Grado en Ingeniería Informática

Trabajo de Final de Grado

Desarrollo de una plataforma SmartBuilding (BMS)

Autor:
Ainhoa Tomás Ballester

Supervisor:
Jose Luis Martínez Perez
Tutor académico:
Michael Gould Carlson

Fecha de lectura: 22 de Junio de 2022
Curso académico 2021/2022

Resumen

Este documento describe la memoria del trabajo de final de grado durante la estancia en prácticas en la empresa IoTsens, en el que se expone el desarrollo de una aplicación web para la gestión de edificios inteligentes y gestión de los dispositivos que tienen funcionando con el fin de visualizar ciertos KPIs que elige el cliente.

En el documento se explica cómo ha ido tomando forma la parte del frontend de la aplicación, empezando desde un diseño de prototipos y un diseño de la base de datos hasta la implementación de las vistas de la aplicación. No está de más decir que el frontend de la aplicación se ha desarrollado utilizando el framework de Angular 12 y el lenguaje de programación TypeScript.

Palabras clave

Internet de las cosas, edificios inteligentes, sensores, aplicación web, Angular.

Keywords

Internet of things, smart buildings, sensors, web application, Angular.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto documentado en esta memoria se ha desarrollado en la empresa IoTsens. Iotsens es una empresa proveedora de soluciones verticales del Internet de las Cosas, es decir, recolectan, integran, almacenan y analizan la información de soluciones verticales como Smart Industrial, Smart Water y Smart City [2]. Esta empresa se creó para proveer soluciones verticales basadas en el Internet de las Cosas para las empresas del Grupo Gimeno y desde hace un tiempo evoluciona de forma paralela a dicha sociedad, apostando por la innovación y explorando oportunidades que ofrece el mercado de las nuevas tecnologías.

En esta empresa se han desplegado múltiples soluciones que se dividen en cuatro áreas bien diferenciadas:

- IOTSENS City [3], también conocida como Smart City, es una solución que proporciona la gestión y el control de dispositivos de forma remota en las ciudades, suministra las herramientas necesarias para el procesamiento y análisis de los datos recolectados de los sensores de dichos dispositivos, además de ayudar a optimizar los recursos disponibles en tiempo real.

- IOTSENS Water [4], también llamada Smart Water, proporciona una solución de telelectura de contadores mediante sensores y dispositivos, y también consigue una gestión rápida y eficiente de la red de suministro.

- IOTSENS Industrial [5], ofrece una solución para la gestión y el control de plantas industriales a distancia y permite el almacenamiento y análisis de los datos originados de los PLCs situados en las industrias.

- IOTSENS Custom [6], donde se agrupan los proyectos IoT (Internet of Things o Internet de las Cosas) personalizados con las necesidades de clientes específicos y que no tienen que ver con las soluciones citadas anteriormente. Dichos proyectos se pueden llevar a cabo desde el área de hardware como de software.

La principal motivación del proyecto es solucionar la falta de optimización de recursos clave en los edificios, así como poder tener acceso en tiempo real a los datos globales del edificio, y más específicos de cada planta e incluso poder compararlos entre plantas o fechas. Este proyecto ayudaría a seguir implantando el modelo de Smart City, añadiendo una plataforma más para poder gestionar elementos de una ciudad, en este caso se incorporará el poder gestionar edificios.

1.2. Objetivos y alcance del proyecto

El principal objetivo de este proyecto es diseñar e implementar una aplicación web que permita la gestión y el control de dispositivos de forma remota en los edificios, además de visualizar los distintos KPIs (Key Performance Indicador) relacionados con dichos edificios.

El objetivo principal se puede desglosar en los siguientes subobjetivos:

- Facilitar, mediante una interfaz, la gestión de edificios inteligentes. Dar de alta edificios, sus respectivas plantas y las zonas de cada planta, actualizar dicha información y poder visualizar dicha información en varios listados.
- Visualizar los datos clave o KPI del edificio como de cada una de las plantas del mismo y de las zonas de cada planta a través de gráficos e indicadores.
- Permitir, a los administradores, la gestión de dispositivos. Poder añadir dispositivos, actualizar su información y poder visualizarlos en un listado, así como poder visualizar la información completa de cada dispositivo, sus mediciones y poder interaccionar con él en el caso de que se pueda.
- Alertar, mediante una interfaz, si algún dispositivo supera o desciende de sus límites establecidos, para que el cliente pueda actuar de forma inmediata.

El alcance de este proyecto incluye unas tareas de estado del arte, para poder investigar otros proyectos que tengan el mismo objetivo, la definición e implementación del modelo de datos, el desarrollo frontend y el desarrollo de APIs; pero no incluye el desarrollo del backend ni el desarrollo hardware de los dispositivos.

- Desde el punto de vista organizativo. Las empresas que harán uso de la aplicación serán todas aquellas que sean clientes de IoTsens y que quieran dar el salto para convertir sus edificios/oficinas en edificios inteligentes.
- Desde el punto de vista funcional. La aplicación servirá para gestionar edificios inteligentes para así poder optimizar ,y por consiguiente, evitar desperdiciar sus recursos. Los usuarios finales sólo podrán acceder a los edificios en los que estén asignados. Los datos que podrán visualizar serán todos los elementos relacionados con dichos edificios, como las plantas, las zonas, los dispositivos que se están utilizando y las distintas gráficas en forma de widgets de los KPIs que requieran dichos usuarios.
- Desde el punto de vista informático. La aplicación interaccionará con los dispositivos instalados en el edificio para que el cliente pueda visualizar lo que ocurre a tiempo real en el edificio.

1.3. Descripción del proyecto

En primer lugar, este proyecto consiste en desarrollar una aplicación de Smart Building que recolecte la información de los dispositivos que estén en un edificio, almacene dichos datos y los muestre para que el usuario pueda visualizar en todo momento los datos de los dispositivos y controlar, por ejemplo, que la calidad de aire es buena.

La aplicación debe permitir gestionar edificios inteligentes así como los dispositivos y datos que vienen de esos edificios. De este modo el cliente puede crear y ubicar en un mapa donde se encuentra el edificio que va a gestionar, así como cada planta y zona del edificio. También podrá especificar de qué dispositivos va a hacer uso y en qué zona de cada planta van a estar, a parte de poder visualizar estadísticas del completo funcionamiento del edificio e interactuar con algunos dispositivos.

Finalmente, el cliente podrá crear y visualizar, mediante widgets personalizados, los distintos KPIs que considere oportunos, para la correcta gestión del edificio.

Una aplicación como ésta permitirá reducir el gasto de los recursos que utiliza un edificio y por consiguiente reducirá los costes, y además facilitará la gestión y el mantenimiento del edificio, lo que a largo plazo consistirá en una mayor facilidad de estar alerta a lo que ocurre en el edificio, y además, reducirá los costes de mantenimiento ya que se tendrá una información completa del estado del edificio y de cada planta. Esta aplicación favorecerá el salto hacia los edificios ecosostenibles por la reducción de recursos utilizados.

Cabe mencionar que la parte backend de la aplicación no se incluye en esta memoria, puesto que fue implementada por otro miembro del equipo de desarrollo.

1.4. Estructura de la memoria

Tras este capítulo de introducción, los siguientes capítulos de esta memoria describen las distintas etapas de desarrollo asociadas al proyecto.

- Capítulo 2, planificación del proyecto, donde se explica la metodología que se ha utilizado, la planificación y seguimiento del estado del proyecto, y una estimación de los recursos y costes del proyecto.
- Capítulo 3, análisis y diseño del sistema, donde se muestra la especificación de los requisitos del sistema, el diseño de la base de datos y de los distintos prototipos de la interfaz gráfica.
- Capítulo 4, implementación y pruebas, donde se documentan diferentes detalles sobre la implementación, y las pruebas realizadas para verificar que la aplicación funciona correctamente.
- Capítulo 5, conclusiones, donde se presentan algunas conclusiones que se dieron tras la ejecución del proyecto.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Para el desarrollo del proyecto se ha decidido utilizar la metodología ágil SCRUM, ya que esta metodología está estructurada para ayudar a los equipos de trabajo a adaptarse de forma natural a los cambios y a los requisitos cambiantes de los usuarios para que el equipo pueda aprender y mejorar constantemente [7].

Figura 2.1: Ciclo de vida SCRUM [1].

Como se puede observar en la figura 2.1, esta metodología consta de diferentes fases que se explican a continuación:

- Requerimientos o backlog. Es la lista principal de tareas del proyecto. Se trata de una lista dinámica de tareas, mejoras y correcciones que actúa como la entrada para las tareas del sprint. Se trata, básicamente, de la lista de cosas por hacer del proyecto.
- Tareas o backlog del sprint. Se trata de la lista de elementos, historias de usuario o

correcciones de errores, seleccionadas por el equipo de desarrollo, para su implementación en el ciclo actual de sprint.

- Iteración o sprint. Es el periodo real en que el equipo trabaja de forma conjunta para finalizar un incremento. La duración de cada sprint suele ser de dos semanas aunque suele cambiar según el incremento que se quiere alcanzar.
- Incremento u objetivo del sprint. Es el producto final utilizable de un sprint.

Cabe añadir que en el ciclo de vida SCRUM se realizan reuniones diarias con el objetivo de que todos los miembros del equipo estén en sintonía, se adecuen al objetivo del sprint y dispongan de un plan para las próximas horas. La forma habitual es que cada miembro responda tres preguntas clave:

- ¿Qué hice ayer?
- ¿Qué tengo pensado hacer hoy?
- ¿He tenido algún problema?

En relación al proyecto, se han organizado dichas reuniones diarias, en las que se contaba con la presencia de los 3 desarrolladores del proyecto y el supervisor asignado de la empresa. Estas dailys sucedían a las 13:30 y comentábamos que habíamos hecho durante el día, si ya se había terminado la tarea o aún faltaba por hacer, y dedicábamos un tiempo a organizar las tareas. Se dedicaban normalmente 30 minutos porque siempre solía haber algún que otro problema o duda, además de que había muchas tareas que se iban desarrollando a la vez que se iban detallando distintos requisitos y había que hacer reuniones bastante más extensas.

Por último, el equipo de SCRUM debe componerse de los tres roles que se describen a continuación:

- Product owner. Es el encargado de decidir la funcionalidad del producto, priorizar las tareas y decidir qué tareas se incluyen en cada backlog del sprint. Asimismo, es importante que sea una única persona.
- Scrum manager. Es el encargado de motivar y coordinar al equipo y es el responsable de detectar los problemas que pueden surgir durante los sprints.
- Team scum. Son un equipo multidisciplinar encargados de desarrollar el producto.

2.2. Tecnologías usadas

2.2.1. Herramientas para el desarrollo

Primero de todo, se ha utilizado el entorno de desarrollo IntelliJ IDEA, para facilitar la implementación del código de la aplicación, tanto para el back-end como para el front-end.

Como muchas aplicaciones, se requería el almacenamiento de distintos tipos de datos, por eso se ha hecho uso de MySQL, un sistema de gestión de base de datos relacional [8], para el diseño de la base de datos se ha usado la herramienta MySQL Workbench, y para manejar la administración de la base de datos se ha utilizado phpMyAdmin.

Por otro lado, se ha hecho uso de GitLab, una plataforma web de control de versiones basada en Git [9], donde se han ido juntando las distintas partes de código que se iban desarrollando. Se iban creando y mergeando las ramas que hacían referencia a las tareas a desarrollar, y una rama principal llamada desarrollo donde se encuentra el código de la apliación que está desplegado.

Para la parte front-end del proyecto se ha utilizado el framework Angular, en la versión 13, para realizar las ventanas de la aplicación web. Además, para el diseño del prototipo de dichas ventanas se ha utilizado el programa de escritorio Balsamiq.

2.2.2. Herramientas de gestión y comunicación

Para facilitar la gestión del proyecto, se ha utilizado la herramienta Redmine, una plataforma web que facilita la gestión, planificación, seguimiento y organización de proyectos [10]. Dispone de un listado de tareas para visualizar las tareas abiertas, en progreso, desarrolladas o cerradas, lo que facilita tener una visión general del estado del proyecto en todo momento. Además se utiliza para imputar las horas diarias que transcurren realizando las tareas.

2.3. Estimación de recursos y costes del proyecto

2.3.1. Recursos totales

Esta sección se va a dividir en comentar los recursos humanos, software y hardware que se han utilizado en el proyecto.

En primer lugar, los recursos humanos. En el caso del proyecto, se ha contado con tres desarrolladores y un supervisor. Teniendo en cuenta la experiencia de los integrantes del equipo, se puede asumir que dos desarrolladores son programadores junior y otro es programador senior, y que el supervisor actúa como el jefe del proyecto.

En segundo lugar, los recursos software. Casi todo el software que se ha utilizado ha sido gratuito, menos la licencia para usar el IDE de IntelliJ IDEA.

Por último, los recursos hardware. Todos los recursos hardware que se han utilizado durante el desarrollo del proyecto han sido proporcionados por la empresa, como el ordenador y todos sus componentes externos, y el servidor.

2.3.2. Costes totales

El principal coste del proyecto, son los sueldos de los tres desarrolladores y del jefe del proyecto. Estos salarios se han calculado utilizando de referencia las horas diarias aproximadas que cada uno de los componentes del equipo ha realizado mientras se desarrollaba el proyecto. En la tabla 2.1 se muestra una tabla detallada de estos sueldos.

