ReservaTuSala: Aplicación web para la reserva de salas de reuniones

Enrique Gómez Becerra

Resumen — Con el aumento del teletrabajo en el panorama laboral actual, muchas empresas cuentan con salas de las que no se hace uso que otras personas podrían utilizar. Para sacar partido a estas salas, las empresas podrían ofrecer a otras personas o empresas la posibilidad de reservar y utilizar estos espacios durante unas horas para reunirse y realizar otras actividades laborales de carácter presencial. ReservaTuSala busca ofrecer a las empresas una aplicación web en la que registrar sus salas disponibles y gestionar las reservas realizadas por los usuarios, además de ofrecer a los usuarios la posibilidad de reservar las salas por horas.

Palabras clave — Salas, reuniones, empresas, reservas, aplicación, web, backend, frontend, Spring Boot, Angular, TypeScript, Docker, JWT.

Resum — Amb l'augment del teletreball en el panorama laboral actual, moltes empreses compten amb sales que no s'utilitzen i que altres persones podrien fer servir. Per treure profit d'aquestes sales, les empreses podrien oferir a altres persones o empreses la possibilitat de reservar i utilitzar aquests espais durant unes hores per a reunions i altres activitats laborals de caràcter presencial. ReservaTuSala busca oferir a les empreses una aplicació web on registrar les seves sales disponibles i gestionar les reserves fetes pels usuaris, a més d'oferir als usuaris la possibilitat de reservar les sales per hores.

Paraules clau — Sales, reunions, empreses, reserves, aplicació, web, backend, frontend, Spring Boot, Angular, TypeScript, Docker, JWT.

Abstract — With the increase of telecommuting in the current job market, many companies have unused rooms that other people could use. To take advantage of these rooms, companies could offer other individuals or businesses the possibility to reserve and use these spaces for a few hours to hold meetings and carry out other in-person work activities. ReservaTuSala aims to offer companies a web application where they can register their available rooms and manage the reservations made by users, as well as providing users with the option to book the rooms by the hour.

Index Terms — Rooms, meetings, companies, reservations, application, web, backend, frontend, Spring Boot, Angular, TypeScript, Docker, JWT.

1 Introducción

Con el aumento del teletrabajo a raíz de la pandemia del COVID-19 muchas empresas cuentan con espacios inutilizados a los que no se les saca beneficio durante largos periodos de tiempo. Algunos de estos espacios inutilizados son salas que otras personas o empresas que no dispongan de estas infraestructuras podrían estar interesadas en utilizar para realizar ciertas actividades profesionales que requieran de un trato presencial.

Por esa razón es posible que a aquellas empresas con salas disponibles les interese explotar ese recurso y reservar dichos espacios para su uso en sesiones de trabajo o reuniones y así conseguir ingresos adicionales con ello.

La idea desarrollar una aplicación web para facilitar a las empresas registrar la información de las salas libres de las que dispongan para que otras personas, que llamaremos clientes, puedan acceder a ellas y reservarlas durante unas horas.

2 Objetivos

Con el desarrollo de este proyecto el objetivo es construir una aplicación web que permita a las empresas registrar sus salas disponibles junto con información sobre su ubicación, materiales y horarios, así como gestionar las reservas realizadas sobre sus salas para aceptarlas o cancelarlas.

En el caso de los clientes, que podrán ser tanto individuales como otras empresas, podrán buscar y ver información sobre las salas y empresas disponibles en el sistema y tendrán la posibilidad de crear una reserva en una sala a unas determinadas horas. Cuando un cliente reserva una sala, ésta se marcará cómo ocupada, pero la reserva no será definitiva hasta que la empresa propietaria la acepte.

- *E-mail de contacto: enrique.gomezb@autonoma.cat*
- *Mención realizada: Ingeniería del Software*
- *Trabajo tutorizado por: Rodolfo Alberto Guichon Aguilar*
- *Curso 2022/23*

3 Estado del Arte

Actualmente, existen varias webs que ofrecen servicios muy similares a los objetivos comentados anteriormente como pueden ser Spathios[1], Regus[2] o SpacesON[3]. En este caso Spathios y SpacesON serian las más cercanas a los objetivos, ya que también se centran en la reserva de espacios para su uso profesional. Regus por otro lado va enfocado a la reserva de espacios de trabajo para su uso habitual, aunque también ofrece reservas de salas por horas.

Algunas webs populares que también ofrecen servicios similares pueden ser Airbnb[4] o Booking.com[5] que ofrecen reservas de espacios pero en este caso van enfocados principalmente al ámbito vacacional.

4 Metodologia

Para el desarrollo de este proyecto se ha utilizado una metodología ágil previamente enfocada al trabajo en equipo adaptada a un proyecto individual. El trabajo se ha repartido en total de 6 sprints quincenales, similares a los comúnmente utilizados en la metodología Scrum.

Aplicando esta metodología, al inicio del proyecto de han redactado un conjunto de tareas para satisfacer los requisitos del proyecto y se han planificado y repartido estas tareas en los 6 sprints previstos. Durante cada uno de estos sprints se han realizado las tareas previstas y al finalizar el sprint se ha revisado el trabajo realizado y se han llevado a cabo los cambios pertinentes en la planificación si existía alguna tarea inacabada. A diferencia de la metodología Scrum, no ha habido reuniones con el equipo de trabajo para revisar el estado del proyecto, ya que el equipo en este caso lo forma una única persona.

5 Planificación

El proyecto se ha dividido en las siguientes fases:

Planificación: En esta primera fase se ha definido la idea de proyecto, estableciendo los objetivos a cumplir, la metodología a seguir, los requerimientos del sistema y una panificación inicial con las tareas de cada uno de los 6 sprints.

Diseño: En la fase de diseño se ha definido la estructura del sistema y las necesidades de la solución. Para ello se han generado un diagrama de casos de uso, el esquema de la base de datos, un diagrama de clases con la representación de la estructura del backend, un esquema de vistas para la interfaz de la aplicación y un documento de especificaciones.

Implementación: En la fase de implementación se ha generado el código tanto de backend como de frontend a partir de los resultados de la etapa de diseño con el objetivo de satisfacer los requerimientos del sistema. Los requerimientos se encuentran disponibles en el anexo A1 de este documento.

Pruebas: En esta etapa final se han implementado pruebas unitarias para los métodos implementados en el backend y para cada uno de los componentes creados en el frontend. De esta manera se comprueba que las distintas partes que componen la aplicación funcionan de la manera esperada.

6 Herramientas y tecnologías

Para este proyecto se ha desarrollado una aplicación web completa, creando desde cero tanto el backend como el frontend.

Para la parte de backend se ha creado una base de datos relacional PostrgreSQL alojada en un contenedor Docker y una API REST desarrollada en Java Spring Boot[6] con la lógica para recibir y ejecutar las peticiones CRUD de los elementos de la aplicación.

En el frontend se implementado una interfaz de usuario creada con Angular[7] utilizando TypeScript cómo lenguaje de programación.

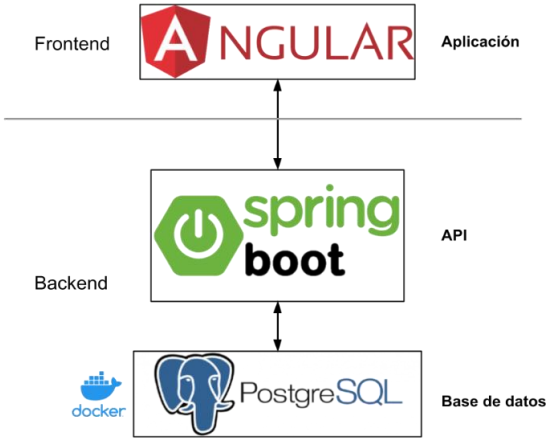

Fig. 1: Esquema de la arquitectura del sistema.

