la que se gestionan los correos mediante la aplicación. Ha sido una experiencia totalmente diferente a cualquier proyecto realizado anteriormente durante la carrera o en cualquier otro ámbito.

El reunirse semanalmente con responsables de una empresa que observan tu trabajo y opinan acerca de él, indicando los siguientes pasos a seguir o aspectos a corregir, ha hecho que tenga una idea aproximada de cómo se trabaja en el ámbito del desarrollo de software en un entorno profesional. Además, el hecho de no tener a nadie especializado en ingeniería informática que me pudiera ayudar o guiar desde la empresa, y el tener que seleccionar todas las herramientas por mí mismo, me ha forzado a buscar en miles de sitios diferentes para descubrir qué es lo que fallaba y conseguir terminar con éxito la aplicación, siendo cada paso que daba bastante satisfactorio, la verdad.

Con todo esto, y habiendo tenido que aprender e investigar sobre nuevas tecnologías para llevar a cabo el proyecto, ha sido un proyecto muy enriquecedor y satisfactorio de completar en cuanto a que he aprendido acerca de herramientas nuevas muy interesantes, sobre todo de cara al futuro, y a desenvolverme en un entorno más profesional.

Por último, agradecer a Alfredo por la ayuda y dirección con la memoria del tfg. A Félix y Aroa por aguantarme cada semana enseñándoles cómo iba avanzando con la aplicación. A mis padres por brindarme la oportunidad de haber estudiado este grado. Finalmente dar las gracias sobre todo a mis compañeros por estar siempre ahí, por todos esos buenos y malos momentos durante estos años de carrera, gracias mis panas.

Bibliografía

[1] *Empresa - Leycolan Sal.* https://www.leycolan.com

[2] Jessica Clark: *The Best Ten Backend Languages*. https://blog.back4app.com/best-backend-language

[3] Jessica Clark: *Top 10 Frontend Languages*. https://blog.back4app.com/front-end-languages

[4] freeCodeCamp: *The Advantages and Disadvantages of JavaScript*. https://www.freecodecamp.org/news/the-advantages-and-disadvantages-of-javascript/

[5] Sanjeev Besra: *Advantages And Disadvantages of HTML You Should Know*. https://www.techtually.com/advantages-and-disadvantages-of-html/

[6] Pawan Sahu: *Advantages and Disadvantages of CSS Everyone Should Know*. https://www.motocms.com/blog/en/advantages-and-disadvantages-of-css/

[7] Jeel Patel: *List of 10 Best Front end Frameworks to Use For Web Development*. https://www.monocubed.com/best-front-end-frameworks/

[8] Jessica Clark: *Los mejores marcos de trabajo de backend*. https://blog.back4app.com/es/los-10-mejores-marcos-de-trabajo-de-backend

[9] Krunal Shah: *Pros and Cons of Node.js Web App Development*. https://www.thirdrocktechkno.com/blog/pros-and-cons-of-node-js-web-app-development/

[10] Álvaro Insignares: *React Pros and Cons: What are the Advantages and Disadvantages of ReactJS?*. https://www.koombea.com/blog/react-pros-and-cons-what-are-the-advantages-and-disadvantages-of-reactjs/

[11] Presentando JSX - React. https://es.reactjs.org/docs/introducing-jsx.html

[12] TrustRadius, Eclipse reviews. https://www.trustradius.com/products/eclipse/reviews

[13] TrustRadius, Visual Studio Code reviews. https://www.trustradius.com/products/microsoft-visual-studio-code/reviews

[14] Priya Pedamkar: *What is Visual Studio Code?*. https://www.educba.com/what-is-visual-studio-code/

[15] James Gallagher: *Most Popular Web Development Frameworks in 2021*. https://careerkarma.com/blog/top-web-frameworks-2021/

[16] Referencia de la API de los Hooks. https://es.reactjs.org/docs/hooks-reference.html