Rol	Horas diarias	Coste la hora (€)	Coste mensual (€)
Programador junior	5	25	2500
Programador junior	8	25	4000
Programador senior	8	35	5600
Jefe del proyecto	0.5	35	350

Tabla 2.1: Tabla de costes de los recursos humanos

Todos los salarios mostrados en la tabla 2.1 se han obtenido contando ya con los costes de contratación y los costes adicionales que constituyen un 20 % adicional cada uno.

A parte de los costes de recursos humanos se han de tener en cuenta los costes de software y hardware. Como se ha mencionado anteriormente, en los costes de software solo cuenta el precio de la licencia para poder utilizar el entorno de desarrollo IntelliJ IDEA, y en los costes de hardware contamos con los costes del ordenador, periféricos y servidor proporcionados por la empresa. Estos costes se pueden observar en las tablas 2.2 y 2.3.

Programa	Coste de licencia (€/mes)	Coste total (€)
IntelliJ IDEA	60,38	181,14

Tabla 2.2: Tabla de costes software

Componente	Cantidad	Coste aproximado (€)
Ordenador	1	181,14
Monitor	2	250
Servidor	1	1200
Ratón	1	50
Teclado	1	80

Tabla 2.3: Tabla de costes hardware

Tras estos desgloses, podemos calcular los costes totales del proyecto, los costes de los recursos humanos se han obtenido a partir de la fila de los costes mensuales de la tabla 2.3 por los 6 meses de duración del proyecto. Dichos datos se pueden contemplar en la tabla 2.4, en la que podemos observar que el coste total del proyecto serán 76642,28 €.

Tipo de coste	Coste (€)
Recursos humanos	74700
Software	181,14
Hardware	1761,14

Tabla 2.4: Tabla de costes totales en el desarrollo

2.4. Identificación y gestión de riesgos

Dada la cantidad de miembros del equipo y la nula experiencia en la empresa, ya que los tres éramos nuevos, muchos de los riesgos presentes en el desarrollo del proyecto están relacionados con conflictos entre miembros del equipo y la poca veteranía de estos en las tecnologías usadas en la empresa.

De esta forma, en la tabla 2.5 se listan los posibles riesgos que se han tenido en cuenta y en la tabla 2.6 se detallan los planes de contingencia para cada uno de los riesgos.

A lo largo del proyecto, se dieron dos casos leves del riesgo R05, ya que el desarrollador senior decidió dos veces reiniciar el proyecto desde cero, por suerte para la parte del front se pudieron reutilizar bastantes cosas, estos casos se solucionaron en 10 horas cada uno, unos 4 dias en total, lo que produjo retrasos en el mismo sprint y en el próximo. Por suerte, no han surgido nuevos riesgos que no se hubiesen planteado al principio.

ID	Descripción
R00	Diseño de interfaces incompleto o erróneo
R01	Falta de requisitos
R02	Baja en el equipo de desarrollo
R03	Falta de experiencia en las tecnologías usadas
R04	Base de datos incompleta o errónea
R05	Falta de entendimiento entre los miembros del equipo

Tabla 2.5: Tabla de riesgos

ID	Plan de contingencia
R00	Rehacer las interfaces erróneas o añadir las faltantes
R01	Hablar con los usuarios para completar la lista de requisitos
R02	Que el desarrollador trabaje desde casa, si es posible
R03	Preguntar al equipo por ayuda o materiales para aprender
R04	Modificar la base de datos para añadir lo que falta o modificar lo que esté erróneo
R05	Realizar una reunión con los miembros del equipo y el jefe de proyecto para llegar a arreglar dicha falta de entendimiento y añadir nuevas tareas que hagan falta

Tabla 2.6: Tabla de planes de contingencia de los riesgos

2.5. Planificación temporal

En este apartado se detalla una versión de la planificación temporal inicial, la cual ha sido modificada a lo largo del proyecto por los riesgos comentados anteriormente y algunos cambios que se comentan en el siguiente apartado, seguimiento del proyecto. Las tareas principales se pueden observar en el esquema de descomposición de trabajo (EDT) de la figura 2.2.

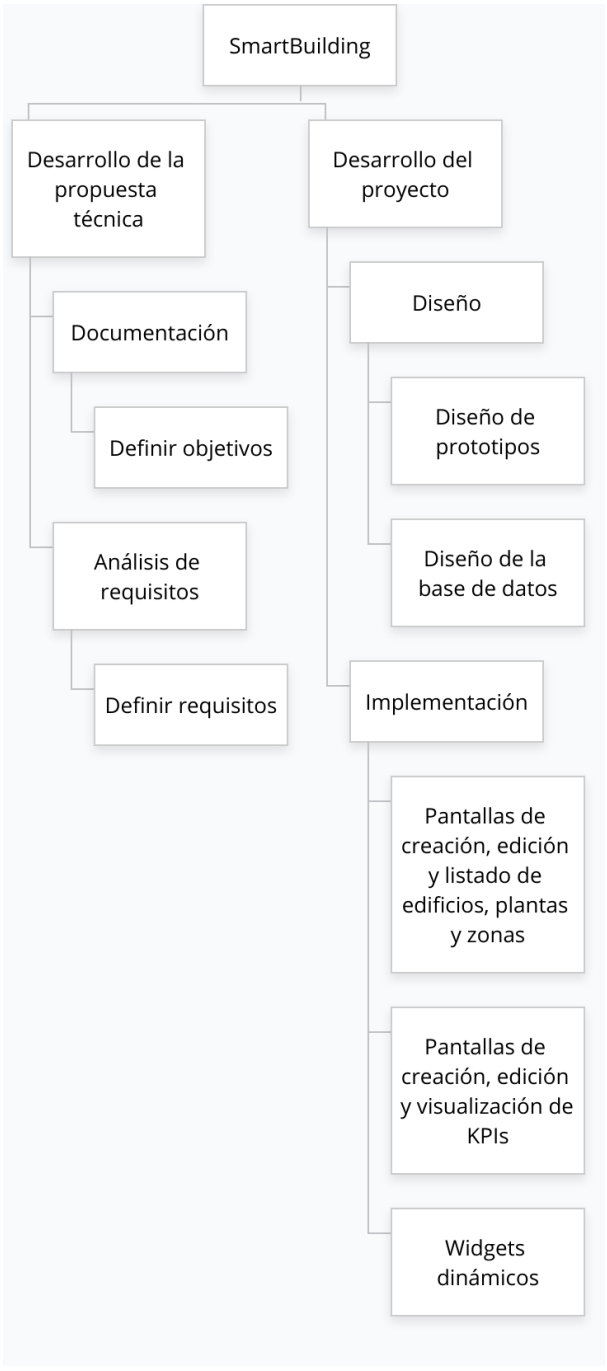

Figura 2.2: EDT del proyecto.

2.5.1. Diagrama de Gantt

En esta sección se detalla un desglose del coste temporal de cada tarea en la figura 2.3 y a continuación el diagrama de Gantt que representa en la figura 2.4 dicho desglose temporal.

	Nombre de tarea	Duración	Predecesoras
1	**Proyecto final de grado**	**64 días**	
2	**Desarrollo de la propuesta técnica**	**10 días**	
3	**Documentación**	**5 días**	
4	Leer documentación de la empresa	1 día	
5	Definir objetivos	2 días	4
6	Documentación del estado del arte	2 días	
7	**Análisis de requisitos**	**5 días**	**3**
8	Definir requisitos	3 días	
9	Definir historias de usuario	2 días	8
10	**Desarrollo del proyecto**	**54 días**	**2**
11	**Diseño**	**10 días**	
12	Diseño de prototipos	5 días	
13	Diseño de la base de datos	5 días	12
14	**Pruebas de concepto**	**10 días**	**11**
15	Selector de svg	5 días	
16	Lazy loading	4 días	15
17	**Implementación**	**26 días**	**14**
18	Gestión de edificios	4 días	
19	Gestión de plantas	3 días	18
20	Gestión de zonas	3 días	19
21	Gestión de dispositivos	3 días	20
22	Integración de Lazy loading	5 días	16
23	Widgets dinámicos	5 días	
24	**Pruebas**	**8 días**	**17**
25	Comunicar frontend y backend	8 días	

Figura 2.3: Desglose del coste temporal de las tareas del proyecto.

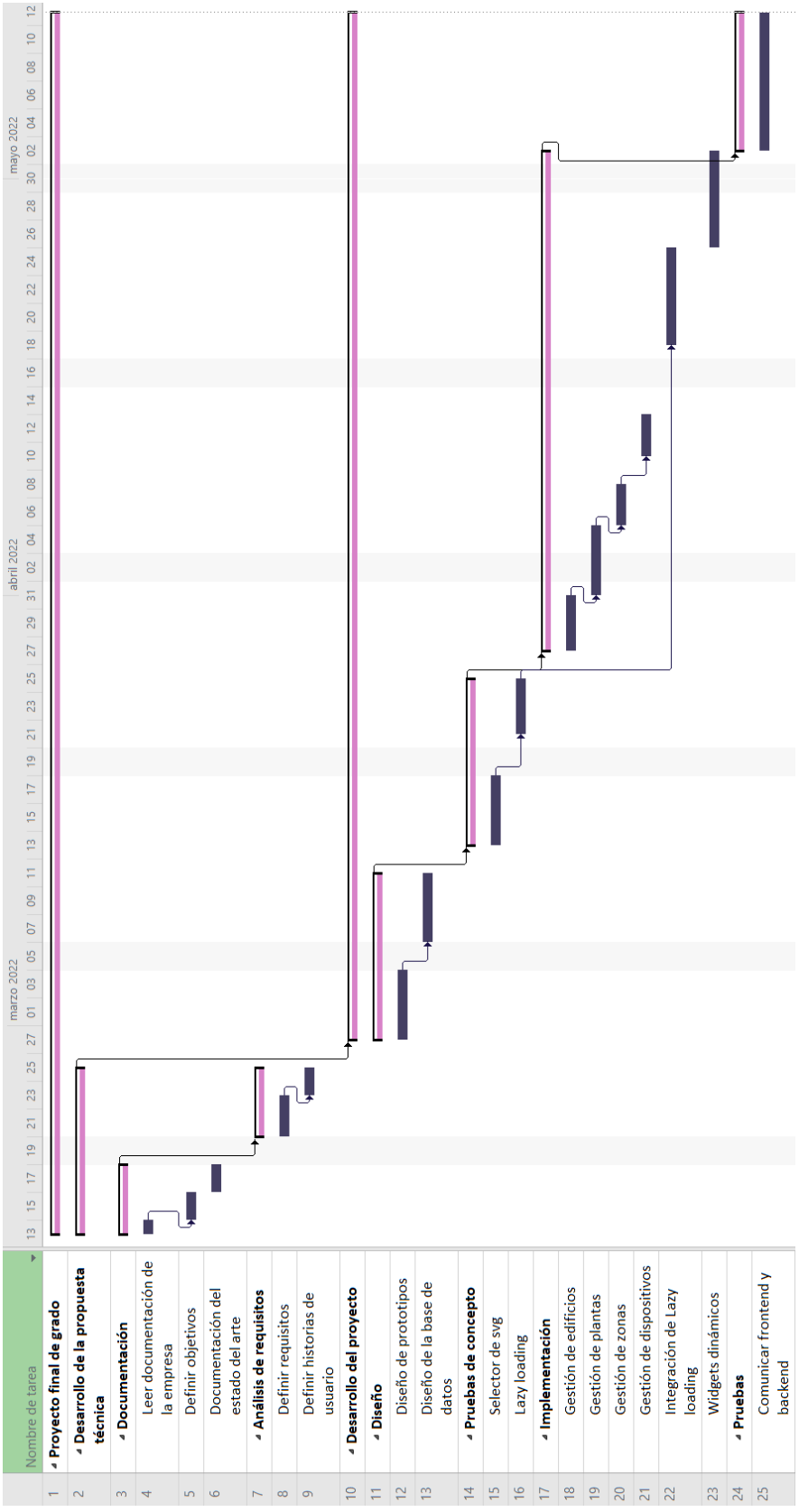

Figura 2.4: Diagrama de Gantt.

2.6. Seguimiento del proyecto

A continuación, se explica el seguimiento del proyecto, durante las distintas fases del mismo, los problemas que surgieron y los cambios que se produjeron respecto a estos. Cabe añadir que el proyecto se empezó a dividir en sprints a falta de tres semanas de terminar la estancia en prácticas, pero se pueden observar las distintas etapas a lo largo del desarrollo. Además, a partir del primer mes de prácticas se empezaron a tener reuniones diarias para realizar un seguimiento de las tareas y del proyecto.

La planificación del proyecto cambió a partir del primer mes de prácticas, surgieron varios problemas, explicados en la sección gestión de riesgos, que obligaron cambiar los objetivos, por lo que se eliminó del alcance el desarrollar todo lo relacionado con los KPIs, y se añadió el realizar pruebas completass y a conciencia del correcto funcionamiento de la apliación.

2.6.1. Fase de documentación y análisis de requisitos

Durante las primeras dos semanas las horas se dedicaron a la planificación del proyecto, leer la documentación proporcionada por la empresa para seguir los estándares de arquitectura que usan todos sus verticales, definir los objetivos necesarios y realizar un análisis de los requisitos necesarios para llevar a cabo el proyecto.