Además, para el desarrollo de la aplicación en las distintas etapas del proyecto se han utilizado un seguido de herramientas mencionadas a continuación.

- **PlantUML:** PlantUML es una plataforma utilizada para la creación de diagramas utilizando el lenguaje UML.
- **GitHub:** Plataforma dónde se alojan los repositorios utilizados para el control de versiones tanto del backend cómo del frontend.
- **Postman:** Herramienta utilizada para realizar las peticiones de prueba a la API para comprobar su correcto funcionamiento.
- **IntelliJ IDEA:** IDE utilizado para la implementación del código del backend en Java Spring Boot.
- **Visual Studio Code:** IDE utilizado en la implementación del código del frontend en Angular.

7 DESARROLLO DEL PROYECTO

En esta sección se detalla el trabajo realizado a lo largo desarrollo del proyecto.

7.1 Diseño

Al inicio del proyecto, con el objetivo de dar una definición sólida del proyecto, se redactó un documento de especificación de requisitos[8] en el que se definen la funcionalidad del sistema, las restricciones en la tecnología a utilizar, las características de los usuarios y los requerimientos funcionales y no funcionales esperados.

Una vez creado este documento, se pasó a la generación de diagramas para dar una definición a la estructura de la aplicación. Se crearon un total de 4 diagramas:

- **Diagrama de casos de uso:** Muestra las funcionalidades del sistema, obtenidas de los requerimientos del sistema, y su interacción con los distintos tipos de usuario.
- **Esquema de la base de datos:** Muestra la estructura de los objetos o tablas que existirán en la base de datos y las conexiones entre éstos.
- **Diagrama de clases de backend:** Muestra la estructura real del código del backend y la interacción entre sus distintos elementos.
- **Esquema de pantallas de la interfaz de usuario:** Muestra las distintas pantallas existentes en la aplicación y con las que el usuario deberá interactuar.

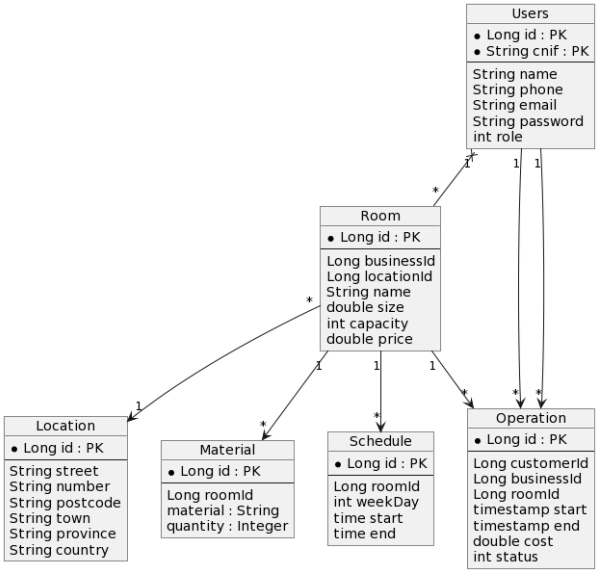

Fig. 2: Esquema de la base de datos.

Para poder entender mejor la aplicación y los elementos con los que trabaja, si observamos el esquema de la base de datos la Figura 2, vemos los distintos objetos presentes en la aplicación. Tenemos los usuarios (*User*) que en caso de ser empresas podrán contar con salas (*Room*). También tenemos operaciones (*Operation*) que contienen la información sobre las reservas, entre ella el cliente, la empresa y la sala asociadas a cada una de las reservas. Por otro lado, las salas (*Room*) cuentan con una ubicación (*Location*) asociada, una lista de materiales (*Material*) disponibles en dicha sala y los días y horas de horario (*Schedule*) de disponibilidad.

7.2 Backend

7.2.1 Base de datos

Para el backend de la aplicación, inicialmente se ha creado la base de datos PostgreSQL en un contenedor Docker utilizando Docker Compose[9]. Con esto se obtiene un contenedor con la imagen de PostgreSQL listo para ser utilizado.

Para inicializar las tablas en la base de datos, basta con definir el modelo de datos en la aplicación con Spring Boot, ya que se hace uso del modulo Java Persistence API[10] (JPA) que gestiona la base de datos y crea las tablas de manera automática al ejecutar el código de la aplicación. Por lo tanto basta con configurar la conexión con la base de datos en el contenedor para que se generen las tablas asociadas a aquellos objetos con la anotación "*@Entity*".

7.2.2 Arquitectura Controller-Service-Repository

Para la creación de API con Java Spring Boot, se ha utilizado la arquitectura Controller-Service-Repository. En esta arquitectura cada uno de los objetos de la base de datos requiere de una clase modelo, una clase repositorio, una clase servicio y una clase controlador.

En la capa de modelo se define la estructura del objeto, es decir, sus atributos. Se crean las clases para cada elemento y se anotan cómo "*Entity*", para hacerlas detectables por el módulo JPA. Además, en este caso se ha hecho uso de la librería Lombok[11] que ofrece una serie de anotaciones que permiten generar los constructores, *getters* y *setters* de las clases de manera automática para hacer un código más comprensible visualmente.

La capa de repositorio cuenta con las clases anotadas cómo "*Repository*" que se encargan de las consultas y operaciones en la base de datos. Las operaciones más sencillas no requieren de una definición explicita de la consulta, sino que ya vienen preestablecidas en la superclase JpaRepository que extienden estas clases. Por esa razón para las operaciones CRUD a la base de datos no ha sido necesaria la creación explícita de consultas.

En la capa servicios se encuentra las clases "*Service*" con toda la lógica de la aplicación y se encarga de las llamadas a las funciones de la capa repositorio. Es en esta capa dónde se hacen las comprobaciones previas a la consulta o modificación de la base de datos, cómo por ejemplo la comprobación de la existencia de un usuario con un correo electrónico antes de crear un nuevo usuario con ese mismo correo.

Por último tenemos la capa controlador en la que, para cada elemento de la base de datos, se encuentran las clases "*RestController*" con el mapeo de las diferentes peticiones

HTTP disponibles y la gestión de la respuesta a cada una de ellas. De forma adicional, se ha utilizado la anotación "*@Valid*" en los objetos recibidos en el cuerpo de las peticiones en los controladores para comprobar que la estructura del objeto se ajusta a la del modelo de datos, para así evitar peticiones erróneas que provoquen fallos en el servidor. De esta forma, cuando se detecte un modelo erróneo se devolverá un mensaje de error del tipo "*bad request*".

7.2.3 Manejo de errores

Existen casos en los que la aplicación deberá devolver mensajes y códigos de error para informar de un mal uso de la API. En este caso se han utilizado 2 métodos distintos para ello.

Para los casos en los que el objeto recibido por la capa de controlador detecte un modelo de datos erróneo o en caso de intentar añadir un nuevo usuario con un correo electrónico ya existente en la base de datos, la aplicación lanza unas excepciones de manera automática para las que se han creado unos *exception handlers* que traducen estas excepciones a mensajes de error.

Para aquellos casos en os que no se lanzan excepciones, que son casos de error encontrados en la capa de servicios, también se devuelve un mensaje de error formado por un código y un texto breve explicando el error.

