BOOKING CLASSROOM

Aplicación Web de reserva de aulas para profesores

ÁLVARO BARREIRO ABAD
IES ABASTOS
Ciclo Superior en Desarrollo de Aplicaciones Multiplataforma
CURSO 2022/2023
GRUPO 7U
26 de junio de 2023
TUTOR: RAFAEL PUIG POZO

Contenido

1 PRESENTACIÓN

La gestión de la reserva de aulas en entornos educativos puede ser un desafío complejo. En muchos colegios e institutos, la organización interna se ve afectada por problemas relacionados con la asignación de espacios disponibles. Los profesores a menudo se encuentran con situaciones frustrantes, como la falta de aulas disponibles cuando necesitan realizar exámenes o actividades académicas. Para abordar esta problemática, se ha desarrollado una aplicación web llamada Booking Classroom.

Booking Classroom es una solución diseñada para facilitar la reserva de aulas por parte de los profesores. Esta aplicación ofrece un sistema centralizado que simplifica el proceso de reserva, permitiendo a los profesores acceder a la disponibilidad de aulas de manera eficiente y realizar sus reservas de forma sencilla.

La aplicación proporciona una interfaz intuitiva que permite a los profesores realizar reservas individuales y en un futuro permitirá programar reservas recurrentes. Además, ofrece una visión general en tiempo real de la disponibilidad de las aulas, evitando conflictos de horarios y optimizando la utilización de los espacios.

Al centralizar la información y agilizar el proceso de reserva de aulas, Booking Classroom ayuda a mejorar la organización interna de los colegios e institutos.

Por tanto, Booking Classroom tiene como objetivo ofrecer una solución práctica y eficiente para la gestión de la reserva de aulas en entornos educativos.

2 IDENTIFICACIÓN, JUSTIFICACIÓN Y OBJETIVOS DEL PROYECTO.

2.1 IDENTIFICACIÓN Y JUSTIFICACIÓN.

En entornos educativos como colegios e institutos, la gestión de la reserva de aulas presenta retos significativos. Uno de los problemas recurrentes es la falta de disponibilidad de aulas en momentos clave, como durante la planificación de exámenes o la organización de actividades académicas. Esta situación dificulta la programación eficiente de las actividades docentes y afecta negativamente la organización interna de las instituciones.

Para abordar esta problemática, se ha desarrollado un proyecto enfocado en la creación de una aplicación web llamada Booking Classroom. El objetivo principal de este proyecto es proporcionar una solución práctica y eficiente para la reserva de aulas, permitiendo a los profesores realizar reservas de manera sencilla y asegurando una programación eficiente de las actividades académicas.

2.2 OBJETIVOS.

Objetivos Funcionales:

1. Facilitar la reserva de aulas:
    - Proporcionar a los profesores una interfaz intuitiva y fácil de usar para realizar reservas de aulas.
    - Simplificar el proceso de reserva, eliminando la necesidad de comunicarse directamente con el personal administrativo o entre profesores.
    - Agilizar el acceso a la disponibilidad de las aulas, permitiendo a los profesores encontrar rápidamente espacios disponibles para sus actividades académicas.
2. Evitar conflictos de horarios:
    - Ofrecer una visión general en tiempo real de la disponibilidad de las aulas, evitando la superposición de reservas y conflictos de horarios.
    - Evitar que los profesores puedan reservar un aula si ya está ocupada para el horario deseado, garantizando una programación eficiente.
3. Administrar las aulas:
    - Los administradores de la aplicación podrán crear y gestionar las aulas disponibles en el sistema.
    - La información de las aulas y las reservas se almacenará en una base de datos, facilitando el acceso y la gestión para el personal administrativo de manera centralizada.
4. Informar sobre los espacios:
    - Proporcionar información sobre la capacidad y características de cada aula, permitiendo una asignación más precisa y adecuada según las necesidades

específicas de cada actividad. Por ejemplo, especificando si el aula dispone de proyector u ordenadores.

Objetivos Técnicos:

1. Implementar un sistema de autenticación utilizando Angular y Firebase:
    - Crear un sistema de registro y autenticación seguro para los profesores y el personal administrativo.
    - Integrar Firebase Authentication para la autenticación y autorización de usuarios.
    - Permitir la recuperación de la contraseña, así como, el cambio de la misma.
2. Diseñar y desarrollar una interfaz intuitiva en Angular:
    - Utilizar Angular para diseñar una interfaz de usuario atractiva y fácil de usar.
    - Implementar componentes y módulos reutilizables para agilizar el desarrollo y mejorar la mantenibilidad del código.
3. Utilizar Firebase Firestore como base de datos:
    - Diseñar y configurar una estructura de base de datos eficiente para almacenar información sobre las aulas, reservas y horarios.
    - Implementar operaciones CRUD (crear, leer, actualizar y eliminar) para gestionar la información de las aulas, así como, de las reservas de aulas.
    - Aprovechar las capacidades de consultas en tiempo real de Firestore para mostrar la disponibilidad de las aulas de manera dinámica.
    - Almacenar los datos de los usuarios que tengan rol de administrador.
4. Implementar funcionalidades de disponibilidad en tiempo real:
    - Utilizar Firestore para mostrar la disponibilidad de las aulas en tiempo real.
    - Actualizar automáticamente la información de disponibilidad cuando se creen nuevas aulas o se realicen nuevas reservas o se cancelen reservas existentes.
5. Configurar Firebase Hosting para el alojamiento de la aplicación:
    - Configurar Firebase Hosting para alojar la aplicación web y garantizar su disponibilidad en línea.

3 DISEÑO DEL PROYECTO

3.1 ANÁLISIS DE LAS TECNOLOGÍAS DISPONIBLES

En la fase de diseño del proyecto, es importante considerar las diferentes soluciones y herramientas disponibles en el mercado que nos permitirán desarrollar la aplicación Booking Classroom de manera efectiva. Además, es fundamental identificar las tecnologías que se pueden utilizar para ofrecer los servicios requeridos.

En cuanto a las soluciones y herramientas disponibles, existen varias opciones que pueden ser consideradas. Por ejemplo, hay plataformas y aplicaciones especializadas en la gestión de reservas y programación de espacios. Es cierto que algunas de las cuales pueden ser adaptadas y personalizadas para satisfacer las necesidades específicas de reserva de aulas en entornos educativos. Estas soluciones comerciales pueden ofrecer características avanzadas y estar respaldadas por un soporte técnico adecuado, pero por contrapartida tienen que suelen ser servicios de pago. Como ejemplo podemos citar a *clearooms*[1] o *kalena*[2].