Durante esas semanas se mantuvieron algunas reuniones con mi supervisor para aclarar los requisitos y los distintos KPIs que se podrían llegar a incluir en el proyecto.

2.6.2. Fase de diseño

Durante las siguientes dos semanas después de la fase anterior, se dedicaron las horas a diseñar los distintos prototipos de las interfaces del sistema y a diseñar el modelo de datos a utilizar en el proyecto.

Para el diseño de prototipos, se mantuvieron reuniones casi a diario con mi supervisor para comprobar que se seguía con el principal diseño de los verticales ya creados, y que el diseño se adecuaba a lo que el cliente buscaba.

Para el diseño del modelo de datos, se mantuvieron tres reuniones, una al principio para comentar y aclarar las tablas necesarias, la segunda para comprobar las distintas relaciones entre las tablas y la última para revisar los cambios realizados y dar por terminado el diseño.

Esta fase de diseño concluyó con una reunión con los clientes para mostrar los diseños tanto de los prototipos como del modelo de datos.

2.6.3. Inicio de la implementación

Durante la fase inicial de la implementación, se incorporaron dos personas en el desarrollo del proyecto. Se realizó la implementación de las distintas pantallas de formularios de creación y edición de edificios, plantas y zonas, así como los listados de estos. Al inicio de la segunda semana un compañero decidió cambiar la estructura del proyecto y reiniciarlo de cero.

Durante las horas que el compañero estuvo cambiando la estructura, se decidió realizar unas pruebas de concepto. Se realizó una primera prueba de concepto para integrar el sistema de buenas prácticas Lazy Loading, y ayudé a mi otra compañera a realizar una prueba de concepto para comprobar si se podrían seleccionar elementos dentro de un SVG. Este cambio de estructura produjo cambios y retrasos en la planificación inicial, ya que se añadieron las nuevas tareas de las pruebas de concepto y una vez finalizados los cambios se tuvieron que volver a implementar las pantallas que ya se habían implementado anteriormente.

2.6.4. Fase de la integración de Lazy Loading

Una vez realizada la prueba de concepto para integrar Lazy Loading en el proyecto, comprobar que se podía integrar y que nos dieran el visto bueno, se llevó a cabo dicha integración en el desarrollo del proyecto. Además de dicha integración se siguieron implementando las pantallas siguiendo los prototipos que se habían diseñado. En paralelo, mi compañera estuvo realizando una prueba de concepto para incluir widgets dinámicos en los que poder representar los distintos KPIs y los listados de dispositivos dependiendo de la ubicación.

2.6.5. Fase de creación de widgets dinámicos

Cuando ya se tenía bastante avanzado el desarrollo del proyecto y se iba a realizar la integración de la prueba de concepto de los widgets dinámicos, el compañero que anteriormente decidió reiniciar el proyecto, lo volvió a hacer por problemas en el back ya que se decidió hacer un cambio en la base de datos para que se adecuara a los verticales y conllevó a una refactorización completa del front-end y back-end. Este cambio volvió a producir retrasos en la planificación inicial ya que se tuvo que volver a implementar todos los cambios que ya se tenían en el front-end.

Una vez se volvió a tener el front-end como estaba, se pudo iniciar el proceso de creación de los widgets dinámicos encargados de mostrar el listado de dispositivos y el widget encargado de mostrar un mapa con la ubicación del edificio.

2.6.6. Fase final

En la última semana, antes de terminar la estancia en prácticas, se decidió probar a empezar a enlazar el front-end con el back-end y solucionar posibles problemas que pudieran surgir con la conexión.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En este apartado se va a realizar un análisis de los requisitos del sistema y se exponen las características que se han implementado. Todos los requisitos han sido revisados por la empresa antes de empezar a desarrollar el proyecto y han servido como guía para no implementar funcionalidad que no fuera necesaria.

Para este análisis se han utilizado las historias de usuario, ya que son uno de los pilares fundamentales del desarrollo de proyectos realizados con metodologías ágiles. Una historia de usuario es la manera en la que un equipo de trabajo scrum puede estimar y planificar la duración de los sprints, lo que ayuda a poseer un pronóstico más preciso y una mayor agilidad en el proyecto, por lo tanto cada historia de usuario es un objetivo final visto desde la perspectiva del usuario del proyecto [11]. Tras esta breve explicación, en la tabla 3.1 se pueden contemplar las historias de usuario especificadas durante la recopilación de los requisitos requeridos para el proyecto. Como se puede ver, se van a manejar dos roles en el sistema, un rol de usuario y un rol de administrador, por facilitar el desarrollo, muchas de las historias de usuario pertenecientes al rol de usuario también pertenecen al rol de administrador.

ID	Historia de usuario
HU01	Como usuario quiero poder ver en un listado los edificios que me pertenecen
HU02	Como administrador quiero poder ver en un listado los edificios presentes en el sistema
HU03	Como usuario y administrador quiero poder ver en un mapa la ubicación del edificio
HU04	Como usuario y administrador quiero poder ver en un listado las plantas y las zonas pertenecientes a un edificio
HU05	Como usuario y administrador quiero poder ver en un listado las zonas pertenecientes a una planta
HU06	Como usuario y administrador quiero poder ver en un listado los dispositivos pertenecientes a un edificio, una planta y una zona
HU07	Como usuario y administrador quiero poder ver un listado de los KPIs pertenecientes a un edificio, una planta y una zona
HU08	Como administrador quiero poder añadir edificios al sistema
HU09	Como administrador quiero poder editar edificios del sistema
HU10	Como administrador quiero poder añadir plantas al sistema
HU11	Como administrador quiero poder editar plantas del sistema
HU12	Como administrador quiero poder añadir zonas al sistema
HU13	Como administrador quiero poder editar zonas del sistema
HU14	Como administrador quiero poder añadir dispositivos al sistema
HU15	Como administrador quiero poder editar dispositivos del sistema
HU16	Como usuario y administrador quiero poder visualizar en gráficas las mediciones de un dispositivo del sistema

Tabla 3.1: Tabla de las historias de usuario

3.1.1. Requisitos de datos

Para poder satisfacer las historias de usuario descritas anteriormente, es necesario tener claro las entidades y atributos que se van a necesitar, es decir, los requisitos de datos. Los requisitos de datos del proyecto, se encuentran en el Anexo A, cabe añadir que dicha especificación puede llegar a tener modificaciones durante el desarrollo del proyecto.

3.2. Diseño de la arquitectura del sistema y de la base de datos

3.2.1. Arquitectura

Como se puede observar en la figura 3.1, en la empresa se cuenta con una misma arquitectura para cada uno de los verticales, así como para la creación del proyecto. La capa de servicios es la que se ha desarrollado, ya que las demás capas ya están en pleno funcionamiento en la empresa. Como se puede observar, los datos en tiempo real que se consultan en el proyecto se obtienen

de la capa de conocimiento, los datos de usuarios y dispositivos de la capa de interoperabilidad, y la autenticación de usuarios de la capa de soporte.

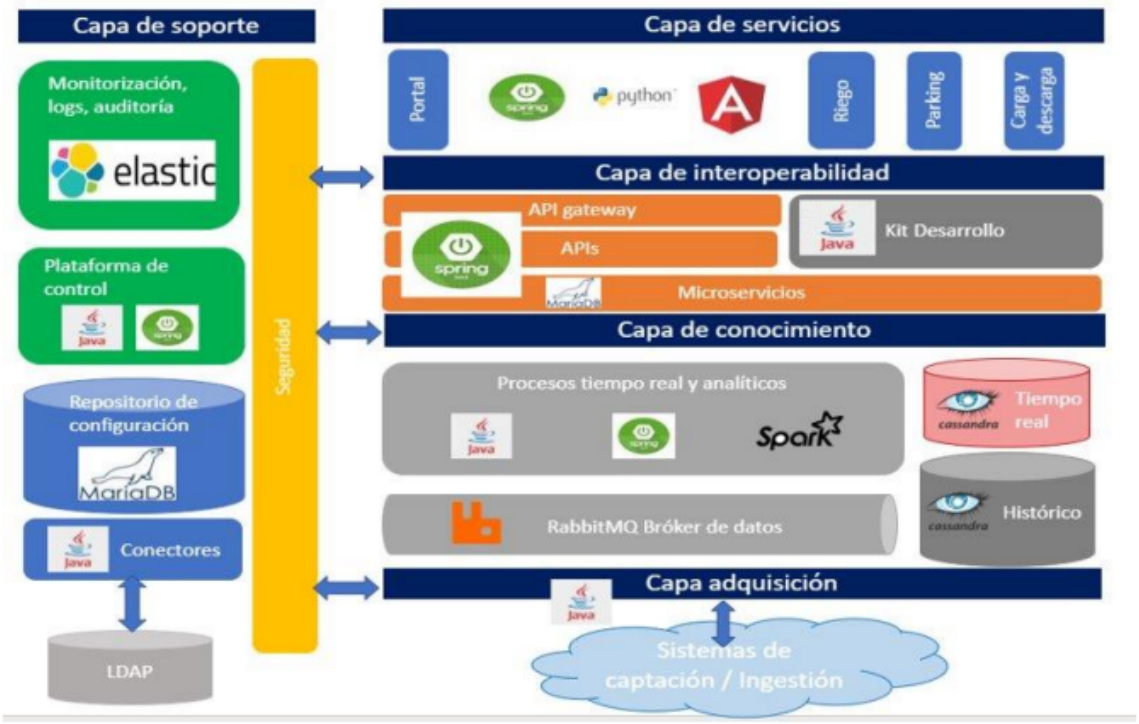

Figura 3.1: Diseño de la arquitectura (figura proporcionada por la empresa).

3.2.2. Base de datos

Al principio se realizó un primer diseño funcional de la base de datos, pero por problemas con el back-end del proyecto, y que se necesitaban ciertos cambios para que pudiera conectar con el front-end y con la API de catálago de donde se busca información relacionada con los dispositivos se decidió realizar un cambio. En la figura 3.2 se puede observar el primer diseño, los cambios realizados sobre esta base de datos no son relevantes ya que no son cambios que se hayan tratado en las horas de prácticas.

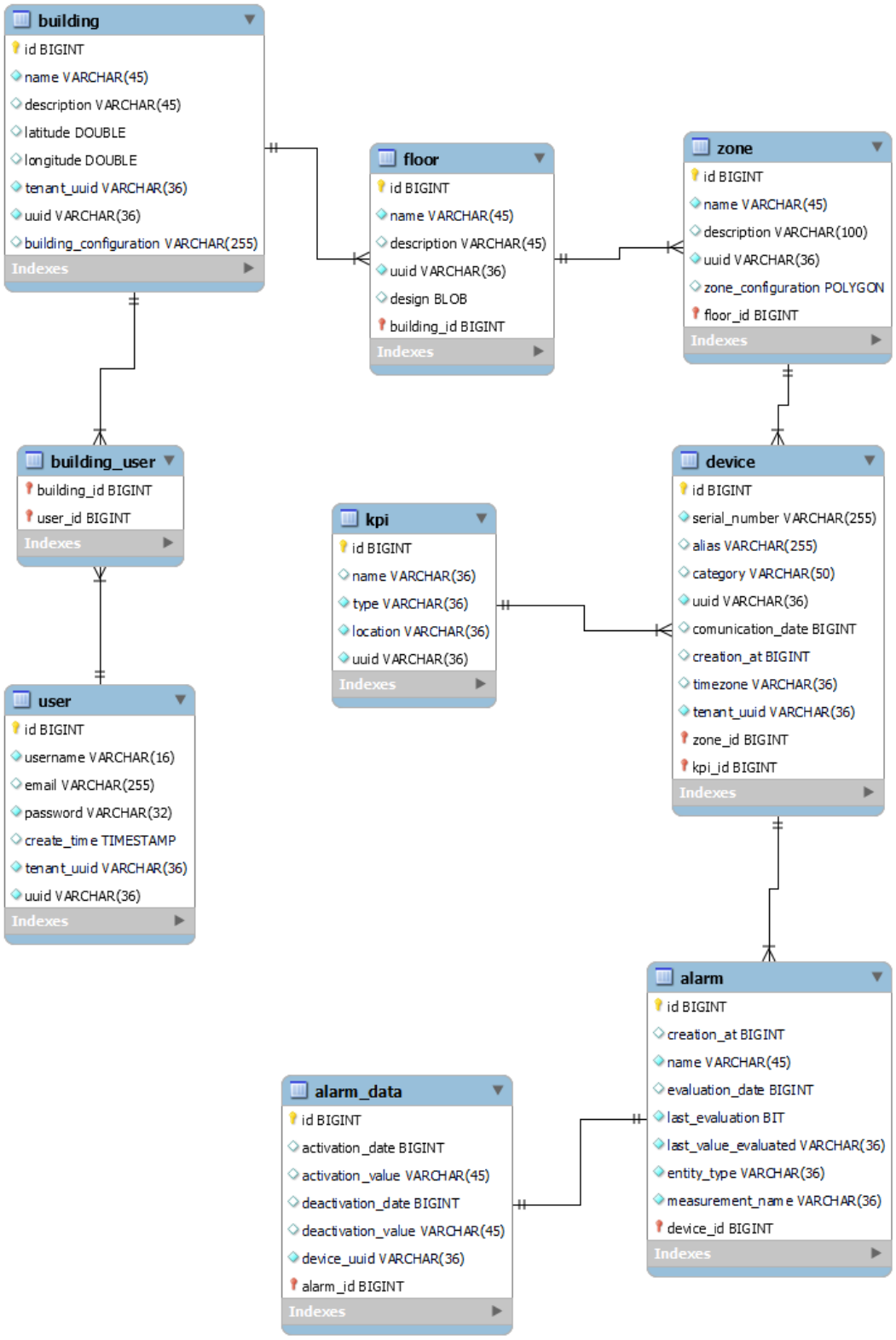

Figura 3.2: Primer diseño de la base de datos.