7.2.4 Seguridad

Para la seguridad de la aplicación, se ha configurado el uso de un token de autenticación Json Web Token[12] (JWT) para el control de acceso utilizando el módulo de Spring Security. Cada una de las peticiones recibidas por la API requerirá de este token de acceso en el encabezado y, en caso de no existir, se devolverá una respuesta de error del tipo "*Unauthorized*".

Para obtener este token de acceso, el usuario deberá autenticarse utilizando correo electrónico y contraseña. Al hacerlo recibirá el token con el que tendrá acceso temporal a la aplicación durante 15 minutos.

7.2.4 Swagger

Con la implementación del backend de la aplicación obtenemos un gran número de peticiones distintas. Para hacer más accesible la estructura de cada petición se ha documentado la API utilizando Swagger con el estándar de OpenAPI[13].

Para acceder a esta documentación basta con ejecutar el backend de la aplicación y acceder a la dirección "*localhost:8081/swagger-ui/index.html*" tal como se observa en la Figura 3.

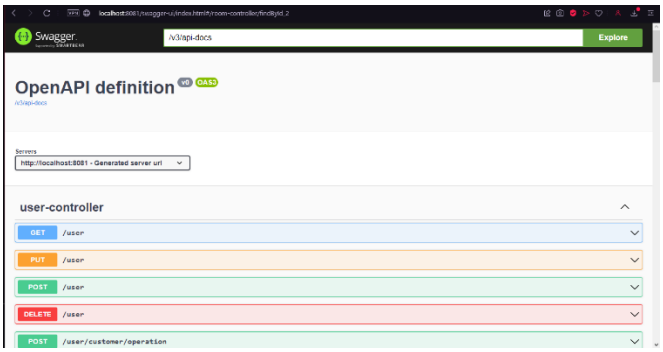

Fig. 3: Página de documentación de la API con Swagger.

7.2.5 Postman

Durante el desarrollo de la aplicación ha sido necesario probar las peticiones a la API para comprobar las respuestas obtenidas y su correcto funcionamiento en general.

Para realizar estas pruebas se ha utilizado la herramienta Postman[14] con la que se puede definir fácilmente una petición asignándole una URL, parámetros, encabezados y cuerpo. De esta forma se hacía posible probar distintos modelos de petición aplicando algunos cambios y analizar la respuesta.

7.2.5 Pruebas

Para las pruebas del código de backend se ha hecho uso de Junit 5[15] y se ha hecho coverage sobre las clases de controlador y servicios, que son las que cargan con el peso de la lógica de la aplicación. Los resultados pueden observarse en la Figura 4.

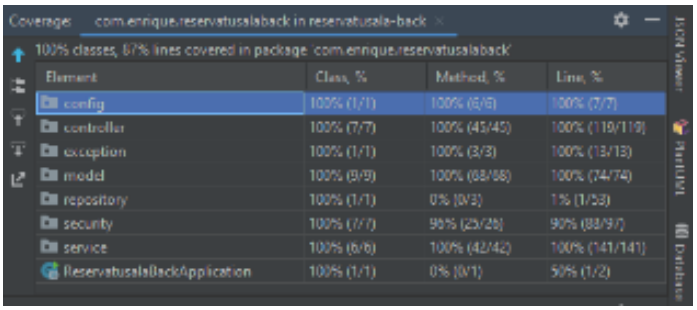

Fig. 4: Resultados de tests y coverage.

La capa repositorio únicamente contiene métodos generados automáticamente por JPA que ejecutan operaciones sobre la base de datos, por lo que no requiere de pruebas al carecer de código implementado.

7.3 Frontend

7.3.1 Angular Material

En el frontend de la aplicación se ha desarrollado con Angular utilizando Typescript. Este framework permite utilizar componentes provenientes de librerías externas que aportan múltiples ventajas a la aplicación ya que cuentan con funcionalidades y animaciones ya implementadas y son altamente personalizables.

Para este proyecto se ha utilizado la librería de

componentes Angular Material[16] que cuenta con un gran numero de componentes y tiene fácil integración en una aplicación con Angular. Con el uso de estos componentes se ha agilizado la creación de nuevas pantallas y se ha mejorado la UX, ya que muchos de estos componentes ya cuentan con animaciones cuando el usuario interactúa con ellos.

7.3.2 Servicios

Para comunicar el frontend con la API se ha hecho uso de servicios para realizar las peticiones HTTP. Cada uno de los elementos de la aplicación (*User*, *Room*, *Operation*, *Location*, *Schedule* y *Material*) cuenta con un servicio en el que hay una función para cada una de las operaciones CRUD que envía la petición HTTP correspondiente a la API.

Los componentes que hacen uso de estos servicios se han implementado de manera que cuando se realiza una petición y esta recibe una respuesta positiva, los elementos de las vistas se actualizan con los datos actualizados. Además, en aquellos elementos que cuentan con fechas u horas entre sus atributos, ha sido necesario convertir los datos obtenidos en formato *string* a objetos de tipo *Date* para trabajar con ellos en el frontend. De la misma forma, para las peticiones en las que hay que enviar esos mismos objetos en el cuerpo de la petición, ha sido necesario revertir esa conversión para enviar la fecha u hora en formato de texto.

También se han creado servicios para implementar aquellas funciones que deberían ser llamadas desde distintos componentes de la aplicación, entre ellos servicios para gestionar el token de autenticación o para controlar las animaciones de algunos componentes para mostrarse u ocultarse.

7.3.3 Interfaces

Para mantener una estructura fija en los elementos de la aplicación, se han definido interfaces en las que se definen los atributos presentes en cada uno. De esta forma se facilita la creación de nuevos objetos y hace posible la especificación del tipo de elemento que se envía en los cuerpos de las peticiones HTTP simplemente indicando la interfaz pertinente.

7.3.4 Interceptores

Las peticiones enviadas por los servicios requieren de la presencia del token de autenticación JWT en sus encabezados, pero asignar ese valor a cada una de las peticiones de forma individual reduce drásticamente la mantenibilidad del código.

Por esta razón se ha creado un interceptor que se encargará de esta tarea. Este elemento añade al encabezado de cada una de las peticiones HTTP emitidas por la aplicación el token de autenticación, siempre que exista. De esta manera se hace posible la correcta comunicación con la API para aquellos usuarios autenticados.

7.3.5 Guards

La aplicación cuenta con 3 tipos de usuario: administrador, empresa y cliente. Cada usuario debe ver las vistas asociadas a su rol y nunca debería ver las de los otros roles. Por lo tanto, se debe controlar el acceso a las distintas vistas para conceder acceso únicamente a aquellos usuarios con el rol pertinente.

Con este fin y con el de controlar el acceso a la aplicación a los usuarios no autenticados, se han creado *guards* para valorar qué usuarios deben acceder a qué rutas. Cada tipo de usuario cuenta con su propio *guard*, que comprueba que el rol del usuario presente en el token de autenticación es el esperado. En caso de no serlo se muestra una vista en la que se informa al usuario que no está autorizado para acceder a la vista.

Además de los *guards* creados para los roles de usuario, se ha creado un guard adicional para controlar el acceso a la vista de edición de usuario, para permitir el acceso únicamente a aquellos usuarios autenticados.

De esta manera cada usuario accede a las vistas asignadas a su rol y los usuarios sin autenticar únicamente tienen permitido acceder a las vistas de inicio de sesión y registro de usuario.

7.3.6 Caducidad de autenticación

Para evitar que los usuarios naveguen por la aplicación sin token de autenticación sin saberlo tras pasar los 15 minutos de tiempo de caducidad, se comprueba que el token no haya caducado en el momento de navegar entre vistas. Si se detecta un token caducado, se muestra al usuario un mensaje indicando que debe iniciar sesión de nuevo junto con un botón “Salir” para dirigirlo a la vista de inicio de sesión.