[17] Linux man page. https://linux.die.net/man/5/crontab

Anexo A: Diagrama UML

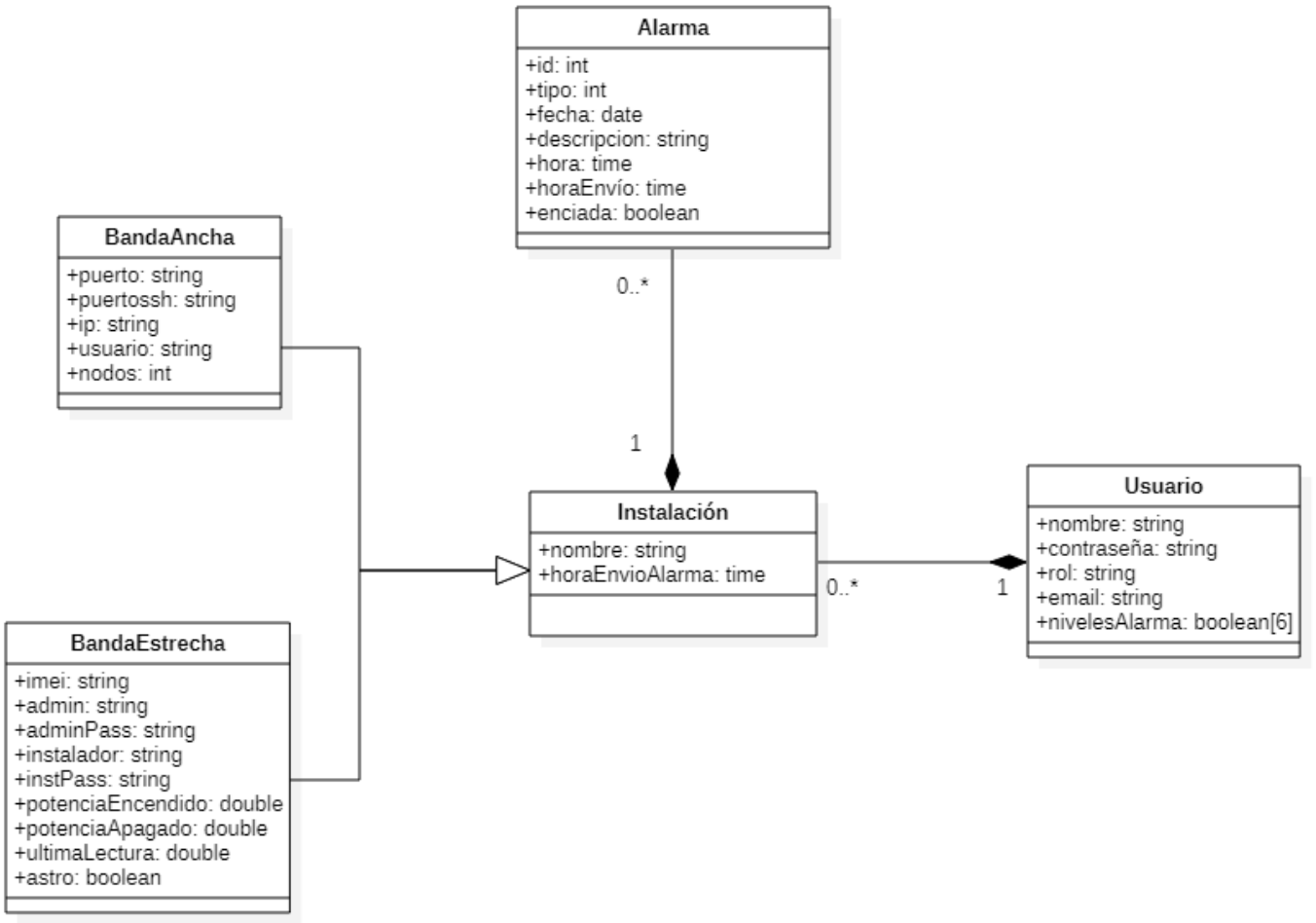
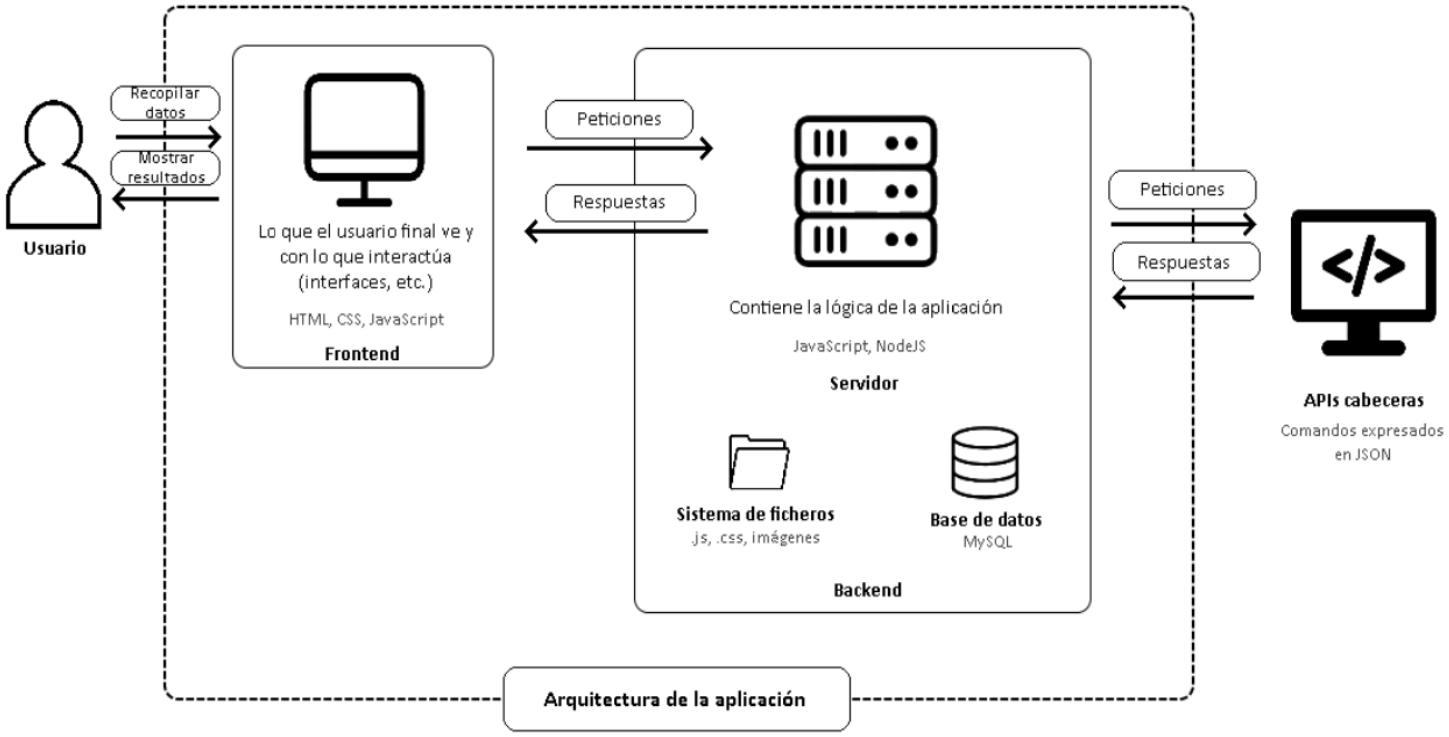