Por otro lado, también se podría explorar el uso de herramientas como Google Calendar[3], aunque su adaptación para la gestión de reservas de aulas en entornos educativos podría resultar laboriosa y limitada en funcionalidad. Además, existe el riesgo de que la configuración manual genere errores y dificulte la eficiencia del proceso.

En este contexto, se ha optado por desarrollar una aplicación personalizada. Para ello en el mercado existen diferentes herramientas de desarrollo de software disponibles, como frameworks y bibliotecas, que pueden facilitar el proceso de desarrollo de la aplicación.

En el lado del front-end, algunas opciones populares son Angular, React y Vue.js, que son frameworks de desarrollo web ampliamente utilizados y que ofrecen un conjunto de herramientas y componentes para la creación de interfaces de usuario interactivas y modernas.

En el ámbito del back-end, se presentan dos opciones distintas: las bases de datos relacionales y las bases de datos NoSQL. En el caso de las bases de datos relacionales, se pueden utilizar tecnologías consolidadas como MySQL o PostgreSQL, en combinación con lenguajes de programación ampliamente adoptados como Java, Python o PHP. Estas tecnologías ofrecen un enfoque tradicional y robusto para la gestión y manipulación de datos estructurados.

Por otro lado, las bases de datos NoSQL ofrecen un enfoque alternativo y moderno. Entre las opciones más populares se encuentran Firebase Firestore, MongoDB y Cassandra. Estas bases de datos NoSQL proporcionan flexibilidad en el esquema de datos y permiten el almacenamiento y procesamiento eficiente de grandes volúmenes de información distribuida.

Después de analizar las diferentes opciones disponibles, hemos tomado la decisión de desarrollar la aplicación Booking Classroom utilizando Angular y Firebase. Estas opciones nos

[1] https://clearooms.com/capterra/?referrer=GetApp
[2] https://kalena.es/
[3] https://www.youtube.com/watch?v=h_mgc4JC95Q

brindan flexibilidad, escalabilidad y facilidad de integración. A continuación, justificaremos nuestra elección:

Angular:

- Amplia comunidad y soporte: Angular cuenta con una gran comunidad de desarrolladores activos y una extensa documentación. Esto nos brinda acceso a recursos valiosos, tutoriales, ejemplos y soluciones a problemas comunes. Además, al ser mantenido por Google, se beneficia de actualizaciones y mejoras constantes.
- Otra razón importante que respalda nuestra elección de Angular es su facilidad para crear templates y darles funcionalidad. Angular ofrece un enfoque claro y estructurado para la construcción de la interfaz de usuario, lo que nos permite separar de manera efectiva la lógica del negocio de la presentación visual. Utiliza el patrón de arquitectura MVC (Modelo-Vista-Controlador) y promueve las buenas prácticas de desarrollo, lo que nos ayuda a mantener un código limpio, estructurado y fácilmente mantenible.
- Otra de las razones para elegir este framework es que es el que estoy utilizando en las prácticas de empresa. Este hecho genera la posibilidad de adquirir experiencia y conocimiento en su uso.
- Angular proporciona diversas herramientas y componentes predefinidos, como Angular Material, que facilitan la creación de una interfaz de usuario interactiva y amigable. Además, Angular nos permite desarrollar componentes personalizados, los cuales pueden ser reutilizados, siendo muy fácil organizar la aplicación. Mediante el uso de componentes, podemos crear templates reutilizables y modulares que encapsulan la lógica y la apariencia de elementos específicos de la interfaz. Además, Angular proporciona enlaces de datos bidireccionales que facilitan la sincronización automática entre la vista y el modelo de datos subyacente.

Firebase:

- Autenticación de usuarios: Firebase proporciona un sólido sistema de autenticación de usuarios, lo que nos permite gestionar de manera segura el acceso y la identificación de los profesores que utilizan la aplicación. Con Firebase Authentication, podemos implementar opciones de inicio de sesión con correo electrónico, autenticación social y más.
- Base de datos en tiempo real: Firestore, la base de datos en tiempo real de Firebase, nos brinda una solución escalable y flexible para almacenar y sincronizar la información de las reservas de aulas. Esto garantiza que los datos estén siempre actualizados y disponibles para los profesores en tiempo real.
- Integración sencilla: Firebase se integra fácilmente con Angular, lo que facilita la configuración y el desarrollo de la aplicación. Además. Esta integración viene favorecida por el hecho de que ambas herramientas son proporcionadas por Google.
- Hosting: Además de lo mencionado, Firebase ofrece un servicio de hosting que nos permite desplegar fácilmente la aplicación y garantizar su disponibilidad en la web.

3.2 ANÁLISIS DE LAS FASES DEL PROYECTO

En esta fase, se identificarán los pasos clave necesarios para llevar a cabo el proyecto.

Fase 1: Análisis del problema.

Se realizará un análisis exhaustivo del problema a resolver y de los requisitos del proyecto. Se identificarán las necesidades de los usuarios y se definirán los objetivos a alcanzar. También se decidirá la tecnología a utilizar.

Fase 2: Definición de los datos necesarios.

Se determinarán los datos necesarios para el funcionamiento de la aplicación. Esto implicará identificar los tipos de información requeridos y cómo se almacenarán y gestionarán.

Fase 3: Diseño gráfico de las pantallas.

Se realizará el diseño visual de las interfaces de usuario de la aplicación. Se crearán los mockups o prototipos de las pantallas para definir la apariencia y la disposición de los elementos.

Fase 4: Estructura del proyecto.

Se establecerá una estructura organizada de carpetas y archivos para el desarrollo del proyecto. Esto facilitará la gestión y el mantenimiento del código.

Fase 5: Codificación.

Se procederá a la implementación de la lógica y funcionalidades de la aplicación.

Fase 6: Pruebas.

Se realizarán pruebas exhaustivas para verificar el correcto funcionamiento de la aplicación. Se identificarán y corregirán posibles errores y se asegurará que cumpla con los requisitos establecidos.

Fase 7: Despliegue.

Una vez que la aplicación haya pasado las pruebas, se procederá a su despliegue en un entorno de producción.

3.3 METODOLOGÍA DE TRABAJO

En cuanto a la hora de cómo enfocar y gestionar el trabajo, se ha elegido utilizar la metodología Agile en conjunto con Kanban.

La metodología Agile es un enfoque de gestión de proyectos que se basa en la colaboración, la adaptabilidad y la entrega incremental. En lugar de planificar exhaustivamente todo el proyecto desde el principio, Agile se centra en ciclos de desarrollo más cortos, conocidos como "iteraciones" o "sprints".

La elección de utilizar esta metodología se fundamenta en varias razones. En primer lugar, nos permite realizar entregas incrementales y frecuentes de funcionalidades, lo que nos brinda la oportunidad de obtener feedback temprano y realizar ajustes en el desarrollo del proyecto.