7.3.7 Vistas

Cada tipo de usuario tiene un conjunto de vistas disponibles con las que puede interactuar para llevar a cabo las diversas acciones disponibles recogidas en los requerimientos del sistema que pueden verse en el anexo A1 de este documento.

Para el usuario administrador se han creado vistas para la gestión de los datos de usuarios, salas, operaciones y localizaciones junto con una vista de inicio con botones para navegar a dichas vistas. Cada una de estas vistas muestra una tabla con los elementos existentes y permite la modificación o eliminación de estos elementos o la creación de nuevas instancias de dichos elementos. Para la creación y modificación de elementos se abre un modal con un formulario que permite la introducción o modificación de los datos respectivamente. En caso de querer eliminar un elemento simplemente es necesario confirmar la eliminación en el modal emergente.
En el caso del usuario empresa, se ha creado una vista de inicio en la que se muestran 2 tablas: una primera con los datos de las reservas a sus salas con la fecha de hoy y otra con los datos de las reservas a sus salas en estado

pendiente a espera de una aprobación o cancelación. También cuenta con una vista para consultar las operaciones hechas a sus salas y modificar sus estados y otra vista para gestionar sus salas en la que tendrá la posibilidad de consultar, modificar o eliminar sus salas actuales o crear nuevas.

Para el usuario cliente se ha creado una vista inicial con un buscador de salas a partir del nombre de sala, empresa o ciudad y con la posibilidad de aplicar filtros sobre el tamaño, capacidad y precio por hora de las salas. Los resultados de la búsqueda se muestran bajo el campo de texto del buscador y muestran información sobre las salas. Cada elemento de los resultados cuenta con un botón de "mostrar detalles" con el que se navega a la vista de detalle de sala con más información sobre la sala seleccionada. Desde esta vista se permite tanto navegar a la vista de detalles de la empresa dueña de sala cómo la creación de una reserva a partir de una fecha y las horas de inicio y final. Antes de confirmar la reserva se muestran tanto el número de horas totales como el coste final de la reserva. Además, se ha creado una vista en la que se muestra una tabla con las reservas creadas por el usuario y le da la posibilidad de cancelar dichas operaciones.

También se ha implementado una vista de edición de usuario con la que un usuario autenticado de cualquier tipo tiene la posibilidad de modificar los datos de su cuenta a partir de un formulario.

Excepto en las vistas de inicio de sesión y registro de usuario, en la barra superior de la aplicación existe un botón de menú en la esquina superior izquierda con la que se abre un menú de navegación con las distintas páginas disponibles para el tipo de usuario actual. Además, en la misma barra superior existe en la parte superior derecha un botón de selección de idioma para el cambio de la aplicación a español, catalán o inglés y un botón con el que se abre un menú desplegable con las opciones de edición de usuario y cierre de sesión.

Las vistas creadas permiten realizar las acciones definidas para cada tipo de usuario en el diagrama de casos de uso que se muestra en el anexo A2.

7.3.8 Manejo de errores

Para cada petición a la API creada por el usuario al realizar una operación, se mostrará el resultado en forma de *snackbar* en la parte baja de la pantalla. Si la petición se ha realizado con éxito se mostrará un mensaje de color verde y en caso de haberse producido un error se mostrará de color rojo.

Además, el error puede contener un código o no. En caso de contar con un código de error, en el *snackbar* se indicará el tipo de error que se ha producido, en caso contrario se mostrará simplemente que se ha producido un error en la operación. En las Figuras 25 y 26 se muestran un ejemplo de mensaje de operación exitosa y un mensaje de error respectivamente.

7.3.9 Pruebas

Para comprobar que los componentes generados para las vistas se construyen y funcionan de la manera esperada, se han creado pruebas unitarias para cada uno de ellos en las que se prueba que los datos se reciben correctamente al inicio y las funciones realizan las operaciones de manera correcta. Como se muestra en la Figura 5, todas las pruebas implementadas obtienen un resultado positivo.

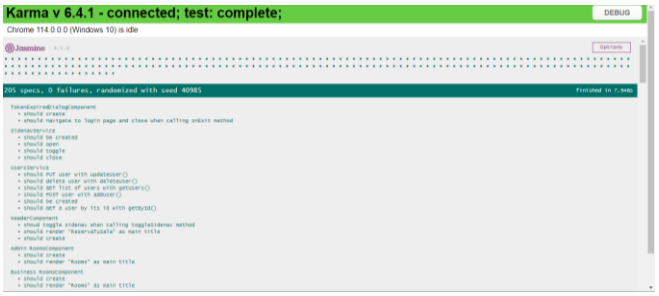

Fig. 5: Resultados de las pruebas en los componentes.

En los requerimientos no funcionales definidos, el RNF_2 para ser exactos, se expone que la aplicación debe ser compatible con los navegadores Chrome, Microsoft Edge, Opera y Firefox. Una de las ventajas de Angular es su compatibilidad con los principales navegadores, entre ellos los navegadores definidos en el requerimiento. Al comprobar el funcionamiento de la aplicación en dichos navegadores, no se detecta ninguna anomalía. En el anexo A3 de este documento se muestran imágenes de la vista de inicio de cliente en cada uno de los navegadores.

8 Resultados

8.1 Backend

Para empezar es necesario iniciar el contenedor Docker utilizando el fichero "docker-compose.yml" proporcionado. Una vez en funcionamiento, tal como se observa en la Figura 6, es posible ejecutar el código del backend.

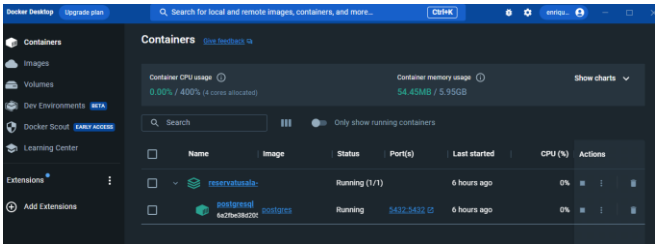

Fig. 6: Contenedor Docker corriendo.

La aplicación cuenta con una API desarrollada en Java utilizando el framework Spring Boot que recibe peticiones para la ejecución de las operaciones CRUD sobre la base de datos para cada uno de los elementos presentes en la aplicación: *User*, *Room*, *Operation*, *Location*, *Schedule* y *Material*. En la Figura 7 se muestra el servidor en funcionamiento y disponible para recibir peticiones.

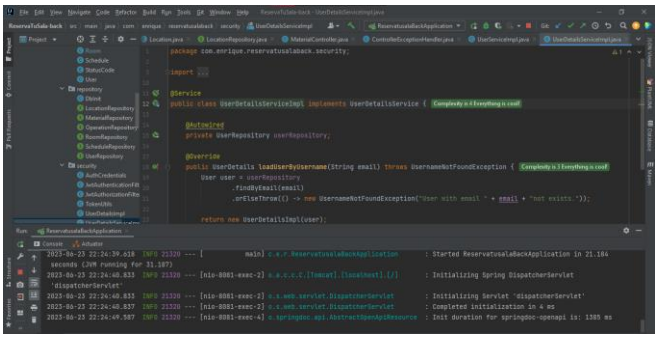

Fig. 7: Servidor API en funcionamiento.

8.2 Frontend

Con la API en funcionamiento ya es posible ejecutar el frontend de la aplicación. En la Figura 8 vemos que el código se compila y ejecuta sin problemas y se nos muestra la ruta en la que se encuentra disponible la aplicación.