Diagrama UML correspondiente a los datos almacenados en la base de datos.

Anexo B: Arquitectura de la aplicación



Arquitectura de la aplicación web desarrollada.

Anexo C: Casos de Prueba

Casos de prueba realizados

Casos de prueba para la página de perfil

Cambio de hora

Entrada: selección de una de las horas disponibles en la lista desplegable.
Salida esperada: actualización tanto en base de datos como en
Salida obtenida: salida esperada.

Cambio de configuración de niveles de alarma a recibir

Entrada: todos los niveles seleccionados, el usuario deselecciona los niveles 1 y 4.
Salida esperada: se actualizan los valores en la base de datos y al volver a acceder al perfil, 1 y 4 están sin seleccionar.
Salida obtenida: salida esperada.

Cambio de contraseña

- Datos correctos
  Entrada: contraseña actual correcta, nueva contraseña diferente a la actual y contraseña repetida coincide con la nueva.
  Salida esperada: actualización de contraseña en base de datos y mensaje indicándolo.
  Salida obtenida: salida esperada.

- Campos sin rellenar
  Entrada: contraseña actual vacía, el resto correcto.
  Salida esperada: mensaje indicando que faltan campos por rellenar, sin cambios en base de datos.
  Salida obtenida: salida esperada.

- Contraseña actual incorrecta
  Entrada: contraseña actual incorrecta, resto de campos correctos.
  Salida esperada: mensaje indicando que la contraseña actual no coincide, sin cambios en base de datos.
  Salida obtenida: salida esperada.

- Contraseñas nuevas diferentes
  Entrada: contraseña actual correcta y repetida diferente a la nueva introducida.
  Salida esperada: mensaje indicando que las nuevas contraseñas no coinciden, sin cambios en base de datos.
  Salida obtenida: salida esperada.

- Contraseña nueva idéntica a la actual
  Entrada: contraseña actual correcta, pero nueva y repetida iguales a la actual.
  Salida esperada: mensaje indicando que la nueva contraseña no puede ser igual a la actual, sin cambios en la base de datos.
  Salida obtenida: salida esperada.