Los datos relacionados con los usuarios, dispositivos y alarmas no se tratan en esta aplicación, estos datos vienen de un API de la empresa llamada catálogo, y en la aplicación lo único que hacemos es guardar los relacionados con esta misma, por ejemplo si se da de alta un dispositivo, se busca desde catálogo y se guarda en nuestra base de datos.

3.3. Diseño de la interfaz

El diseño de la interfaz de usuario partió de la base de que había que seguir con la misma dinámica de diseño que las otras aplicaciones web que tienen en funcionamiento, por lo que se tuvo que seguir la idea del diseño plano o flat design y que tuviera cierto parecido a las otras aplicaciones, al menos en las partes comunes como el menú lateral, iconos y disposición de los distintos elementos en las diferentes pantallas de navegación.

El diseño plano es un estilo de diseño que pretende reducir lo máximo posible la decoración de la interfaz para facilitar la funcionalidad de una aplicación [12]. En este caso se utilizan bastantes iconos para acompañar las partes importantes, una paleta de colores reducida, etc, para conseguir dar a entender lo que se necesita con las menores palabras y llegar a tener una interfaz lo más gráfica posible.

Antes de empezar con el desarrollo del proyecto, se realizaron los diseños de los prototipos de la interfaz, estos prototipos se realizaron sin una paleta de colores ya que aún no se tenía claro. En las figuras 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10 se muestran los prototipos principales.

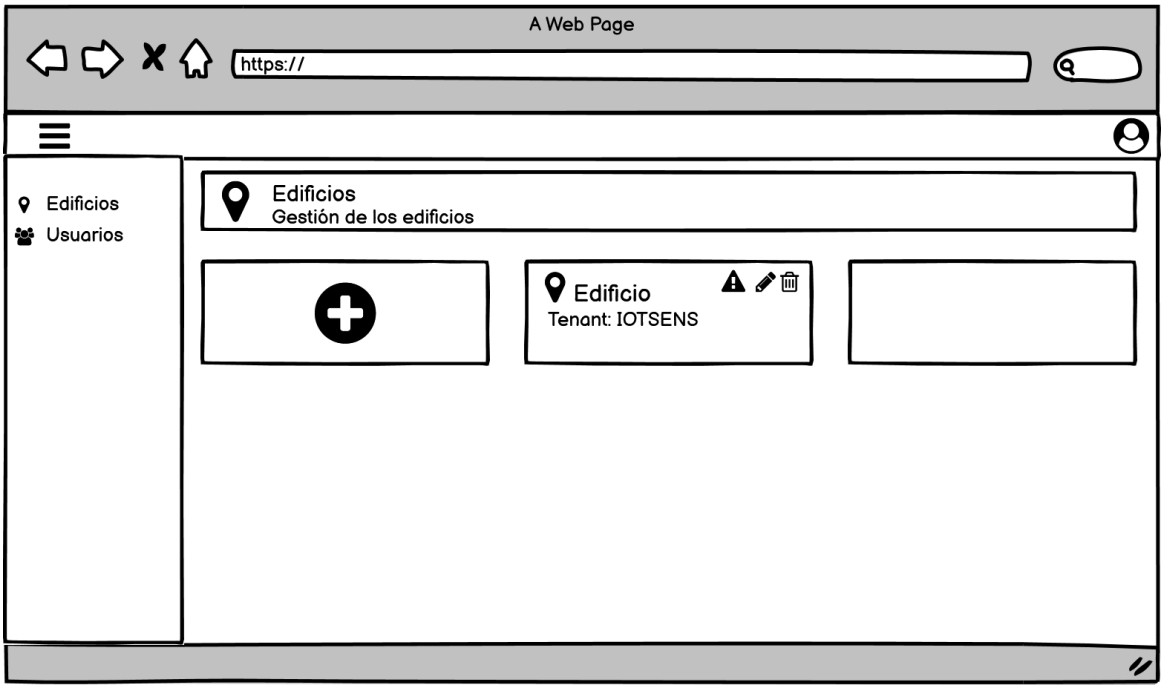

Figura 3.3: Vista del listado de edificios.

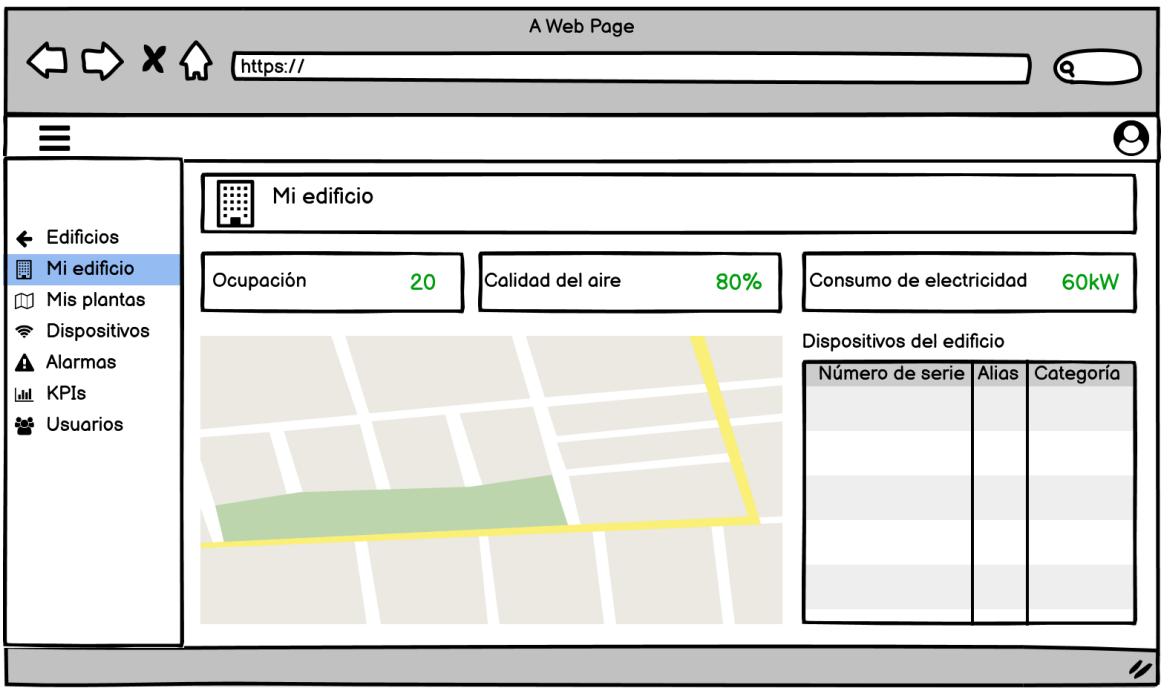

Figura 3.4: Vista del dashboard de un edificio.

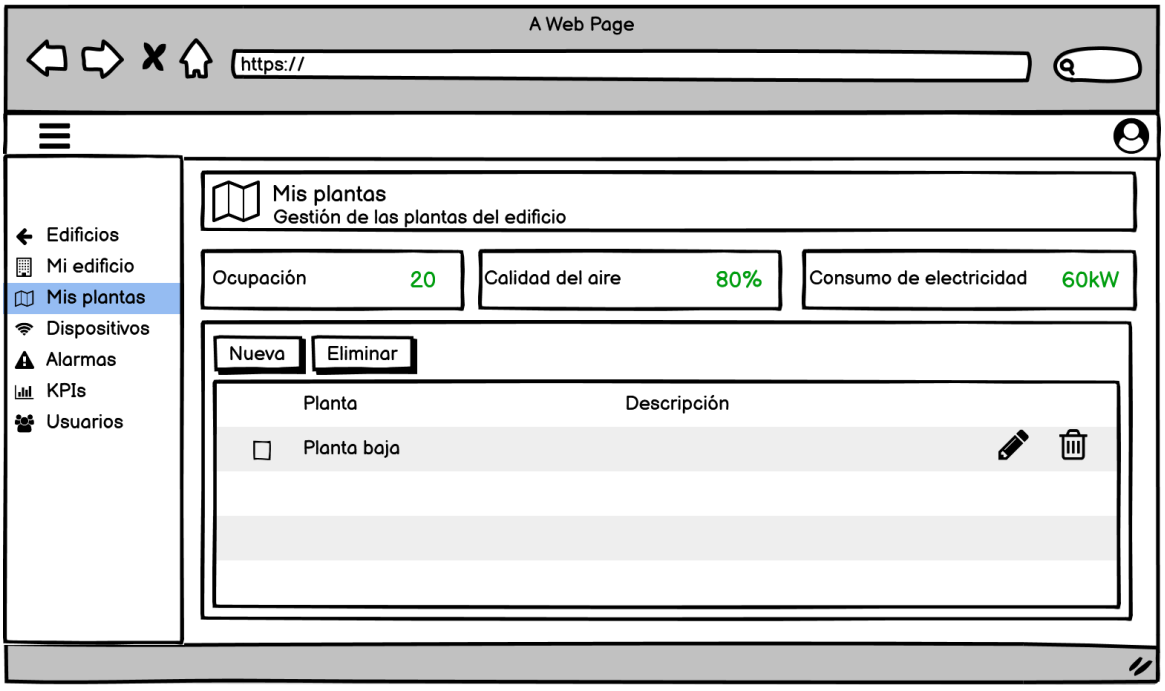

Figura 3.5: Vista del listado de plantas.

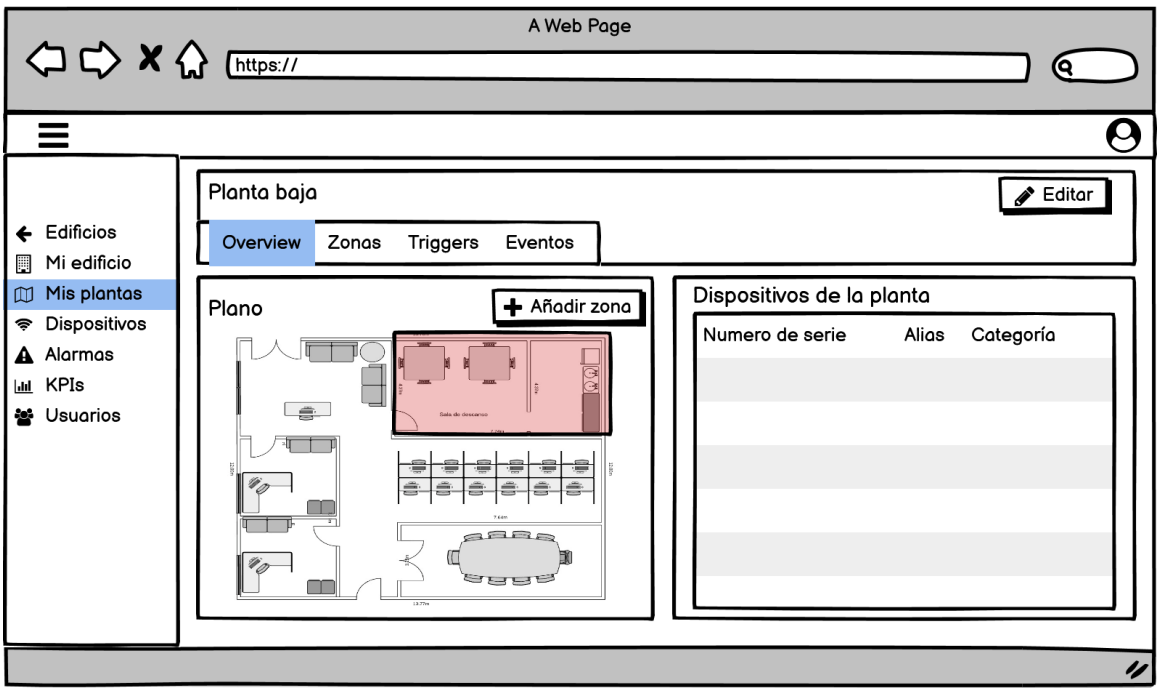

Figura 3.6: Vista del dashboard de una planta.