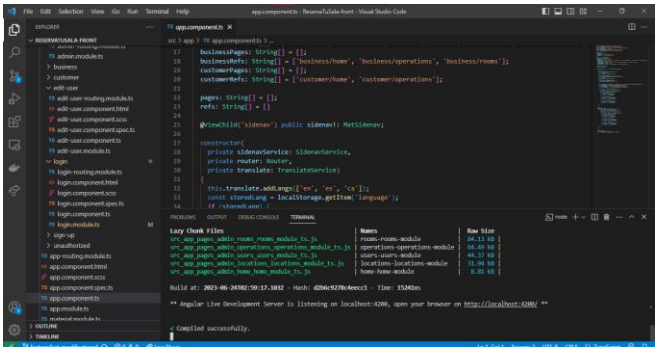

Fig. 8: Frontend en funcionamiento.

Si accedemos a la ruta que se nos indica, en el caso de la Figura 8 sería "*http://localhost:4200/*" se nos redirige a la pantalla de inicio de sesión que se muestra en la Figura 8. Para acceder a la vista de registro de usuario basta con pulsar uno de los botones "Registrarse" ya sea en la parte superior derecha de la pantalla o bajo el formulario de inicio de sesión.

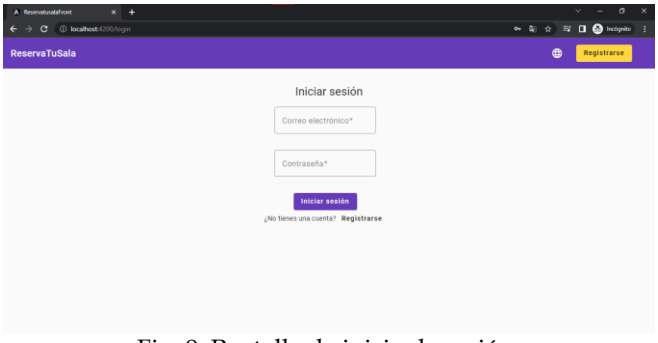

Fig. 9: Pantalla de inicio de sesión.

Al pulsar el botón "Registrarse" se nos dirige a la página de inicio de sesión en la que podemos introducir nuestros datos para crear una nueva cuenta tal como se ve en la Figura 10.

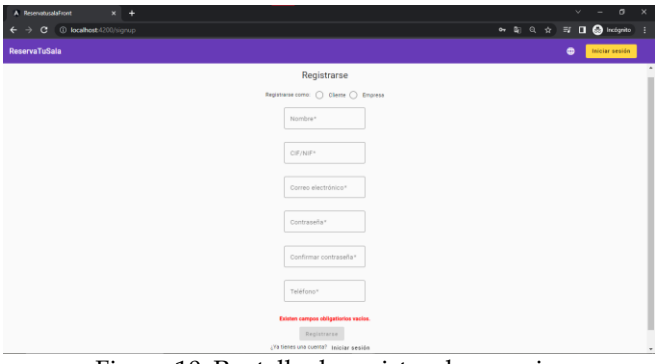

Figura 10: Pantalla de registro de usuario.

Si creamos un nuevo usuario de tipo cliente, se nos dirige a la página de inicio de cliente en la que tenemos el buscador de salas por nombre de sala, empresa o ciudad. Si pulsamos sobre el botón "Filtrar" podremos filtrar los resultados por tamaño, capacidad y precio por hora de las salas. En la Figura 11 se muestra la pantalla de inicio de cliente con el modal de selección de filtros abierto.

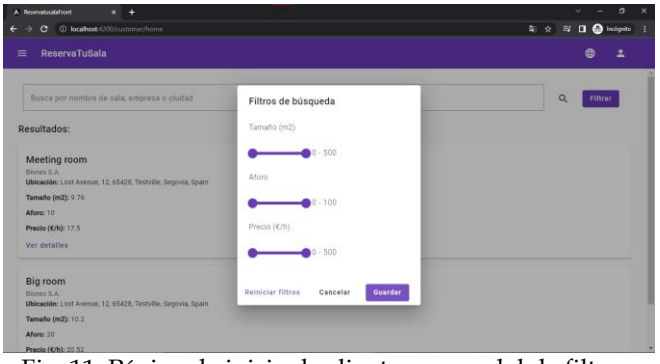

Fig. 11: Página de inicio de cliente con modal de filtros.

Al pulsar el botón "Ver detalles" de una de las salas se nos muestra la vista de detalle de sala con la información de la sala y un formulario en la parte derecha para la creación de una reserva, tal como se muestra en la Figura 12.

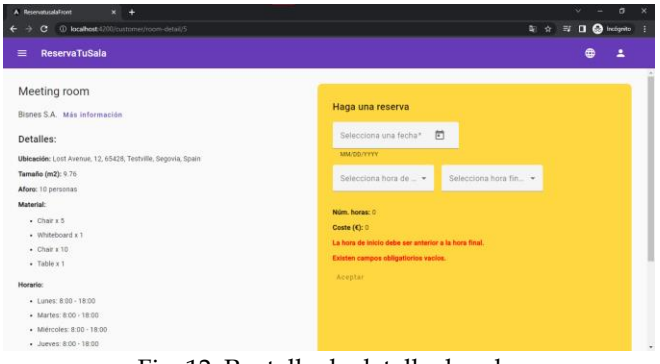

Fig. 12: Pantalla de detalle de sala.

Si pulsamos el botón "Más información" que se encuentra al lado del nombre de la empresa, se nos dirige a la página de detalle de empresa que se muestra en la Figura 13 con la información de contacto de la empresa.

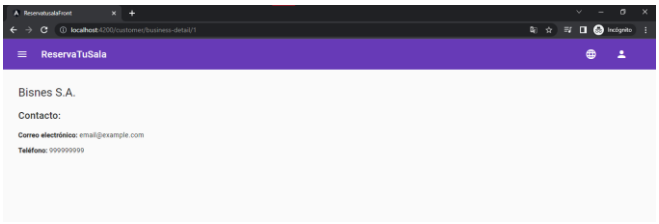

Fig. 13: Vista de detalle de empresa.

El usuario cliente también cuenta con una vista de operaciones con el histórico de las operaciones que ha realizado. La vista es muy similar a la vista de operaciones de empresa de la Figura 17, con la diferencia de que la única acción posible es cancelar la operación.

También existen algunos botones en la barra superior de la aplicación que están disponibles para todos los tipos de usuario. Si pulsamos en el botón con las tres barras horizontales en la parte superior izquierda, se abre un menú lateral visible en la Figura 13 en el que se muestran las vistas disponibles para el usuario actual, en este caso el usuario cliente. En cambio, el botón con el icono circular en la parte superior derecha permite la selección del idioma de la aplicación entre las opciones español, catalán e inglés y el botón con el icono de usuario abre un menú con la información del usuario y las opciones de modificar el usuario o cerrar sesión, tal como se observa en la Figura 14.

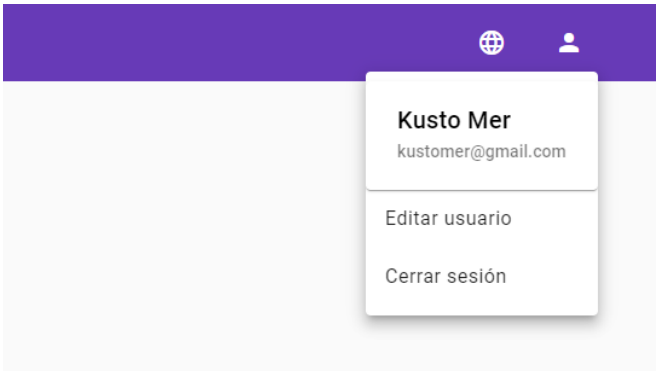

Fig. 14: Menú de usuario.