Esta flexibilidad permite adaptarnos a medida que avanzamos y garantiza que la solución se ajuste a las necesidades que se han detectado.

Otro aspecto importante de Agile es su enfoque iterativo e incremental. En lugar de esperar a tener la aplicación completa antes de realizar entregas, Agile propone la entrega de funcionalidades en ciclos cortos y frecuentes. Esto nos permite obtener resultados tangibles en cada iteración.

Por otro lado, Kanban es una metodología visual de gestión de proyectos y procesos que se centra en la visualización del flujo de trabajo y la optimización del rendimiento. Se basa en la idea de utilizar tableros visuales con tarjetas o notas adhesivas para representar las tareas y el progreso de un proyecto.

El método Kanban se originó en el ámbito de la fabricación, específicamente en Toyota, y luego se adoptó en el desarrollo de software y otros sectores. La palabra "Kanban" significa "tarjeta" o "etiqueta" en japonés, y hace referencia a las tarjetas utilizadas para representar las tareas en el tablero.

El tablero Kanban consta de columnas que representan las diferentes etapas del flujo de trabajo, como "Por hacer", "En progreso" y "Finalizado". Cada tarea se representa mediante una tarjeta o nota adhesiva, y se mueve de una columna a otra a medida que avanza en el proceso. Esto permite tener una visión clara y visual del estado de cada tarea y del flujo general del trabajo.

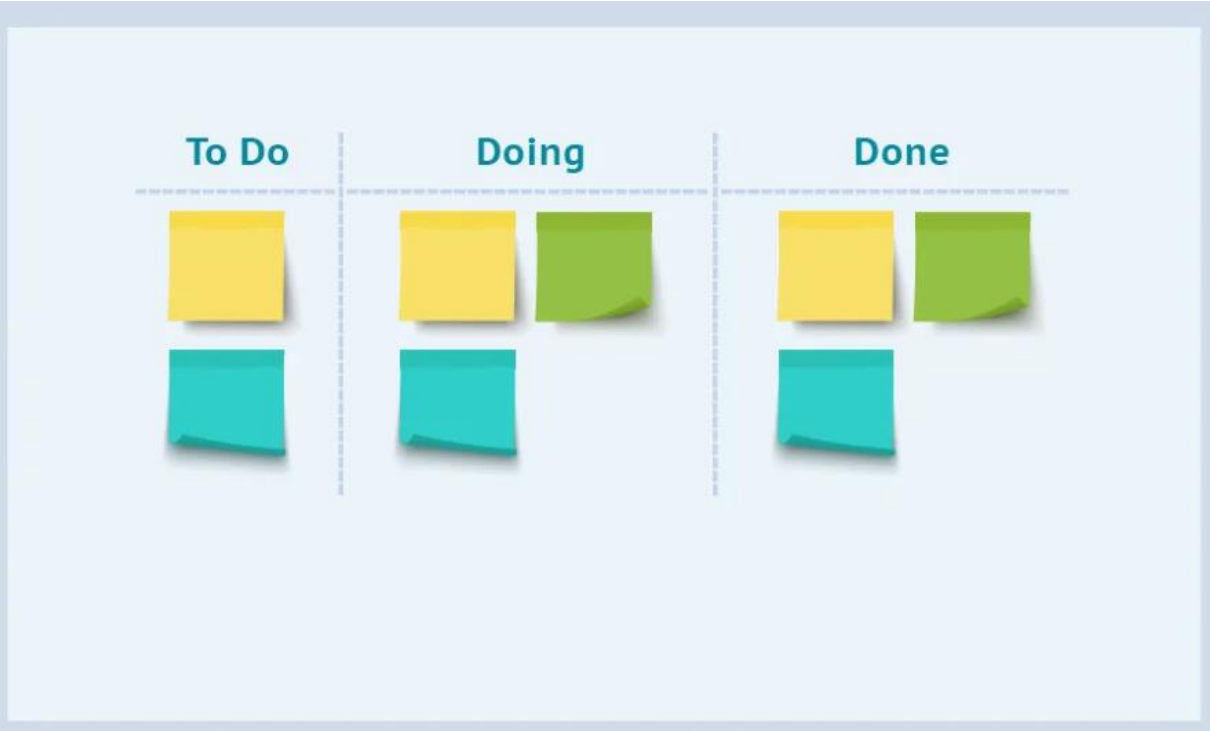

Ilustración 1. Tablero Kanban

Entre las tareas más importantes a llevar a cabo podemos destacar las siguientes:

- Análisis de problema.
- Determinar el modelo de datos.

- Diseño de bocetos o mockups de los elementos visuales de la aplicación.
- Crear los templates en Angular acordes a esos bocetos.
- Definir la estructura de la aplicación. Es decir, qué módulos vamos a necesitar y como va a ser la estructura de carpetas.
- Crear los servicios necesarios que permitirán conectar con la base de datos.
- Crear los formularios para que los usuarios puedan interactuar con la aplicación.
- Desarrollar el apartado de autenticación.
- Definir los roles de usuario.
- Establecer el Routing de la aplicación e implementar la técnica Lazy Loading.
- Probar la aplicación y corregir errores.
- Desplegar la aplicación.

3.4 MODELO DE DATOS

En este apartado se va a explicar qué datos se han necesitado y cómo se han organizado.

Los datos han sido organizados en colecciones dentro de Firebase Firestore. Una colección es un grupo lógico de documentos que comparten un tema o una entidad común. Cada documento dentro de una colección almacena los datos relacionados en formato de pares clave-valor.

Por otro lado, es importante destacar que se ha utilizado el enfoque de modelado de datos basado en interfaces en el código del proyecto. Esto implica que cada colección definida en Firestore tiene su equivalente en forma de interfaces en el código de la aplicación.

Usuarios:

Los usuarios serán los profesores del instituto. Éstos tendrán la capacidad de reservar y gestionar sus reservas. Se necesitarán pocos datos relativos a los mismos, ya que solo necesitamos el email como identificador del profesor y la contraseña para poder entrar. En este caso los datos se almacenan directamente en Firebase en Users de la herramienta Authentication.

- Interfaz

```
export interface Usuario {
  nombre: string;
  apellidos: string;
  email: string;
  password: string;
}
```

Administradores:

Los administradores de la aplicación serán aquellos usuarios cuyo correo electrónico esté incluido en esta colección. Los administradores tendrán las mismas funcionalidades que los usuarios normales, pero con el añadido de poder crear y eliminar las clases disponibles.