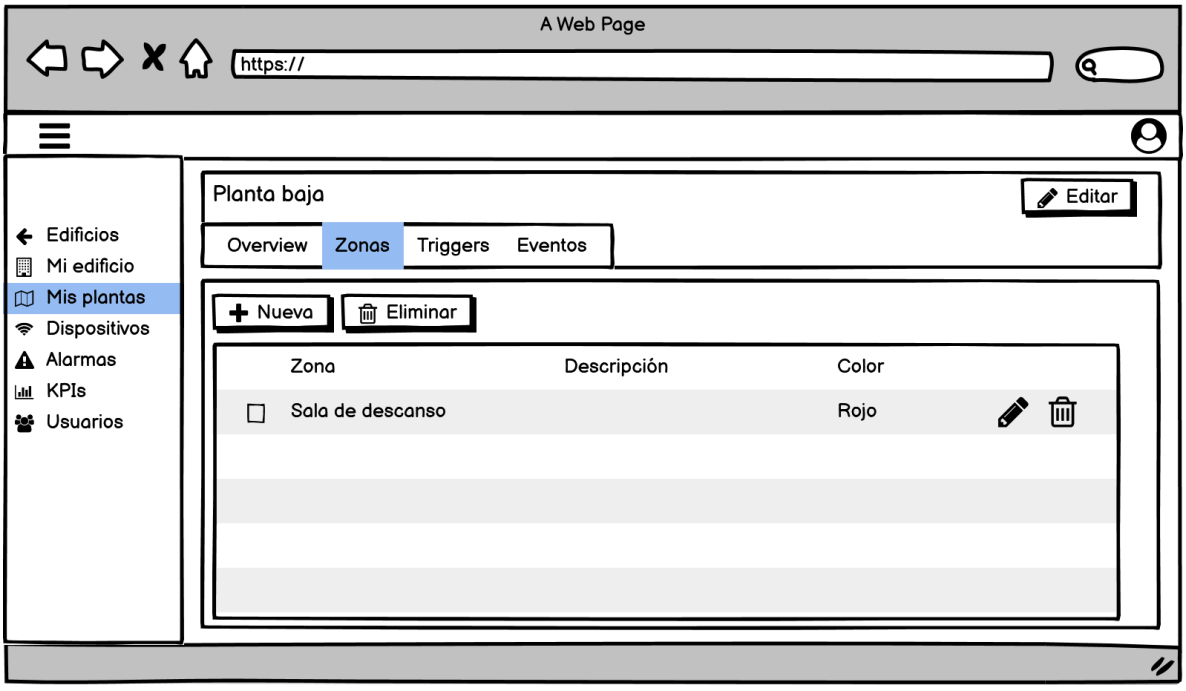

Figura 3.7: Vista del listado de zonas en una planta.

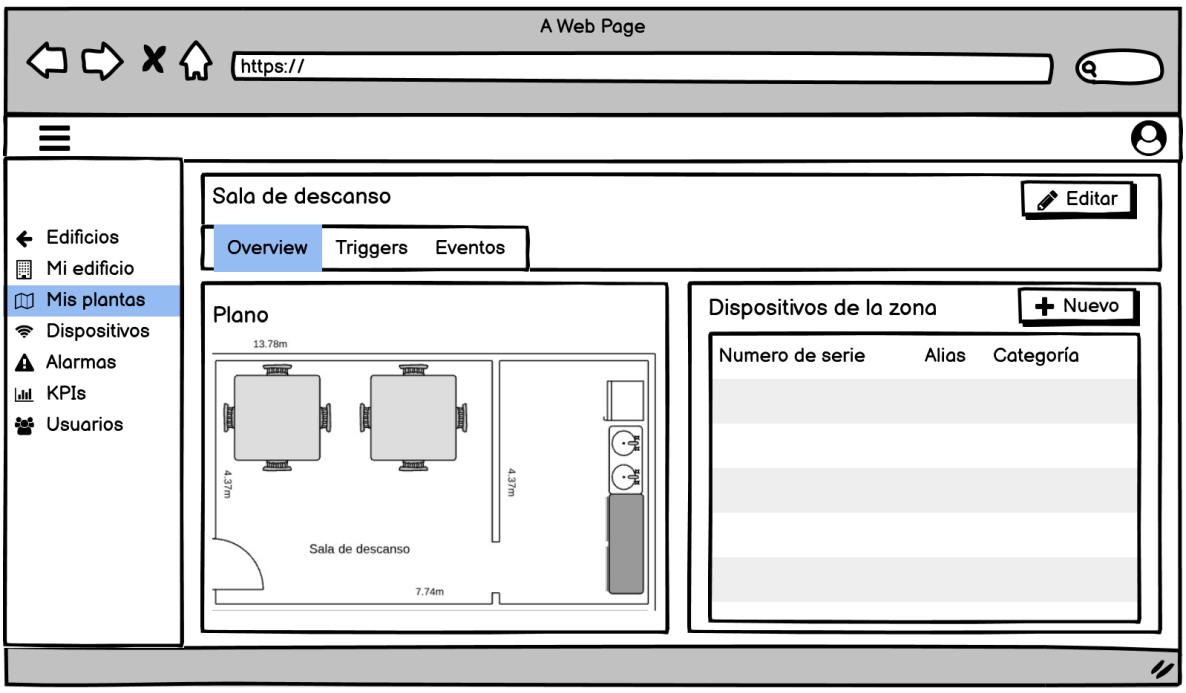

Figura 3.8: Vista del dashboard de una zona.

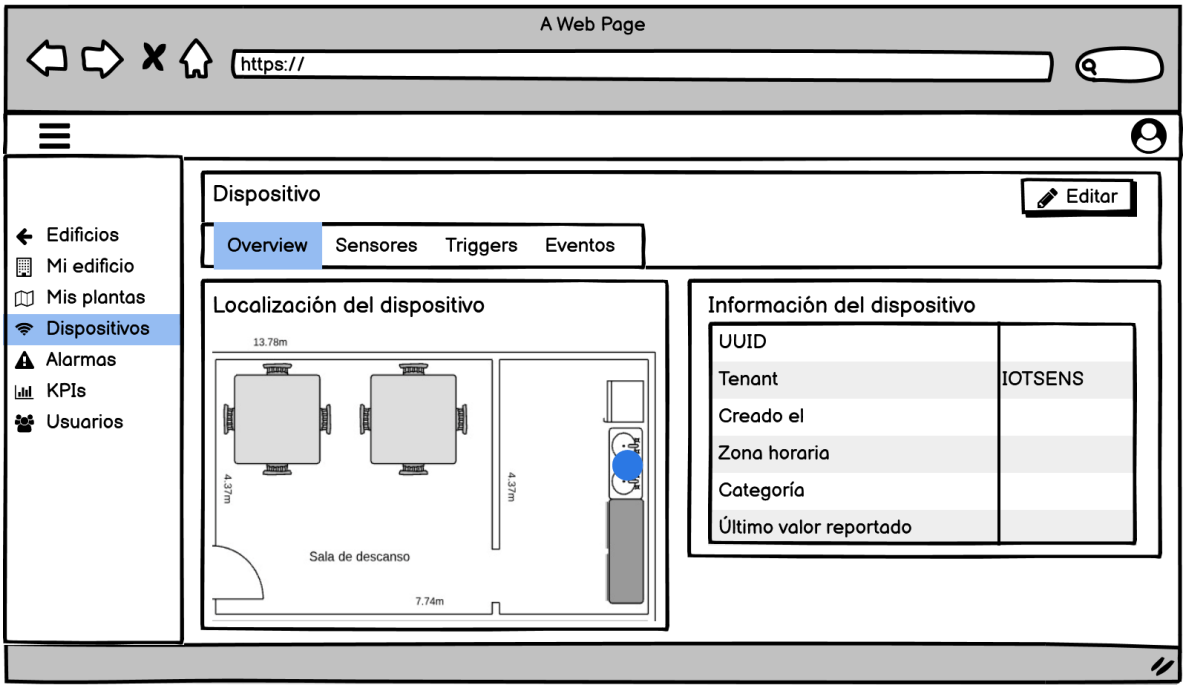

Figura 3.9: Vista del dashboard de un dispositivo.

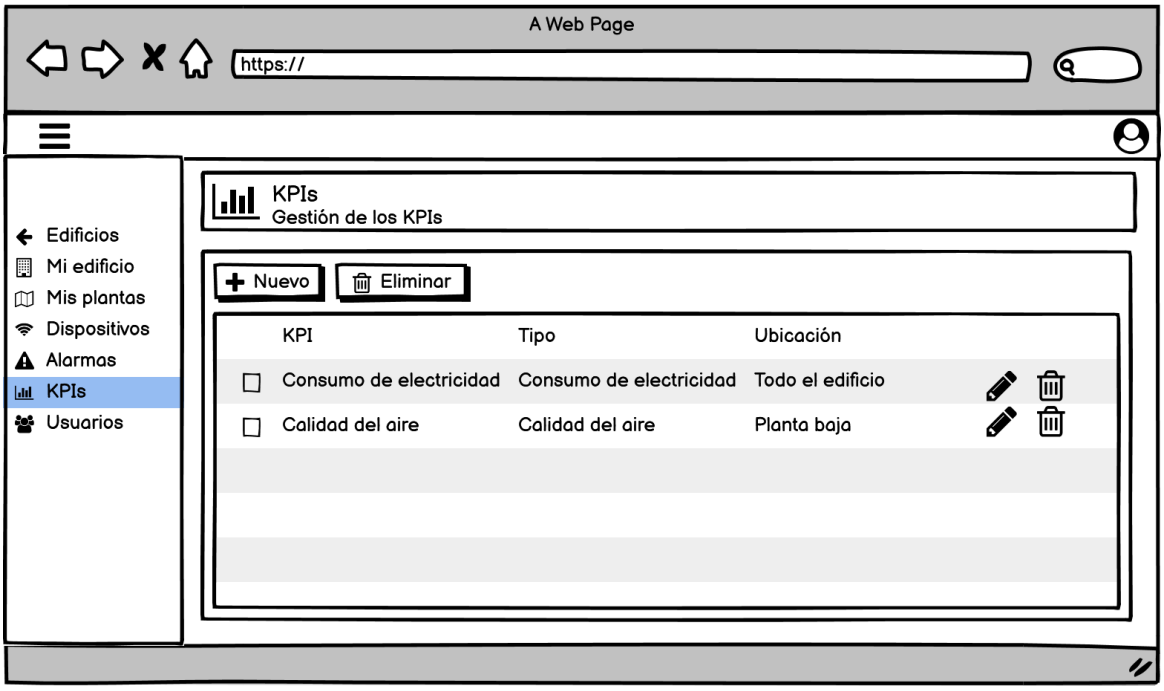

Figura 3.10: Vista del listado de KPIs.

Capítulo 4

Implementación y pruebas

4.1. Detalles de implementación

La parte del frontend se implementó con el framework Angular, además se han utilizado varias librerías externas, como gridster2 para la creación y utilización de widgets dinámicos, mapbox para la visualización de los mapas, transloco para la traducción de todos los textos mostrados en la interfaz, y primeng para los iconos y el diseño en tarjetas.

4.1.1. Estructura del frontend

En esta sección solo voy a explicar los detalles de implementación por parte del frontend, ya que de la implementación y desarrollo del backend se ha hecho cargo otro miembro del equipo.

La estructura del frontend se dividió, al principio, en cuatro carpetas principales, común, componentes, modelos y servicios; pero como se vió que se iban a necesitar componentes dinámicos como los widgets o como los formularios, ya que se necesitaba usar un mismo formulario desde distintas pantallas, se optó por seguir una estructura algo más distinta, ya que pasamos a tener común, componentes, acceso a los datos y ui y así identificar mejor donde se ubican los archivos.

Como se puede observar en la figura 4.1, el directorio app es el que contiene todos los archivos necesarios para la lógica y diseño del frontend. Los directorios de los que se está haciendo uso son common, components, data-access, i18n, shared y ui; los directorios model y services estaban en la aplicación base y contienen archivos que vamos migrando a sus carpetas correspondientes conforme se van necesitando, ya que hay archivos que aún no sabemos si se van a llegar a reutilizar. A continuación, se explica con un pequeño resumen el contenido de las carpetas importantes.

- Common: en esta carpeta se encuentran los componentes que se reutilizan en distintos sitios de la aplicación, como el componente de autenticación, validación o para logearse.

- Components: en esta carpeta se encuentran los componentes propios de la aplicación que se reutilizan para facilitar la lógica y el diseño, como los formularios dinámicos, los widgets, la barra lateral del menú, etc.
- Data-access: en esta carpeta se ubican por carpetas según su identidad, una carpeta domain con los modelos value-objects del proyecto y una carpeta services con los ficheros de los endpoints que se comunican con el backend.
- i18n: en esta carpeta se encuentra el fichero necesario para la traducción a varios idiomas, por el momento la aplicación cuenta con español e inglés.
- Ui: en esta carpeta se ubican por carpetas según su identidad, las interfaces de usuario que se muestran en la aplicación web, como pueden ser los listados, las llamadas a los formularios, o las llamadas a los dashboards.

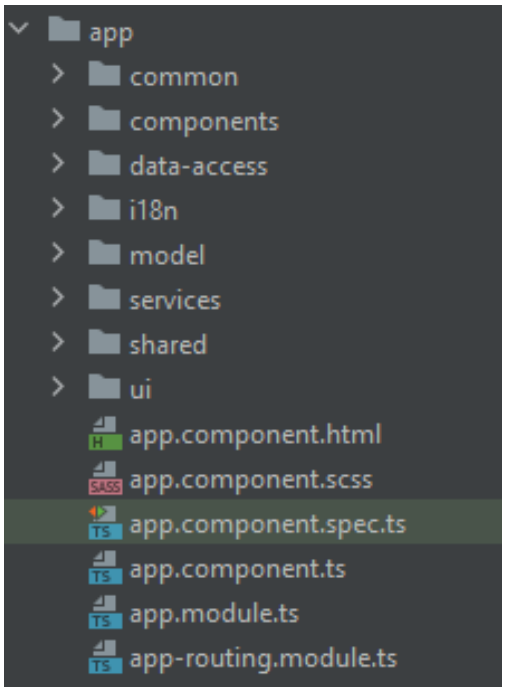

Figura 4.1: Estructura del frontend de la aplicación.