Si pulsamos la opción “Editar usuario” se nos dirigirá a la vista de edición de usuario que se muestra en la Figura 15 y si pulsamos en “Cerrar sesión” se os dirigirá a la pantalla de inició de sesión ya vista en la Figura 9.

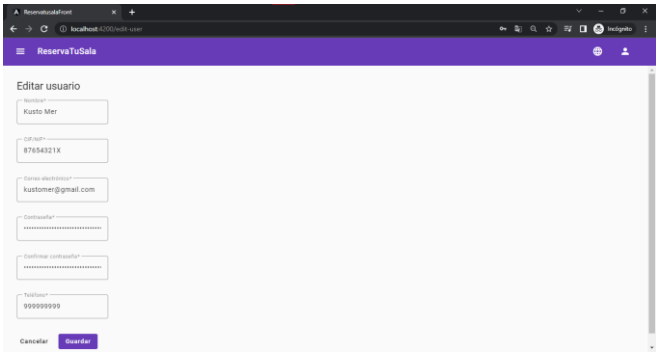

Fig. 15: Vista de edición de usuario.

Si ahora accedemos a la aplicación cómo empresa, se nos dirige a la página de inicio de empresa en la que se muestran las tablas con las reservas de hoy y las reservas pendientes. En la Figura 16 observamos que no existen reservas para hoy, por lo que no se muestra la primera tabla. En esta misma figura también vemos que el menú lateral de navegación también cambia respecto al usuario cliente.

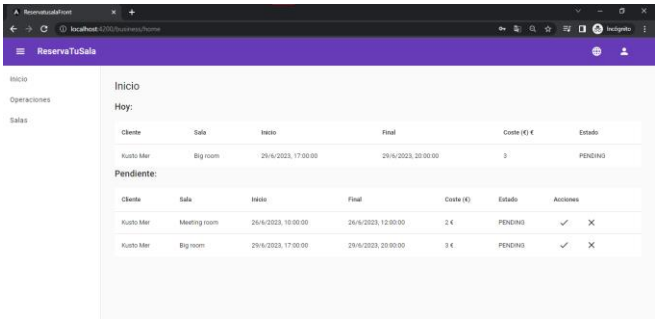

Fig. 16: Vista inicio de empresa.

En la segunda tabla de la pantalla de inicio se permite aprobar o cancelar las reservas pendientes. Si aprobamos una reserva, ésta se eliminará de la tabla. Si ahora navegamos a la pantalla de operaciones, dicha operación se mostrará con el estado modificado tal como se observa en la Figura 17. Desde esta misma vista de operaciones se nos permite modificar el estado de la operación.

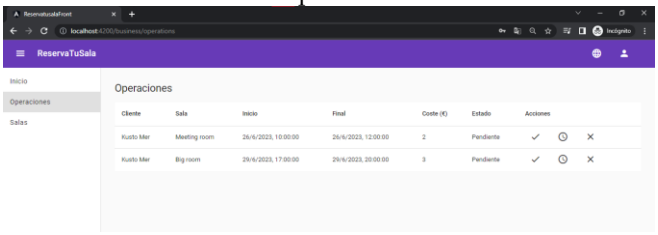

Fig. 17: Vista de operaciones de empresa.

Al navegar a la pantalla de salas, observamos en la Figura 18 que es posible consultar las salas de la empresa. Además, se permite modificar o eliminar salas utilizando los iconos en la parte derecha de la tabla y añadir nuevas salas pulsando el botón “Nueva sala” que abre un modal con un formulario. En el caso de la modificación de una sala, el modal seria el mismo, pero con los campos del formulario rellenados con los datos de la sala a modificar tal como se muestra en la Figura 19.

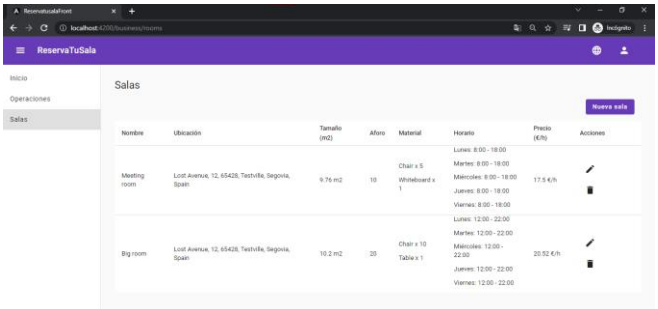

Fig. 18: Vista de salas de empresa.

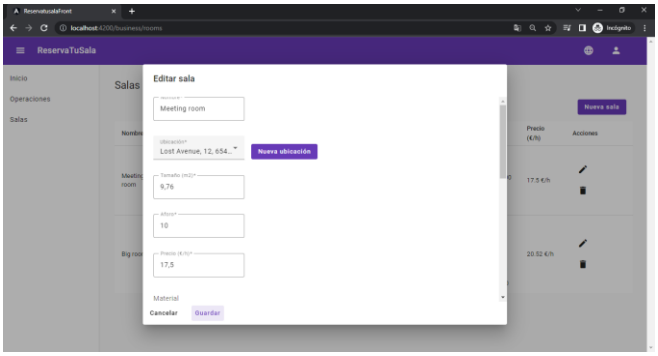

Fig. 19: Vista de salas de empresa con modal de editar sala.

Al acceder a la aplicación cómo administrador, nos encontramos con una página de inicio simple con botones para navegar a las distintas vistas disponibles como vemos en la Figura 20.

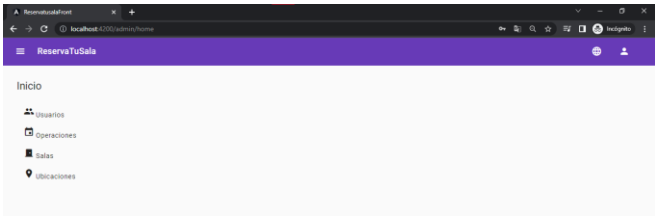

Fig. 20: Vista de inicio de administrador.

En este caso las vistas son muy similares. En todas ellas nos encontramos una vista cómo la de la Figura 21, en la que se muestra una tabla con todas las instancias en la base de datos del elemento en cuestión y se nos permite modificar o eliminar elementos y añadir nuevos elementos a la tabla. En la Figura 22 se muestra el modal con el formulario para la creación de un nuevo usuario, que sigue la misma dinámica que los modales mostrados para la creación y modificación de cualquiera de los elementos: usuarios, operaciones, salas y ubicaciones.

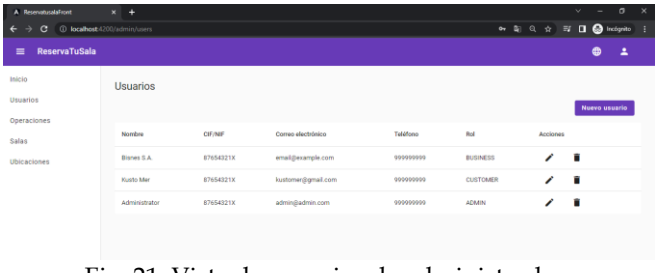

Fig. 21: Vista de usuarios de administrador.