- Interfaz

```
export interface Administradores
{
  id: string;
  email: string;
}
```

- Colección

```
{
  "id": "string",
  "email": "string"
}
```

Aula:

En cada aula se almacenará las características y disponibilidad de la misma. Por ejemplo, el día de la semana que está libre, las horas o el aforo.

- Interfaz

```
export interface Aula {
  id: string;
  aula: string;
  aforo: number;
  hora_inicial: number;
  hora_final: number;
  ordenadores: string;
  proyector: string;
  dia: number;
}
```

- Colección

```
{
  "id": "string",
  "aula": "string",
  "aforo": number,
  "hora_inicial": number,
  "hora_final": number,
  "ordenadores": "string",
  "proyector": "string",
  "dia": number
}
```

Reservas:

Las reservas almacenarán la fecha para la cual se quiere reservar el aula y el email de la persona que hace la reserva, para saber quién la ha reservado. Además, también se incluirán los datos de cada aula en cada una de las reservas para que los usuarios tengan acceso a dichos datos.

- Interfaz

```
import { Aula } from
"./aula.model";

export interface Reservas {
  id: string
  email: string;
  fecha: number;
  aula: Aula
}
```

- Colección:

```
{
  "id": "string",
  "email": "string",
  "fecha": number,
  "aula": {
    "id": "string",
    "aula": "string",
    "aforo": number,
    "hora_inicial": number,
    "hora_final": number,
    "ordenadores": "string",
    "proyector": "string",
    "dia": number
  }
}
```

3.5 DISEÑO GRÁFICO DE LA INTERFAZ

En el proceso de desarrollo de la interfaz de nuestra aplicación, nos hemos enfocado en diseñar pantallas que sean intuitivas, atractivas y funcionales para brindar una experiencia de usuario satisfactoria. Para lograrlo, se han utilizado bocetos y prototipos que nos han permitido visualizar y evaluar diferentes opciones de diseño.

A continuación, mostraremos algunas de las pantallas más representativas de la aplicación. Empezaremos por la primera pantalla que ve el usuario al entrar en la aplicación.

Login:

- Boceto:

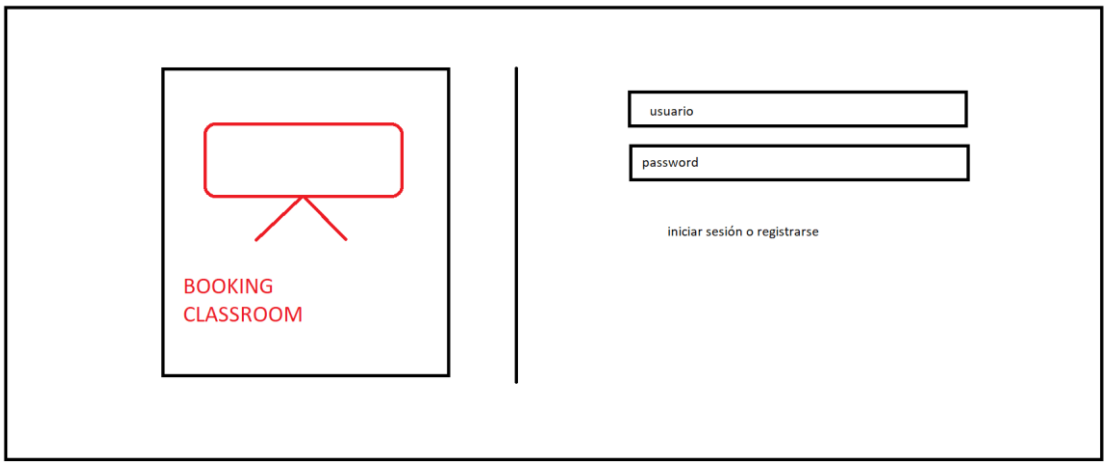

Ilustración 2. Boceto de la pantalla de Login

- Resultado final

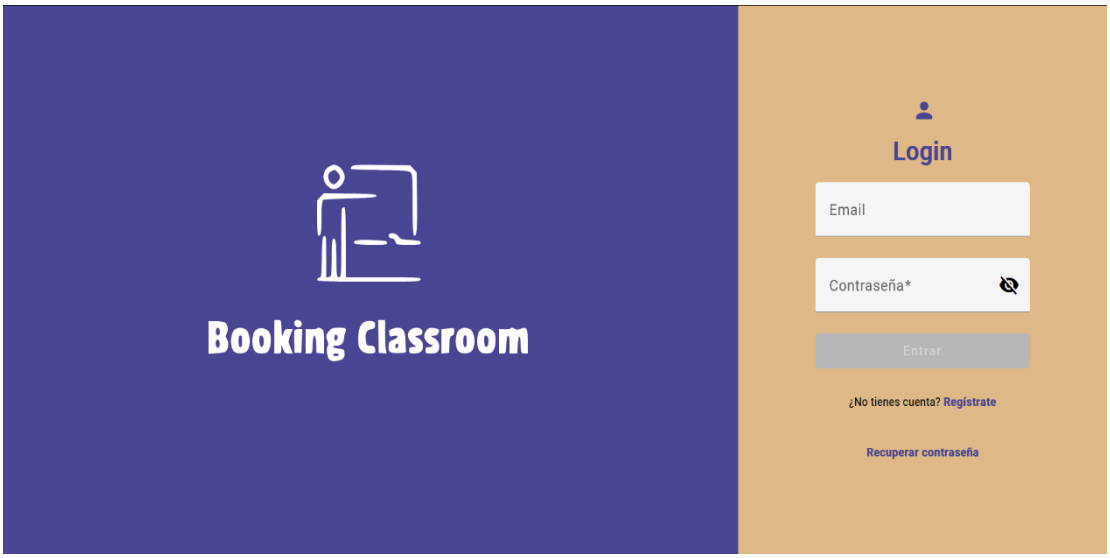

Ilustración 3. Pantalla de Login

Ésta es la pantalla de Login. En ella se le pedirá al usuario que ingrese su email y contraseña para poder ingresar a la aplicación.

En caso de no tener cuenta, tiene la opción de registrarse. Esta opción le llevará a una nueva pantalla en la que podrá ingresar sus datos en un formulario y crear su cuenta.

Por otro lado, si ha olvidado su contraseña, el usuario tendrá la opción de recuperarla introduciendo su email en el campo del formulario. Éste recibirá un correo electrónico en el que tendrá un enlace a través del cual podrá cambiar su contraseña.