4.1.2. Integración del patrón de diseño Lazy Loading

Lazy Loading es un patrón de diseño contenido en las buenas prácticas de programación, con el que se consigue mejorar el rendimiento de la aplicación y un ahorro de recursos, ya que en vez de cargar todos los datos al principio a la vez, se van cargando a medida que el usuario los necesita [13].

Para integrarlo, primero se realizó una prueba de concepto para llegar a entender cómo funcionaba y luego poder realizarlo en el proyecto. Para implementarlo se han creado los componentes module.ts y routing.module.ts en cada componente del que vayamos a necesitar una

carga diferida, para así poder especificar las rutas hijas que contendrán, o no, otras rutas especificadas con Lazy Loading. Como se puede observar en la figura 4.1, hay un componente llamado app-routing.module.ts, que es el componente padre de la aplicación, de donde parte la ruta principal que utiliza Lazy Loading, ya que el componente del menú siempre va a estar presente (véase figura 4.2).

```
const routes: Routes = [
  { path: 'not-found', component: NotFoundComponent },
  { path: 'not-authorized', component: NotAuthorizedComponent },
  { path: '', loadChildren: () => import('./ui/ui.module').then(m => m.UiModule) },
  { path: '**', redirectTo: '/not-found' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes, config: {
    paramsInheritanceStrategy: 'always'
  })],
  providers: [AuthGuard],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Figura 4.2: Componente app-routing.module.ts

Para mostrar bien la implementación de este patrón de diseño, se puede ver en la figura 4.3 el componente ui-routing.module.ts que se utiliza al dirigirse a la ruta vacía de la figura 4.2. Donde se pueden ver las rutas principales de la aplicación, ya que se tiene un menú con edificios, usuarios, reportes y KPIs, y un submenú que parte de edificios.

```
const routes: Routes = [{
  path: '', component: LayoutComponent, children: [
    { path: '', redirectTo: 'buildings', pathMatch: 'full' },
    { path: 'buildings', loadChildren: () => import('./buildings/buildings.module').then(m => m.BuildingsModule) },
    { path: 'users', canActivate: [AuthGuard], data: ADMIN,
      loadChildren: () => import('./users/users.module').then(m => m.UsersModule) },
    { path: 'reports', canActivate: [AuthGuard], data: ADMIN,
      loadChildren: () => import('./reports/reports.module').then(m => m.ReportsModule) },
    { path: 'kpis', canActivate: [AuthGuard], data: VIEWER,
      loadChildren: () => import('./kpis/kpis.module').then(m => m.KpisModule) },
    { path: '**', redirectTo: '/not-found' },
  ]
}];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class UiRoutingModule { }
```

Figura 4.3: Componente ui-routing.module.ts

Antes también se ha mencionado que se crea el componente module.ts junto al routing.module.ts, y esto es debido a que cada grupo de componentes que agrupan hacen uso de componentes y módulos distintos, por lo que hay que importarlos o declararlos en el componente module.ts. Como se puede ver en la figura 4.4 se declaran los componentes/imports que se van a cargar o utilizar en ese momento.

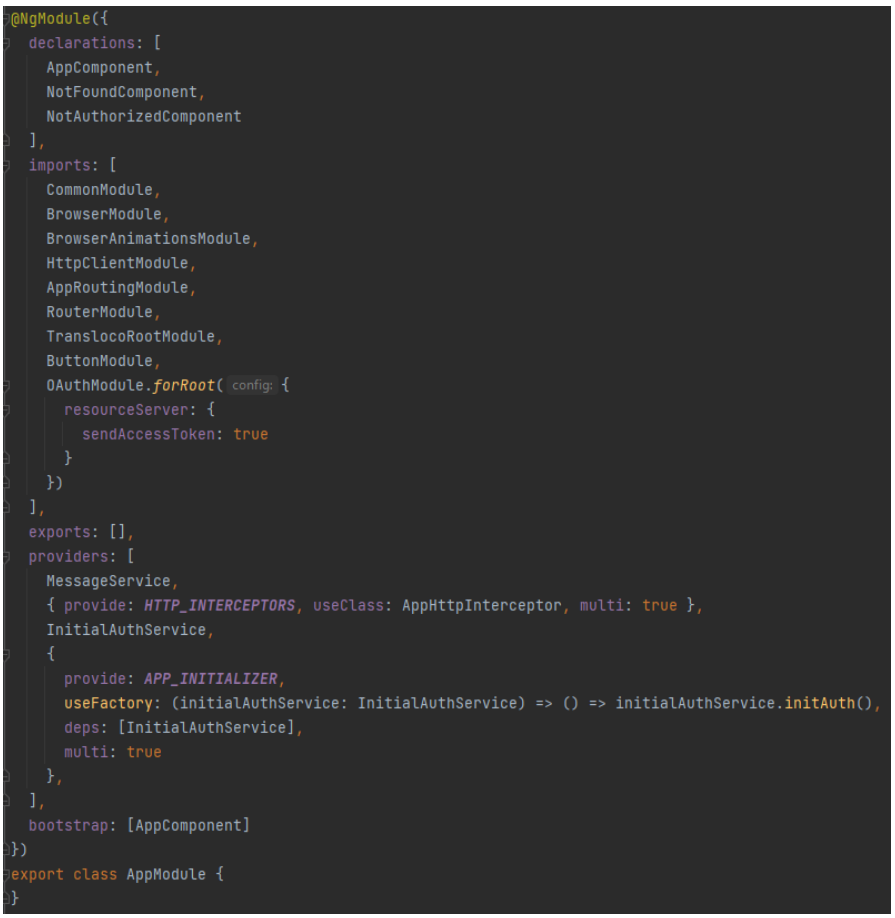

Figura 4.4: Componente app.module.ts

4.1.3. Widgets dinámicos

A lo largo de la realización del proyecto se decidió que para mostrar los KPIs, mapas, planos o dispositivos en los dashboards tanto a nivel de edificio, de planta y de zona, se harían uso de widgets dinámicos, ya que aportan cierto grado de elasticidad con el contenido y confort de cara al usuario final. Para estos widgets se ha hecho uso de la librería gridster2, ya que aporta muchos componentes modificables para el front, ya que se pueden establecer unas medidas mínimas y máximas para un determinado widget, y que el usuario pueda redimensionarlo a su gusto dentro de esos límites; también cuenta con una cuadrícula que se puede hacer más grande o pequeña para que los widgets se puedan mover y recolocar al gusto.

La prueba de concepto de los widgets dinámicos la realizó una compañera, y una vez comprobado que se podía integrar en la aplicación y de realizar un widget que muestra la ubicación del edificio en un mapa, pude realizar la fase de creación de un widget que muestra los dispositivos existentes en ese edificio. En la figura 4.5 se puede observar el esquema del componente widgets ubicado en el directorio components.

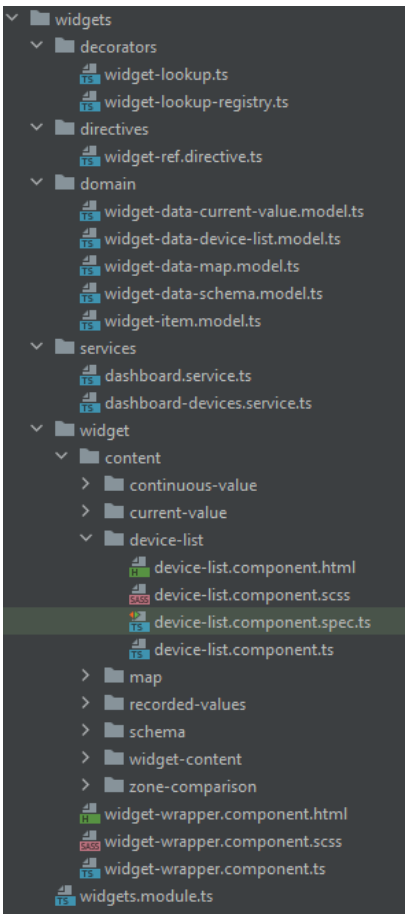

Figura 4.5: Estructura del componente widgets

El widget creado, en sí es una simple tabla con los parámetros acordados que se van a mostrar dependiendo de si se pide desde un edificio, una planta o una zona, ya que desde el edificio se va a querer mostrar la planta y la zona en la que está cada dispositivo, desde la planta se va querer mostrar la zona en la que está cada uno y desde la zona no hará falta mostrar ningún parámetro de ubicación relativa al edificio.

Además, se ha creado para que cada dispositivo mostrado sea un objeto observable [14], para que cuando se haga click sobre uno, el objeto que esté suscrito pueda ver los cambios, la razón fue para suscribir el widget del mapa para que al hacer click sobre un dispositivo el mapa mostrara la ubicación del mismo [15]. Para eso se ha hecho uso de un tipo especial de observable, el BehaviourSubject, que permite que cada vez que un nuevo componente se suscribe al observable se recibe el valor actual. Para ello, se ha creado un servicio llamado dashboard-devices.service.ts, donde se establecen los observables y se recuperan los dispositivos (véase figura 4.6).

```
get notifications$(): Observable<Device> {
  return this.device.asObservable();
}

setCurrentDevice(device: Device): void {
  this.device.next(device);
}

clearCurrentDevice(): void {
  this.device.next( value: null);
}

public getDevicesBuilding(buildingUuid: string): Device[] {
  this.deviceService.getDevicesByBuildingUuid(buildingUuid)
    .subscribe( next: response => {
      this.devices = response.devices;
    });
  return this.devices;
}

public getDevicesFloor(floorUuid: string): Device[] {
  this.floorService.getFloorDevices(floorUuid)
    .subscribe( next: response => {
      this.devices = response.devices;
    });
  return this.devices;
}

public getDevicesZone(zoneUuid: string): Device[] {
  this.zoneService.getZoneDevices(zoneUuid)
    .subscribe( next: response => {
      this.devices = response.devices;
    });
  return this.devices;
}
```

Figura 4.6: Servicio dashboard-devices.service.ts

Cada componente tipo de widget utiliza un decorador, para indicar que widget requiere de que tipo de widget, por ejemplo si se quiere mostrar un widget de tipo mapa solo hace falta indicarlo en el servicio y se busca en los tipos de widget qué componente es el que se necesita, para eso se especifica un @WidgetLookup(key) en cada tipo operativo (véase figura 4.7).