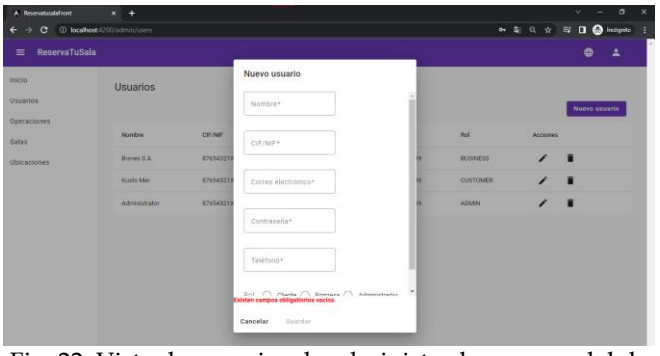

Fig. 22: Vista de usuarios de administrador con modal de nuevo usuario.

Al intentar acceder a una ruta perteneciente a otro tipo de usuario no encontramos con la vista de usuario no autorizado que se muestra en la figura 23.

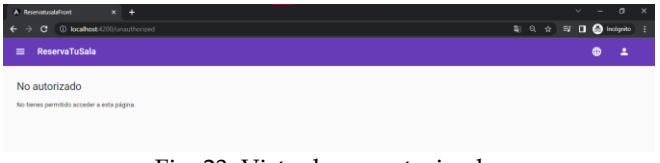

Fig. 23: Vista de no-autorizado.

Tras pasar 15 minutos del inicio de sesión, al usuario se le mostrará un modal como el que se muestra en la Figura 24 indicándole que la sesión ha caducado y deberá volver a iniciar sesión. Con el botón “Salir” se le dirigirá a la vista de inicio de sesión.

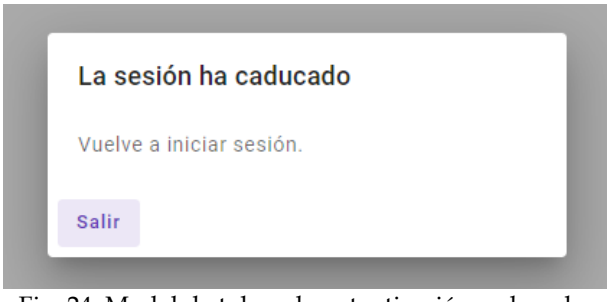

Fig. 24: Modal de token de autenticación caducado.

Para cada operación realizada por parte del usuario que requiera de una petición a la API se muestra un mensaje indicando si se ha realizado correctamente o se ha producido un error. En las Figuras 25 y 26 vemos un ejemplo de cada caso.

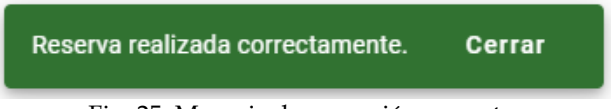

Fig. 25: Mensaje de operación correcta.

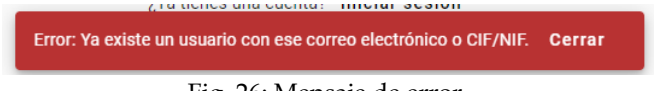

Fig. 26: Mensaje de error.

9 Conclusiones

El objetivo buscado con el desarrollo de este proyecto era crear aplicación web que permita la gestión de salas y sus reservas por horas.

Con el trabajo realizado se ha implementado una aplicación web completa desde cero, para la que se ha creado desde la base de datos hasta la interfaz de usuario pasando por el servidor con una API funcional. El resultado obtenido con ello cumple con los objetivos y requerimientos definidos al inicio del proyecto.

Aunque el estado actual de la aplicación deja bastante que desear visualmente, en cuanto a funcionalidad ofrece un servicio de calidad al usuario final, que puede hacer uso de la aplicación de manera intuitiva y sin errores inesperados.

En el futuro se podrían aplicar algunas mejoras sobre la aplicación. Algunas de estas mejoras pueden ser: añadir la posibilidad de subir imágenes de las salas; añadir imágenes de perfil a los usuarios; mejorar el estilo y darle una paleta de colores representativa a la aplicación; o mejorar la usabilidad utilizando un token de refresco en lugar de requerir iniciar sesión de nuevo para acceder al sistema.

Agradecimientos

Me gustaría agradecer a mi tutor Rodolfo Guichon toda la paciencia que ha tenido y todos los comentarios de mejora que han hecho de este proyecto lo que es hoy.

También agradecer a mis tutores de prácticas Francisco Javier Delgado y Alejandro Marín que me dieron las herramientas necesarias en las fases iniciales de la aplicación para convertirla en lo que es hoy.

Finalmente, también agradecer a todos mis compañeros y familia que me han acompañado durante todos estos años y que tampoco hubiera sido posible sin ellos.

Bibliografía

[1] Spathios. (s. f.). https://app.spathios.com/es
[2] Regus. (s.f.). https://www.regus.com
[3] SpacesON. (s.f.). https://www.spaces-on.com
[4] Airbnb. (s.f.). https://www.airbnb.es
[5] Booking.com. (s.f). https://www.booking.com
[6] Learn Spring Boot Tutorial - javatpoint. (s. f.). https://www.javatpoint.com/spring-boot-tutorial
[7] Angular. (s. f.). https://angular.io/guide/what-is-angular
[8] Documento de especificación de requisitos.
[9] Docker Compose overview. (s.f.). Docker Documentation. https://docs.docker.com/compose/
[10] Spring Boot JPA - javatpoint. (s. f.). www.javatpoint.com. https://www.javatpoint.com/spring-boot-jpa
[11] Project Lombok. (s. f.). https://projectlombok.org/
[12] Spring Security - JWT. (s. f.). https://www.tutorialspoint.com/spring_security/spring_security_with_jwt.htm
[13] OpenAPI Specification - Swagger. (s. f.). https://swagger.io/specification/
[14] Postman. (s. f.). https://www.postman.com/
[15] Baeldung, & Baeldung. (2023c). A Guide to JUnit 5 | Baeldung. Baeldung. https://www.baeldung.com/junit-5
[16] Angular Material. (s. f.). https://material.angular.io/

APÉNDICE

A1. REQUERIMIENTOS DEL SISTEMA

A1.1. REQUERIMIENTOS FUNCIONALES

ID	Descripción
RF_1	El sistema debe permitir a los usuarios crear una cuenta con su nombre, NIF/CIF, teléfono, correo electrónico y contraseña.
RF_2	Si el usuario tiene una cuenta, el sistema debe permitir a los usuarios iniciar sesión con su correo electrónico y contraseña.
RF_3	El sistema debe permitir a los usuarios registrarse como empresa o cliente.
RF_4	El sistema debe permitir al usuario ver el histórico de sus reservas.
RF_5	El sistema debe permitir a los usuarios modificar los datos de su cuenta.
RF_6	El sistema debe autenticar a un usuario a partir de su correo electrónico y contraseña.
RF_7	Si el usuario pertenece a una empresa, el sistema debe permitir al usuario consultar, añadir o eliminar salas y modificar sus características: nombre, tamaño, precio por hora, ubicación, equipamiento y horarios.
RF_8	Si el usuario pertenece a una empresa, el sistema debe permitir al usuario aceptar o rechazar las reservas pendientes sobre sus salas.
RF_9	Si el usuario pertenece a una empresa, el sistema debe permitir al usuario consultar las reservas sobre sus salas.
RF_10	Si el usuario pertenece a un cliente, el sistema debe permitir al usuario buscar salas por el nombre de la empresa o su localidad.
RF_11	Si el usuario pertenece a un cliente, el sistema debe permitir al usuario aplicar filtros sobre la búsqueda de salas.
RF_12	Si el usuario pertenece a un cliente, el sistema debe permitir al usuario ver los detalles de una empresa.
RF_13	Si el usuario pertenece a un cliente, el sistema debe permitir al usuario ver los detalles de una sala.
RF_14	Si el usuario pertenece a un cliente, el sistema debe permitir al usuario reservar una sala durante unas horas concretas si éstas están disponibles.
RF_15	Si el usuario pertenece a un cliente, el sistema debe permitir al usuario cancelar sus reservas.
RF_16	Si el usuario es administrador, el sistema debe permitir al usuario consultar, añadir, eliminar o modificar los datos pertenecientes a usuarios, salas u operaciones.
RF_17	El sistema debe estar disponible en los idiomas español, catalán e inglés.