Pantalla principal:

- Boceto

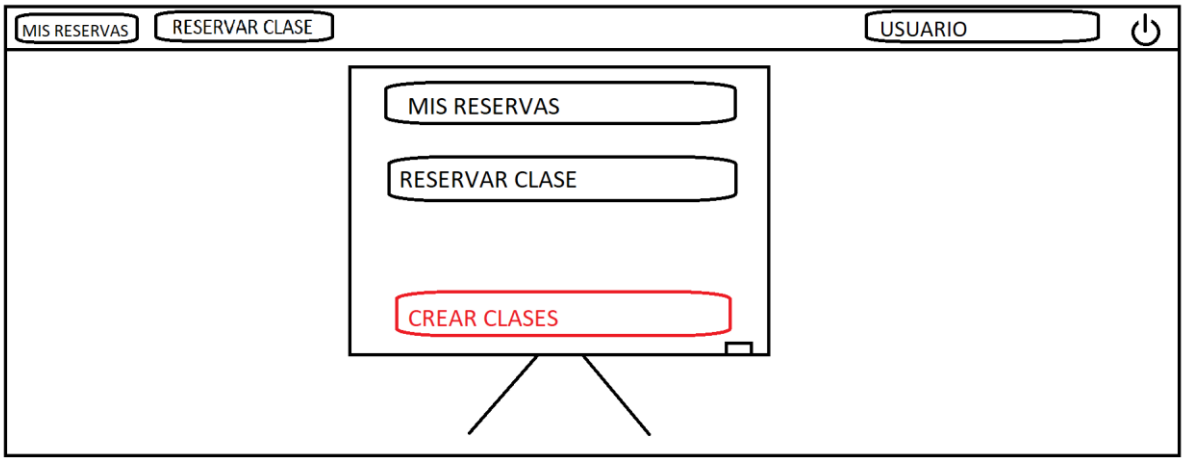

Ilustración 4. Boceto de la pantalla principal

- Resultado final

Ilustración 5. Pantalla principal

Esta es la pantalla principal de la aplicación. Una vez hecho el login correctamente, está es la primera pantalla que verá el usuario. Hay que matizar que el botón de 'Administrar Clases' solo estará disponible si se accede con rol de administrador.

Se puede observar que el resultado final difiere un poco del boceto. Esto es debido a que una vez creada la pantalla se decidió que no tenía sentido duplicar funcionalidades y se cambiaron

los botones de arriba por el nombre del usuario que esté usando la aplicación en ese momento. Consiguiendo, así, una experiencia más personalizada.

Vemos que, desde aquí, el usuario, tendrá acceso a toda la funcionalidad de la aplicación.

Primero explicaremos la barra de navegación.

El primer botón que vemos es un botón que nos lleva al home. Es decir, a esta pantalla principal. Es un botón que se encuentra disponible en todas las pantallas para poder volver al inicio cuando se desee.

El segundo botón que vemos es el de usuarios. Éste nos permitirá cambiar la contraseña de acceso a la aplicación.

Por último, encontramos el botón de logout, el cual, desconecta la sesión y nos redirige a la pantalla de login.

En cuanto a la botonera central, se puede decir que aquí se encuentra la parte funcional de la aplicación. Encontramos tres botones.

El primero de ellos, será utilizado para ver las reservas que ha realizado el usuario. Él mismo, podrá borrar aquellas a las que no vaya a poder asistir o que haya reservado por error.

El botón central, es el botón que se utilizará para reservar las aulas. Redirigirá al usuario a una pantalla en la que encontrará un formulario que le servirá de filtro para seleccionar el aula disponible que se adecue a sus necesidades.

Por último, encontramos el botón de 'Administrar Clases'. Como se ha comentado anteriormente, éste sólo será visible por aquellos usuarios que tengan el rol de administrador. Únicamente estos usuarios tendrán la capacidad de crear los espacios disponibles y mantenerlos actualizados.

Administrar Clases:

- Boceto

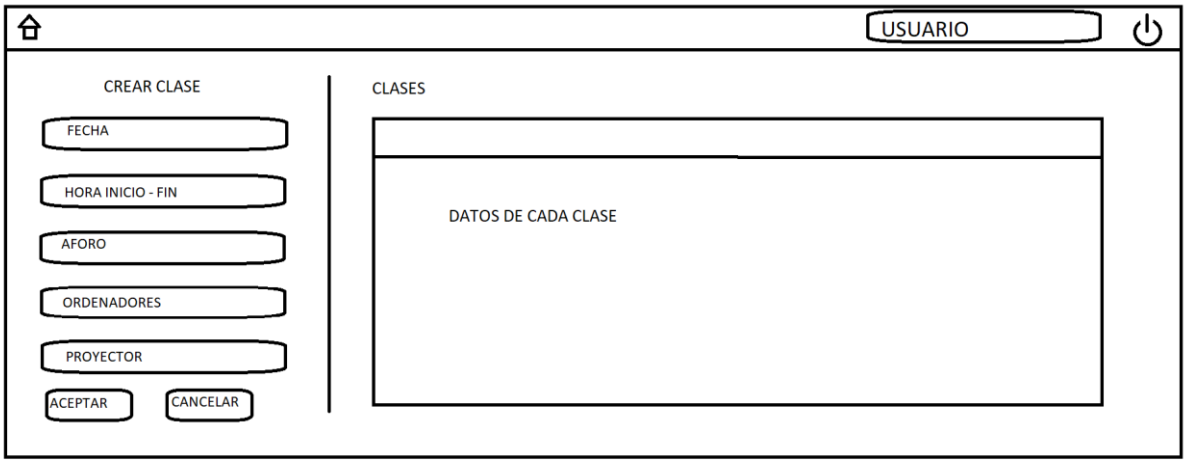

Ilustración 6. Boceto de la pantalla Administrar Clases

- Resultado final

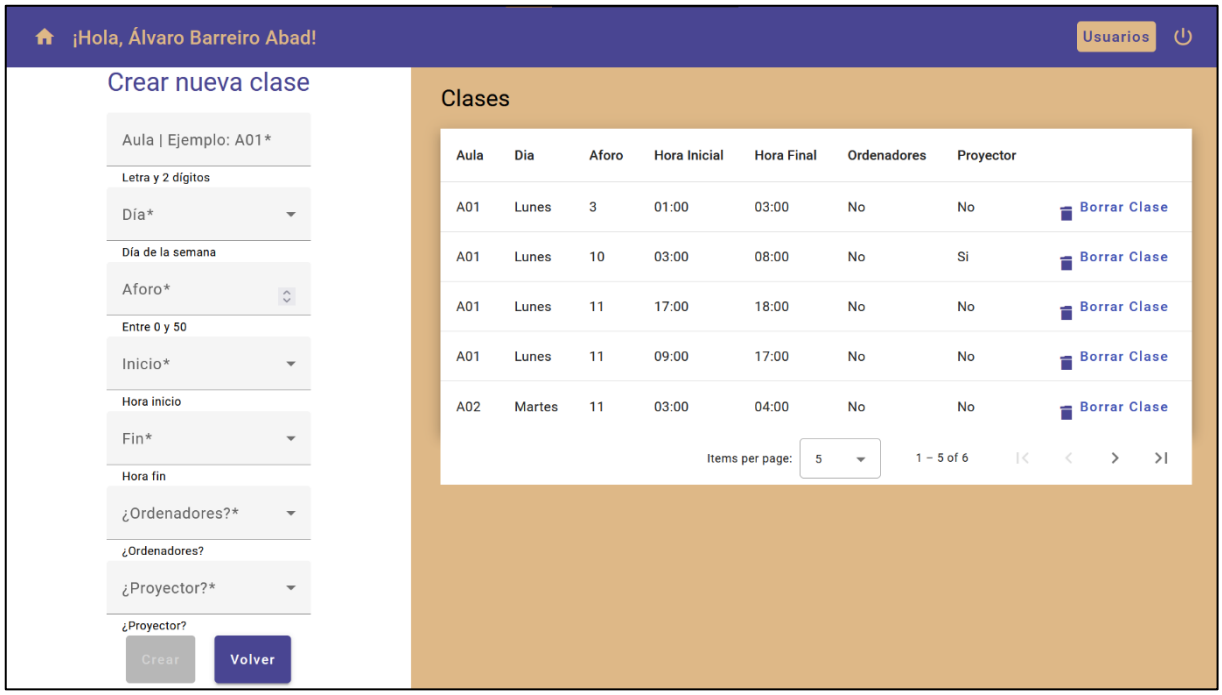

Ilustración 7. Pantalla Administrar Clases

Por último, se describirá la pantalla Administrar Clases por ser la que más funcionalidades tiene.

Como ya se ha dicho, a este recurso solo tendrán acceso los administradores.

Se puede observar a la izquierda un formulario que sirve para la creación de una nueva clase. En él se proporcionará las características de la clase. Vemos que todos los campos son obligatorios y hasta que no se completen todos no se activará el botón ‘Crear’.

En la parte derecha, se puede ver una tabla en la que aparecerán todas las clases existentes en el sistema, con sus respectivas características. También se podrán borrar las clases que ya no estén disponibles gracias al botón ‘Borrar Clase’.

Por último, hemos añadido un “paginador” al final de la tabla para facilitar la visualización de las aulas disponibles. El “paginador” es un elemento que permite dividir la lista de aulas en varias páginas, mostrando un número limitado de registros por página. Esto ayuda a evitar la saturación de información y permite al usuario navegar de forma más ágil entre las diferentes páginas de aulas.

3.6 ESTRUCTURA DEL PROYECTO

Angular es un framework de desarrollo web que se basa en el concepto de componentes. En su arquitectura, la aplicación se estructura en torno a dichos componentes los cuales son reutilizables. Éstos encapsulan la lógica y la presentación de una parte específica de la interfaz de usuario. Estos componentes permiten dividir la aplicación en unidades más pequeñas y manejables, lo que facilita el desarrollo, la mantenibilidad y la reutilización del código.

Además de los componentes, Angular utiliza otro tipo de componentes o módulos adicionales que sirven de apoyo al resto de componentes de la aplicación. Por ejemplo, entre otros, podemos encontrar servicios, el módulo de Routing y otro tipo de módulos o carpetas que habitualmente se utilizan para organizar y modularizar la aplicación, como por ejemplo la carpeta “shared”.

Los servicios en Angular permiten separar la lógica de negocio y los datos compartidos, lo que promueve la reutilización y la organización del código. Estos servicios se encargan de proporcionar funcionalidades específicas que pueden ser compartidas entre múltiples componentes de la aplicación.

Por otro lado, el Routing en Angular facilita la navegación entre las diferentes vistas de la aplicación. Mediante la configuración de rutas, es posible definir las URLs correspondientes a cada vista y establecer la lógica de navegación.

El uso de la carpeta "shared" ayuda a mantener un código más limpio, organizado y facilita la reutilización de recursos en toda la aplicación En esta carpeta, se suelen colocar aquellos elementos que no están directamente relacionados con un componente o módulo en particular, pero que son necesarios en diferentes partes de la aplicación.

Teniendo estos conceptos claros podemos definir la estructura del proyecto acorde al siguiente esquema:

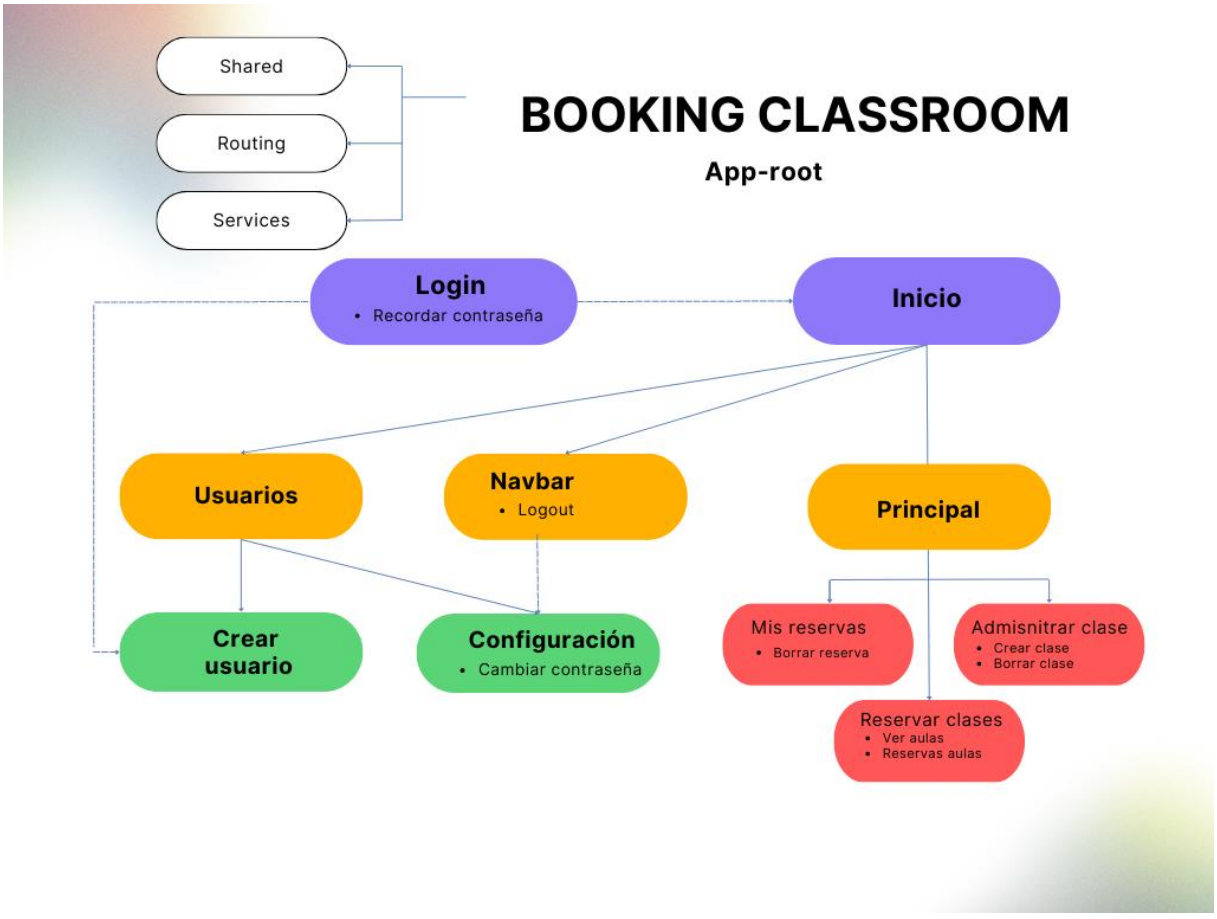

Ilustración 8. Estructura de la aplicación

En el esquema se puede observar que el proyecto se organiza como si de una muñeca rusa se tratara. Es decir, cada componente contiene a otro u otros componentes y así sucesivamente, creando una jerarquía entre ellos.

El componente principal y más importante es “app-root”. Este componente es el que contiene a todos los demás. En toda aplicación Angular ha de existir como mínimo dicho componente.

En nuestro caso, el “app-root” contiene a “Login” y al componente “Inicio”, el cual contiene al resto de elementos. A este último únicamente se puede acceder una vez el usuario se ha autenticado correctamente.

Desde el “Login” tenemos acceso al componente “Crear Usuarios” el cual pertenece al componente “Usuarios”, y por otro lado, también tenemos la funcionalidad de recordar la contraseña en caso de haberla olvidado.

En cuanto al componente “Inicio” vemos que hace de contenedor de los demás. El más especial de los que contiene es el “NavBar” o barra de navegación, ya que aparecerá visualmente en conjunto con todos los demás, sirviendo de cabecera. Tiene la funcionalidad, de acceder al menú de inicio, de hacer logout y también sirve para acceder al componente “Configuaración” el cual permite cambiar la contraseña a los usuarios.

Por último, tenemos los tres componentes que aparecen en la botonera de la pantalla principal: “Mis reservas”, “Reservar Clases” y “Crear Clases”. A estos se acceden a través de un componente llamado “Principal” que sirve para hacer de enlace únicamente.

4 DESARROLLO DEL PROYECTO

4.1 INSTALACIÓN DEL ENTORNO

Una de las cosas a tener en cuenta antes de empezar a desarrollar código es tener el entorno correctamente instalado. En general, contar con un entorno de desarrollo bien configurado antes de comenzar a codificar proporciona una base sólida y permite trabajar de manera más eficiente, evitar problemas y aprovechar al máximo el tiempo disponible.

Lo primero que instalaremos es el IDE. En este caso se ha optado por la opción de **Visual Studio Code** por la experiencia adquirida en el uso del mismo en el desarrollo de otros proyectos y también porque cuenta con numerosas extensiones para los diferentes frameworks que se van a utilizar. Estas extensiones son complementos que mejoran el editor de código al proporcionar funcionalidades adicionales. Son clave para ampliar capacidades, adaptarse a diferentes lenguajes y tecnologías, y sobre todo mejorar la productividad. A continuación, se muestran algunas de las extensiones utilizadas:

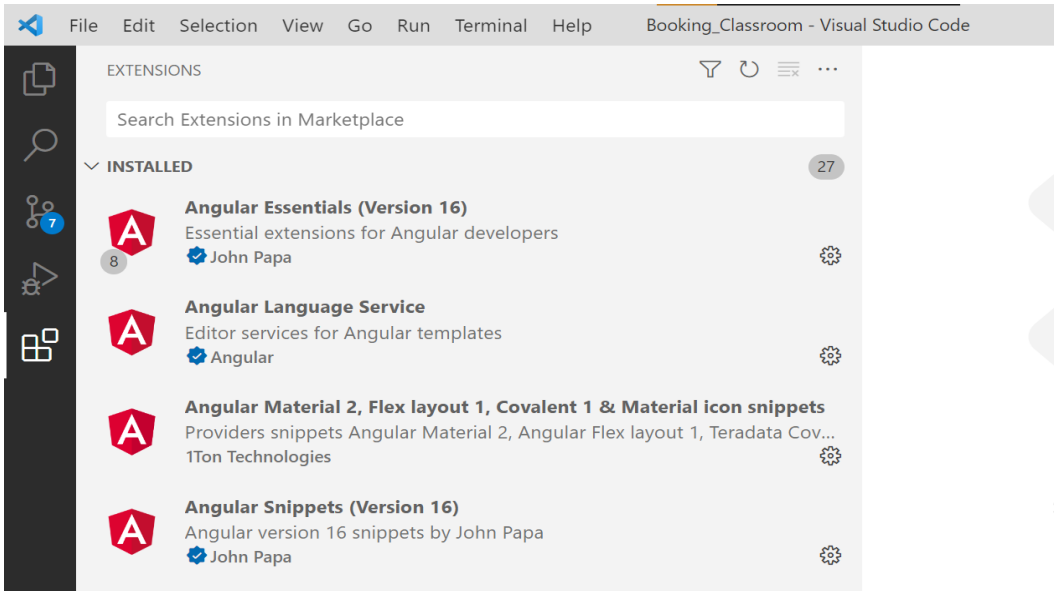

Ilustración 9. Extensiones de Visual Studio Code

Una vez instalado el IDE se ha de instalar el software necesario para desarrollar la aplicación.

En nuestro caso, lo primero que necesitaremos instalar es **Node.js** ya que Angular requiere del mismo. Node.js es un entorno de ejecución de JavaScript basado en el motor de JavaScript V8 de Chrome, es decir, permite ejecutar código JavaScript fuera del navegador.

Junto con Node.js se instala "**npm**". Node Package Manager es el sistema de gestión de paquetes para Node.js. Es una herramienta que permite instalar, actualizar y administrar las dependencias de un proyecto de Node.js.