```
@WidgetLookup( key: 'DeviceList')
@Component({
  selector: 'app-device-list',
  templateUrl: './device-list.component.html',
  styleUrls: ['./device-list.component.scss'],
  providers: [{ provide: LOCALE_ID, useValue: 'es' }, DatePipe]
})
export class DeviceListComponent extends WidgetComponent implements OnInit {
```

Figura 4.7: Especificación del tipo de widget

4.2. Resultados de la implementación

A continuación, se muestran los resultados de la implementación descrita en el apartado anterior, exceptuando la implementación de Lazy Loading, que se encuentra explicada en la siguiente sección. Estos resultados engloban tanto la funcionalidad como la apariencia final de la aplicación, los cuales están presentados en las figuras de la 4.8 a la 4.13.

4.2.1. Vista del listado de edificios

La figura 4.8 muestra la vista principal de la aplicación. En este listado se ha optado por hacer uso de tarjetas en vez de una tabla, porque es más atractivo al usuario final como pantalla principal.

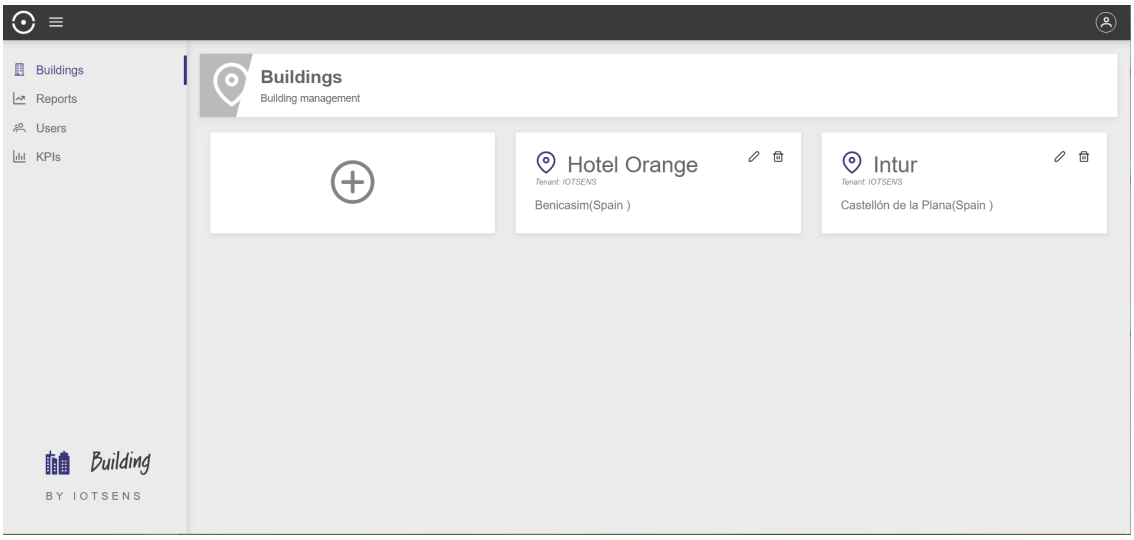

Figura 4.8: Vista del listado de edificios

4.2.2. Vista del listado de plantas, zonas y dispositivos

Las tres listas son parecidas, comparten los componentes de crear, eliminar los seleccionados, editar y borrar por separado y los filtros. Además las tres contienen paginación, por lo que solo se hace una petición al backend con el número de filas que se quiere mostrar y así no sobre cargar la aplicación, ya que si tuviéramos 50 dispositivos dados de alta en un edificio, y cada vez que se quisiera recuperar el listado, se recuperarán los 50 de golpe se acabaría bloqueando la aplicación.

A parte, los listados de zonas y dispositivos se han dinamizado, para poder reutilizarlos para mostrar las zonas y dispositivos desde una planta concreta, o para mostrar los dispositivos desde una zona concreta. Los listados son los mismos, solo que según si el origen de la url es desde un edificio o desde una planta o zona, se hace una petición distinta al backend, y así se recuperan las zonas o dispositivos según lo que se necesite sin tener que crear vistas exactamente iguales para componentes distintos.

La figura 4.9 muestra la vista del listado de plantas que hay en un edificio, indicando el número de zonas y dispositivos que contiene.

La figura 4.10 muestra la vista del listado de zonas que hay en un edificio, indicando la planta en la que se sitúa la zona y el número de dispositivos que contiene.

La figura 4.11 muestra la vista del listado de dispositivos que hay en un edificio, indicando la planta y zona en la que se ubican.

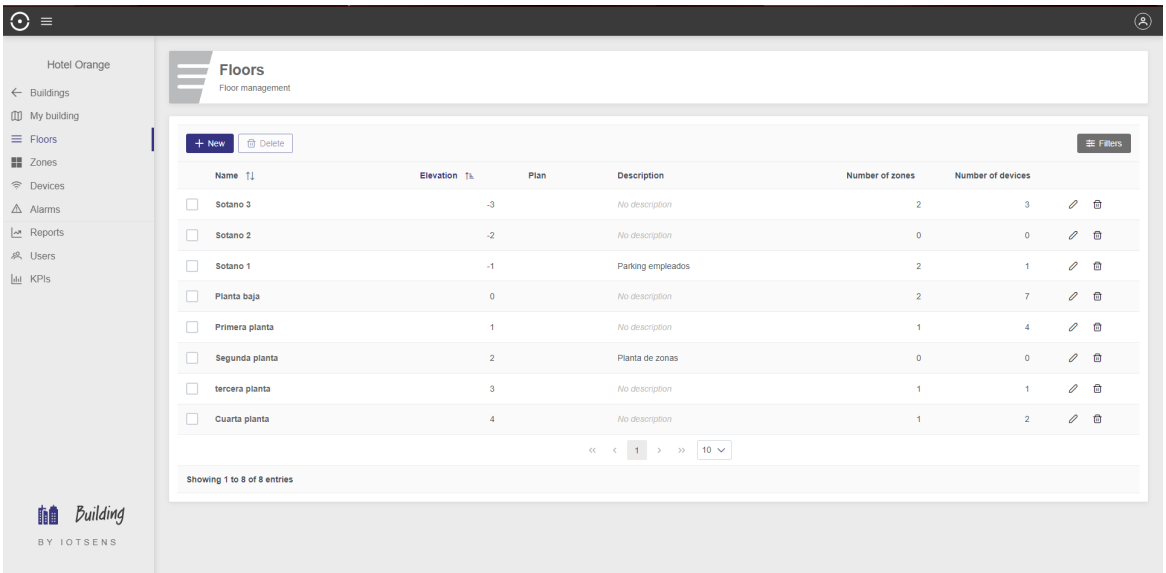

Figura 4.9: Vista del listado de plantas

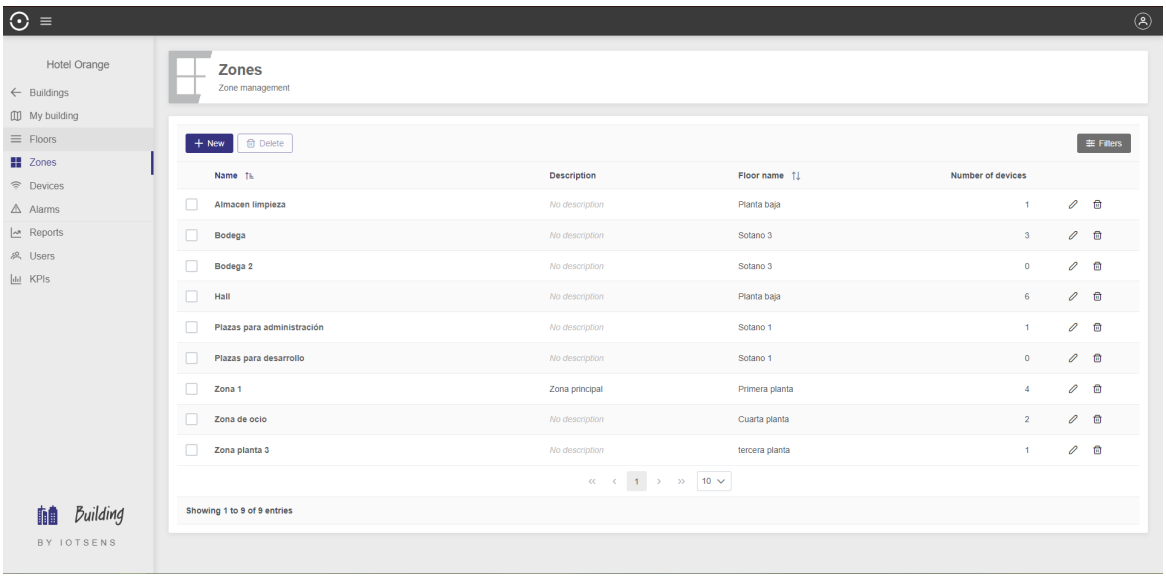

Figura 4.10: Vista del listado de zonas

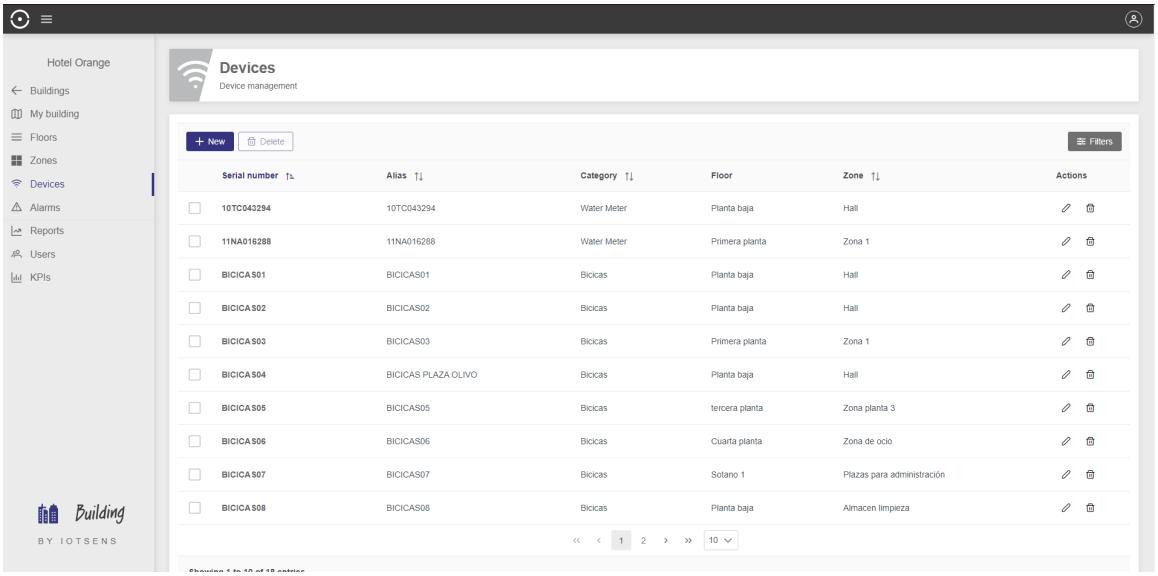

Figura 4.11: Vista del listado de dispositivos

4.2.3. Vista del dashboard de un edificio

En las figuras 4.12 y 4.13 se muestra el dashboard de un edificio, donde se pueden observar los tres tipos de widgets que se han creado, los dos pequeños situados a la derecha son widgets de valor actual, en los que se muestra el valor actual de los KPIs de temperatura exterior e interior; y el grande que es el widget que muestra la ubicación del edificio en un mapa.

Como se puede observar, todos los widgets tienen tres botones, el primero es el de bloqueo/desbloqueo para bloquear o no el redimensionamiento o reubicación del widget, el segundo es el de edición que por el momento solo deja editar el título del widget y el tercero es el de eliminar que sirve para eliminar el widget del dashboard.

Además, se han dispuesto dos botones que siempre van a aparecer al final del dashboard, para bloquear la ubicación y dimensiones de los widgets almacenados, y otro para agregar nuevos widgets, esos botones se pueden ver en la parte inferior derecha de la figura 4.13.

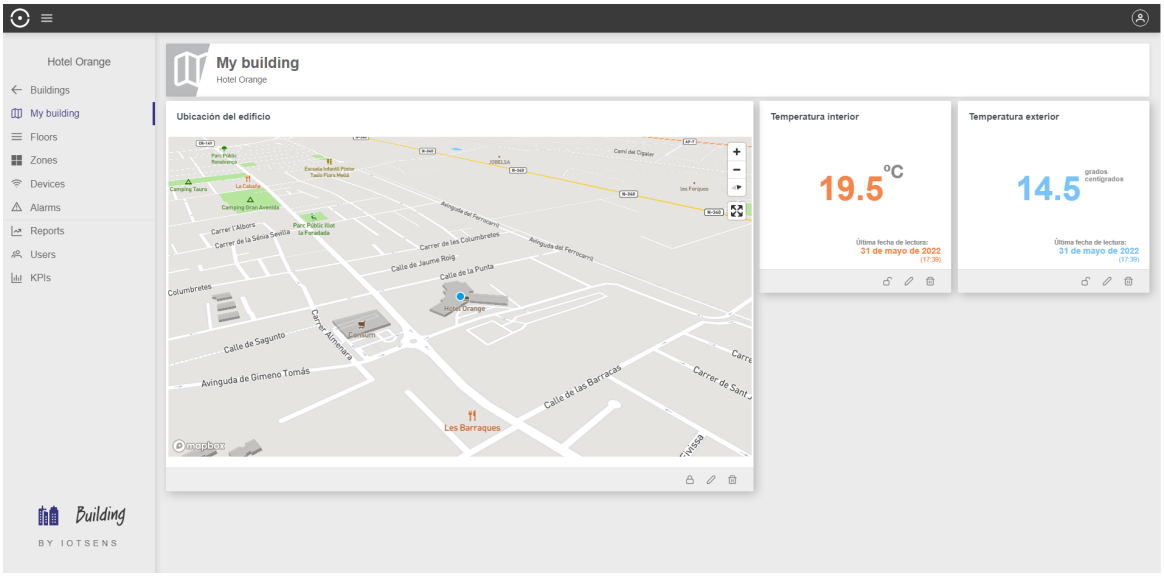

Figura 4.12: Vista del dashboard de un edificio

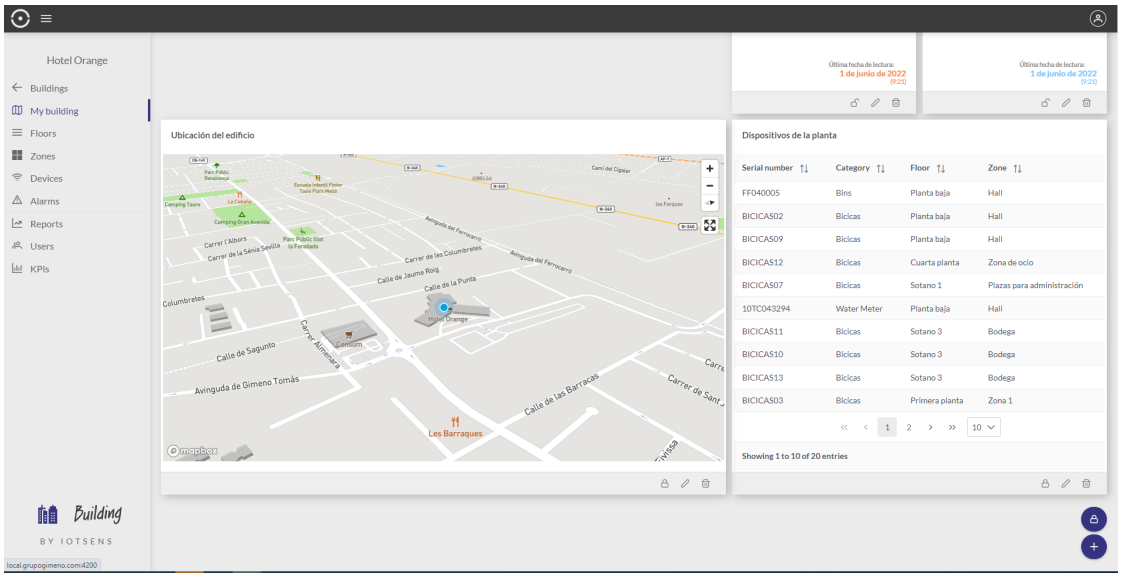

Figura 4.13: Vista del dashboard de un edificio con el widget del listado de dispositivos

4.3. Verificación y validación

El objetivo de la fase de verificación y validación es comprobar que el sistema funciona correctamente y según sus requisitos a todos los niveles. La verificación consiste en comprobar al final de cada fase de desarrollo si se satisfacen los requisitos especificados al inicio de la fase, esto ayuda a localizar rápidamente errores y no propagarlos a todo el proceso de desarrollo. Por otro lado, la validación consiste en comprobar que las especificaciones del proyecto cumplan con los requisitos del usuario final [16].

A lo largo del desarrollo del proyecto, se han pasado por distintas fases en las que se ha ido aplicando un proceso de verificación. Estas fases han sido el diseño de prototipos, el diseño de la base de datos, la implementación de los prototipos y de la lógica requerida, la implementación de Lazy Loading y la implementación de los widgets dinámicos.

En las dos primeras fases, el diseño de prototipos y el diseño de la base de datos, primero se mantuvo una reunión con mi supervisor para concretar unos requisitos mínimos que se necesitaban en para los diseños y al final de cada fase se volvía a tener una reunión con mi supervisor para comprobar si se cumplían los requisitos, si hacía falta añadir funcionalidad nueva por parte de los usuarios o si hacía falta cambiar cosas para que funcionara bien.

En la fase de implementación de los prototipos y de la lógica, se partió de unos requisitos que eran acercarse lo máximo posible a los prototipos y ver si podían surgir mejoras en el diseño. En cuanto a la verificación ya se añaden nuevos componentes como el uso del inspector de elementos, consola y red del navegador, para depurar errores de TypeScript, o para poder hacer uso del comando console.log para comprobar si se está pasando el objeto que se necesita, como para comprobar lo que se está enviando y recibiendo del backend. También se ha usado para comprobar cambios de CSS en caliente y ver si funcionan para luego añadirlos a la aplicación.

Por ejemplo, en la figura 4.14 se puede observar una manera de verificación de una parte de código, donde podemos ver que falla al intentar agregar las plantas en el formulario, y que gracias a eso se pudo comprobar que falta volver a cargar el listado de plantas para que al borrar una, se pudiera actualizar, y así conseguir solucionar el error.

Como también podemos utilizar la parte de red, para confirmar los datos que llegan desde el backend, como los datos que se envían desde el frontend al backend, como por ejemplo para ver si al listar las plantas de un edificio el backend manda el objeto que se necesita (véase figura 4.15).

Por último en la verificación, en la fase de implementación de Lazy Loading, podemos hacer uso de la parte de red del navegador, donde se puede observar que cada parte se carga únicamente cuando se necesita (véase figura 4.16). Se puede observar que se van cargando los datos por partes, primero las plantas, luego las zonas y luego los dispositivos, en función de lo que solicita el usuario.

Figura 4.14: Utilización de la consola del navegador

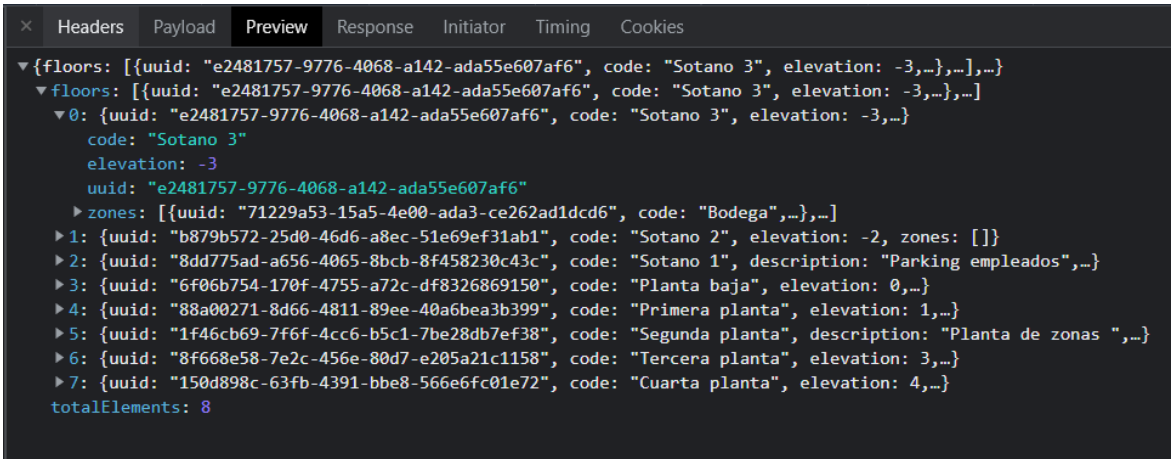

Figura 4.15: Utilización de la red del navegador para observar los datos que llegan del backend