A1.1. REQUERIMIENTOS NO FUNCIONALES

ID	Descripción
RNF_1	El sistema debe estar protegido con JSON Web Token (JWT).
RNF_2	El sistema debe ser accesible desde los navegadores Chrome, Microsoft Edge, Opera y Firefox.

A2. Casos de uso

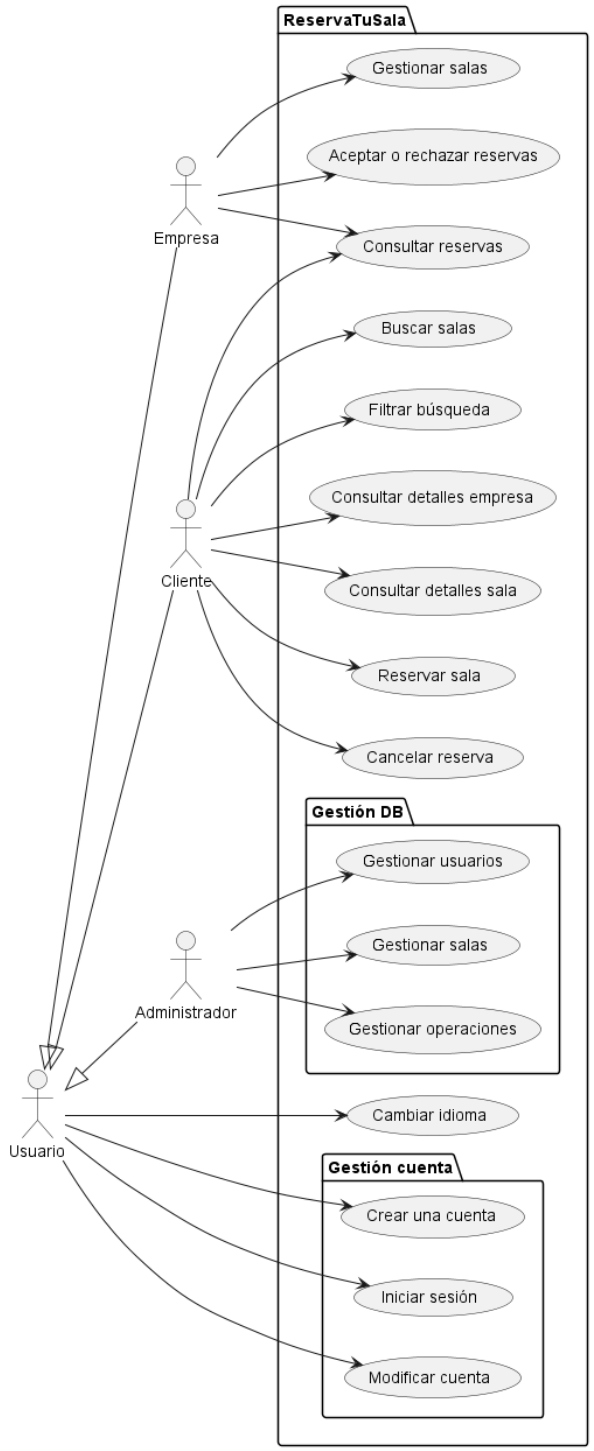

Fig. 27: Diagrama de casos de uso.

A3. Vistas en distintos navegadores

A3.1. Chrome

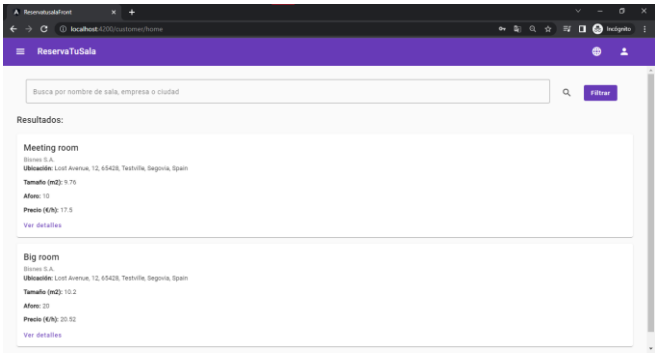

Fig. 28: Vista de inicio de cliente en Chrome.

A3.1. Microsoft Edge

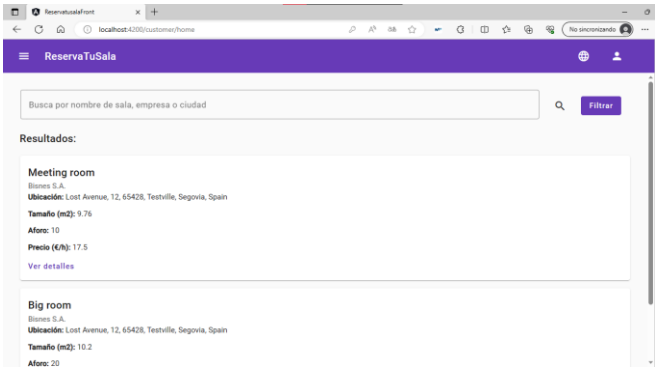

Fig. 29: Vista de inicio de cliente en Microsoft Edge.

A3.1. Opera

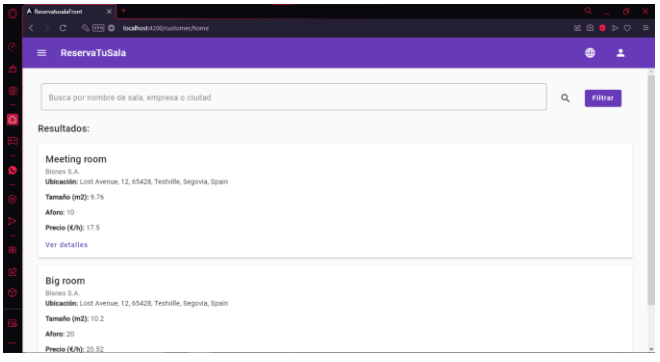

Fig. 30: Vista de inicio de cliente en Opera.

A3.1. FIREFOX

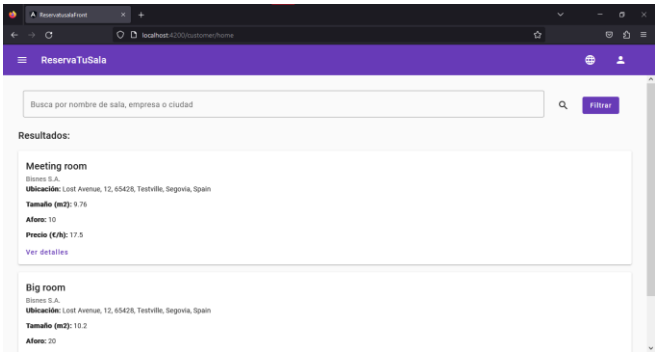

Fig. 31: Vista de inicio de cliente en Firefox.