Casos de prueba para la página de instalaciones

Prueba de conexión de banda ancha

- Conexión exitosa
  Entrada: prueba de conexión con cabecera en marcha.
  Salida esperada: alerta informativa indicando la conexión con éxito al puerto en el que se encuentra la cabecera.
  Salida obtenida: salida esperada.

- Error de conexión
  Entrada: prueba de conexión con una cabecera sin conexión/cobertura.
  Salida esperada: alerta informativa indicando que ha habido un error de conexión con el puerto en el que se encuentra la cabecera.
  Salida obtenida: salida esperada.

Comprobación de alarmas de banda ancha

- Comprobación correcta
  Entrada: comprobación de alarmas de una instalación funcionando correctamente.
  Salida esperada: alerta informativa mostrando las alarmas que ha devuelto la API.
  Salida obtenida: salida esperada.

- Error de conexión
  Entrada: comprobación de alarmas de una instalación cuyo servidor que contiene la API para las alarmas no está funcionando.
  Salida esperada: alerta informativa indicando que ha habido un error de conexión con el servidor de la API de alarmas.
  Salida obtenida: salida esperada.

Lectura de potencia de banda estrecha

- Lectura correcta
  Entrada: lectura de potencia de una cabecera en marcha sin errores de conexión.
  Salida esperada: alerta informativa con la potencia entregada (en W) de la instalación seleccionada.
  Salida obtenida: salida esperada.

- Error de lectura
  Entrada: lectura de potencia de una cabecera en marcha con errores de conexión o de exceso de temperatura.
  Salida esperada: alerta informativa indicando el error de lectura.
  Salida obtenida: salida esperada.

Escaneo de instalación de banda estrecha

- Escaneo correcto
  Entrada: escaneo de una cabecera en marcha sin problemas de conexión ni exceso de temperatura.
  Salida esperada: alerta informativa con los identificadores de los nodos con errores.

Salida obtenida: salida esperada.

- Error de escaneo
  Entrada: escaneo de una cabecera en marcha con problemas de conexión o exceso de temperatura.
  Salida esperada: alerta informativa con un mensaje de error.
  Salida obtenida: salida esperada.

Casos de prueba para la página de alarmas

Comprobación de alarmas

- Existe registro de alarmas en la fecha seleccionada
  Entrada: selección de una fecha en la que existen alarmas en la base de datos.
  Salida esperada: tabla con las alarmas encontradas.
  Salida obtenida: salida esperada.

- No existe registro de alarmas en la fecha seleccionada
  Entrada: selección de una fecha para la que no hay alarmas en la base de datos.
  Salida esperada: mensaje indicando que no hay alarmas.
  Salida obtenida: salida esperada.

Funciones utilizadas en las pruebas periódicas

- Comprobación de conexión de cabeceras de banda ancha.
    - Varios días, a todas horas, cada 15 minutos.

- Comprobación e inserción de alarmas de banda ancha.
    - Varios días, a las 23:00 y a las 5:00.
        - Comprobación del estado de la instalación al poco de encenderse y poco antes de apagarse.

- Comprobación de conexión y lectura de potencia de instalaciones de banda estrecha
    - Varios días, a todas horas, cada 15 minutos.
    - Esta función funciona diferente en 2 casos:
        - Cuando detecta el encendido de alguna instalación, comprueba la última lectura de encendido de potencia almacenada en la base de datos, y genera alarma si hay alguna anomalía en dicha lectura.
        - Cuando detecta el apagado de alguna instalación, comprueba la última lectura de apagado de potencia almacenada en la base de datos, y genera alarma si hay alguna anomalía en dicha lectura.

- Comprobación de alarmas de banda estrecha mediante el comando de escaneo de las cabeceras.
    - Varios días, a las 2:00 y 2:30
        - Estos horarios se deben a que las empresas que también trabajan con estas instalaciones realizan sus lecturas a diferentes horas del día. Si hay algún momento en el que la cabecera reciba demasiadas solicitudes, puede ocasionar errores por exceso de temperatura, debido a que el escaneo hace uso de la red eléctrica física para obtener los resultados.

- Envío de alarmas por correo a los clientes registrados.
    - Varios días, todo el día, cada media hora.
        - El hecho de ejecutar esta función cada media hora no es por ninguna razón en especial. Es simplemente porque cada usuario puede programar el envío de sus alarmas a las horas del día que desee, y actualmente la aplicación solo les permite elegir horas en punto o a y media (09:00, 10:30, 17:00, 21:30…).