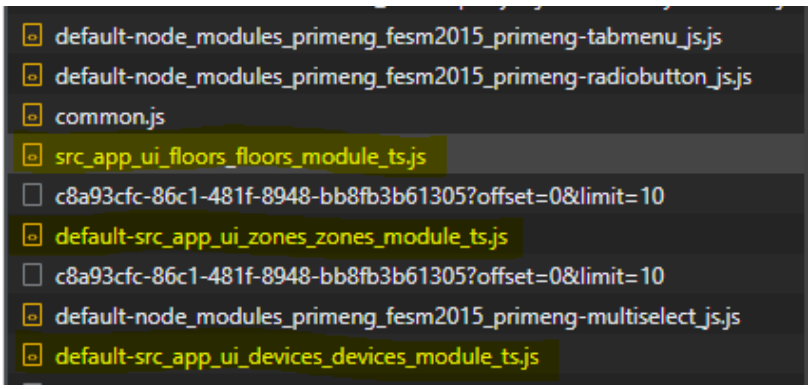

Figura 4.16: Utilización de la red del navegador para observar los datos que llegan del backend

Para la validación se han ido realizando reuniones con los jefes de la empresa para mostrar los avances y la funcionalidad que se iba consiguiendo, en las que se dejaban abiertas las tareas a posibles cambios, por si se estaba desarrollando alguna parte de la aplicación que no llegara a cuadrar con su idea. Además, el departamento de comunicación iba manteniendo reuniones con posibles clientes mostrando la aplicación y si luego hacía falta nos comunicaban un feedback con posibles cambios a realizar.

Capítulo 5

Conclusiones

El proyecto ha resultado ser un auténtico reto, en cuanto al aprendizaje de herramientas que apenas había utilizado antes tanto personal. En primer lugar, he aprendido mucho sobre el frontend utilizando el framework Angular y el lenguaje de programación Typescript, son dos tecnologías que no había utilizado antes de empezar en este proyecto, y la verdad que son bastante chulas y con mucha flexibilidad para añadir toda la lógica que se le ha podido añadir al proyecto.

En segundo lugar, he adquirido bastante experiencia respecto al trabajo en equipo en remoto, ya que mis dos compañeros de proyecto trabajan en remoto al no vivir cerca de la oficina, por lo que la comunicación en las primeras semanas era un poco escasa y difícil, pero pronto empezamos a realizar reuniones diarias online, para comentar un poco todo lo que hacía cada uno y poder poner un poco de orden a las tareas.

La realización de este proyecto me ha aportado unas sensaciones del todo positivas y gratificantes, el poder formar parte de la empresa, que los compañeros te ayuden con todas las dudas que surgen, y al final ver que vas consiguiendo realizar tareas y que lo que se realiza se está utilizando para efectuar demostraciones a los clientes sobre cómo va avanzando el desarrollo de la aplicación.

La metodología utilizada ha sido un éxito a mi parecer, nunca había realizado un proyecto utilizando esta metodología pero me ha sorprendido bastante, sobre todo la gran tolerancia a los cambios tras cada fase, poder cambiar requisitos o poder valorar qué tareas hacer antes que otras ya que tienen más importancia con los cambios de requisitos. Además ha ayudado a no tener que ir con el agua al cuello con los retrasos y cambios surgidos sobre todo en los dos primeros meses.

Por otro lado, no se han podido completar todos los requisitos iniciales, porque al finalizar las prácticas han faltado integrar los selectores de SVG tanto para los planos de las plantas como para determinar las zonas en el plano y el tema de los KPIs. A mitad de las prácticas se decidió dejar esos requisitos para cuando se tuviera toda la funcionalidad básica del proyecto en funcionamiento, porque con los dos reinicios de la aplicación al primer commit, se perdió bastante tiempo, y el compañero encargado del backend tenía muchos problemas con el token

de autenticación y fue todo mucho más lento.

Al no haber podido llegar a implementar la parte de los KPIs, de los cuatro objetivos principales, se ha llegado a conseguir tres, de los cuales estoy muy orgullosa de haber llegado a conseguir. Se ha conseguido tener una inteerfaz clara y consisa de todos los elementos necesarios del edificio (plantas, zonas y dispositivos) y se ha conseguido implementar los widgets dinámicos que se van a utilizar para mostrar los KPIs.

Lo bueno es que me han dado una oportunidad de entrar a trabajar en la empresa para seguir en el proyecto y poder terminarlo, por lo que me siento feliz porque me apetece seguir avanzando el proyecto y seguir aprendiendo cosas, como realizar el selector de SVG. Así que se podría decir que la empresa está bastante satisfecha con mi trabajo y los avances que he conseguido.

Bibliografía

[1] Metodología scrum. https://www.sinnaps.com/blog-gestion-proyectos/metodologia-scrum. [Consulta: 8 de Mayo de 2022].

[2] Iotsens. https://www.iotsens.com/es/. [Consulta: 5 de Mayo de 2022].

[3] City solution. https://www.iotsens.com/city. [Consulta: 5 de Mayo de 2022].

[4] Water solution. https://www.iotsens.com/water. [Consulta: 5 de Mayo de 2022].

[5] Industrial. https://www.iotsens.com/solucion/smart-industrial/. [Consulta: 5 de Mayo de 2022].

[6] Custom. https://www.iotsens.com/solucion/custom/. [Consulta: 5 de Mayo de 2022].

[7] ¿qué es scrum? https://www.atlassian.com/es/agile/scrum. [Consulta: 8 de Mayo de 2022].

[8] Mysql. https://www.mysql.com/. [Consulta: 6 de Mayo de 2022].

[9] Gitlab documentation. https://docs.gitlab.com/. [Consulta: 6 de Mayo de 2022].

[10] Redmine wiki. https://www.redmine.org/projects/redmine/wiki. [Consulta: 7 de Mayo de 2022].

[11] Historias de usuario con ejemplos y plantilla. https://www.atlassian.com/es/agile/project-management/user-stories. [Consulta: 15 de Mayo de 2022].

[12] Qué es el flat design o diseño plano. https://www.departamentodeinternet.com/que-es-flat-design-diseno-plano/. [Consulta: 15 de Mayo de 2022].

[13] Optimizar la carga inicial de una aplicación en angular con lazy load. https://pablotolentino.medium.com/optimizar-la-carga-de-una-aplicaci%C3%B3n-en-angular-con-lazy-load-d795b167fa7a. [Consulta: 29 de Mayo de 2022].

[14] Gestionar el estado de una aplicación angular usando rxjs behaviorsubject para servicios de datos observables. https://rneto.es/blog/gestionar-estado-angular-rxjs-behaviorsubject-servicios-datos-observables/. [Consulta: 29 de Mayo de 2022].

[15] Angular: Rxjs behaviorsubject. https://dev.to/dipteekhd/angular-behaviorsubject-p1. [Consulta: 29 de Mayo de 2022].

[16] ¿qué es verificación y qué es validación?
`https://www.sqs.es/que-es-verificacion-y-que-es-validacion/`. [Consulta: 30 de Mayo de 2022].

Anexo A

Requisitos de datos

Requisito de datos	
Código	RD01
Nombre	Edificio
Datos	Para cada edificio se deberá almacenar su identificador único, el nombre, el tenant, la latitud, la longitud.
Comentarios	Cada edificio debe de tener al menos una planta a la hora de ser creado.

Tabla A.1: Requisito de datos RD01

Requisito de datos	
Código	RD02
Nombre	Planta
Datos	Para cada planta se deberá almacenar su identificador único, el nombre, la elevación, una imagen del plano de la planta y el identificador único del edificio.
Comentarios	Cada planta debe de tener al menos una zona a la hora de sr creada.

Tabla A.2: Requisito de datos RD02

Requisito de datos	
Código	RD03
Nombre	Zona
Datos	Para cada zona se deberá almacenar su identificador único, el nombre, la posición dentro del plano de la planta y el identificador único de la planta.
Comentarios	-

Tabla A.3: Requisito de datos RD03

Requisito de datos	
Código	RD04
Nombre	Dispositivo
Datos	Para cada dispositivo se deberá almacenar su identificador único, el alias, la categoría y el identificador único de la zona en la que se ubica.
Comentarios	No se almacenan las medidas ni los sensores debido a que esos datos ya están almacenados en otra base de datos.

Tabla A.4: Requisito de datos RD04

Requisito de datos	
Código	RD05
Nombre	Usuario
Datos	Para cada usuario se deberá almacenar su identificador único y el nombre.
Comentarios	No se almacenan otros datos como el email o tipo de rol debido a que esos datos ya están almacenados en otra base de datos.

Tabla A.5: Requisito de datos RD05