Después de instalar Node.js y "npm" ya podemos instalar **Angular**. Lo primero que se hará será instalar Angular CLI (Command Line Interface) la cual es una herramienta de línea de comandos que facilita la creación y gestión de proyectos de Angular. Para instalar Angular

CLI, se abrirá una terminal o línea de comandos y se ejecutará el siguiente comando: “npm install @angular/cli”.

Una vez instalado Angular CLI, se creará el nuevo proyecto de Angular utilizando el siguiente comando: “ng new Booking-Classroom”.

Después de instalado, lo único que nos queda es arrancarlo y para ello se ejecutará el siguiente código: “ng serve”. Este comando iniciará el servidor de desarrollo y se podrá ver la aplicación en el navegador web visitando “http://localhost:4200”.

Por último, solo faltará por instalar **Firebase**. Para ello, iniciaremos sesión con una cuenta de Google en el navegador. Después, accederemos a (https://console.firebase.google.com/) y crearemos un nuevo proyecto en “Agregar Proyecto”. A medida que se avance con el proyecto se instalarán las librerías necesarias. Se comentará más adelante cuando se expliquen los servicios.

4.2 CREACIÓN DE COMPONENTES

En este momento ya se estará en condiciones de empezar a desarrollar todos los componentes de la aplicación. Así pues, siguiendo la estructura antes mencionada procederemos a crear cada componente dentro de la carpeta que le corresponda dentro de esa estructura.

Para crear un componente nuevo se utilizará el siguiente comando: “ng generate component ‘nombre-del-componente’” y si queremos que el componente se sitúe dentro de una carpeta en concreto se usará el siguiente comando: “ng generate component ‘carpeta/nombre-del-componente’”.

Como resultado, después de crear un nuevo componente, se obtendrá algo como lo siguiente:

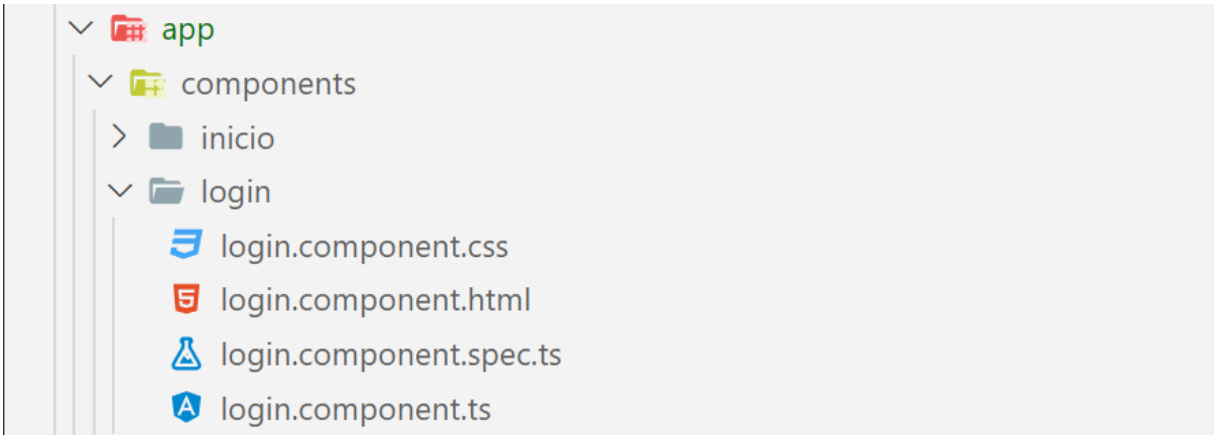

Ilustración 10. Ejemplo de componente

Aquí se puede apreciar que el componente “login” está dentro de una carpeta llamada “components”, en la que se almacenarán todos los componentes de la aplicación (esto no es obligatorio, pero conviene proceder de esta manera para mantener el código ordenado) que a su vez se encuentra dentro de la carpeta “app”, la cual contiene al componente raíz, quien a su vez hace de contenedor del resto de componentes.

Dentro del componente “login” se observan cuatro archivos. Estos son creados por Angular automáticamente en la creación de este. Esta estructura permite que cada componente tenga una funcionalidad específica y aislada del resto, así como, una vista propia.

Resumiendo, la utilidad de cada archivo que vemos en la imagen, tenemos lo siguiente:

- Archivo del componente (.component.ts): Este archivo contiene la lógica del componente, incluyendo su clase, propiedades, métodos y ciclo de vida. Aquí se define cómo se comporta el componente y cómo interactúa con los datos y eventos.
- Archivo de la plantilla (.component.html): En este archivo se define la estructura y contenido HTML del componente. Aquí se colocan los elementos visuales y se establecen las vinculaciones de datos con la lógica del componente.
- Archivo de estilos (.component.css o .component.scss): Este archivo contiene las reglas de estilo CSS o SCSS específicas del componente. Aquí se definen los estilos visuales y la apariencia del componente.
- Archivo de pruebas (.component.spec.ts): Este archivo se utiliza para escribir pruebas unitarias del componente. Aquí se definen escenarios de prueba y se verifica que el componente funcione correctamente.

Estos son los archivos principales generados al crear un componente en Angular. Cada archivo desempeña un papel importante en la estructura y funcionalidad del componente, permitiendo separar claramente la lógica, la presentación y los estilos.

4.3 MÓDULOS PRINCIPALES DE LA APLICACIÓN

En una aplicación Angular, el módulo raíz, conocido como “AppModule”, es esencial. Este módulo actúa como el punto de entrada principal de la aplicación y proporciona la configuración inicial necesaria. Es en “AppModule” donde se definen y se importan los demás módulos y componentes de la aplicación.

A medida que se agregan características y funcionalidades adicionales a la aplicación, es recomendable organizarlas en módulos separados. Cada módulo puede agrupar

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';

/* the AppModule class with the @NgModule decorator */
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

componentes, servicios y otros recursos relacionados. Esto ayuda a mantener una estructura modular y escalable en la aplicación, facilitando la gestión y el mantenimiento a medida que la aplicación crece.

En nuestro caso se ha añadido un módulo adicional a “AppModule”. Este es “InicioModule”. Aquí se agrupan todos los componentes de la aplicación a excepción del componente de “Login”. La razón de ser de esta organización es que, por un lado, seguimos las mejores prácticas de Angular para modularizar la aplicación y mejorar la estructura y el mantenimiento del código, y por otro, podemos aprovecharnos de técnicas como la de carga en diferido o Lazy Loading, donde los módulos se irán cargando a medida que se necesiten.

Cuando se crea un nuevo módulo, se utiliza la anotación “NgModule” para definir sus propiedades y configuraciones. Algunas de las propiedades más comunes que se definen en un módulo son:

- Declarations: Aquí se especifican los componentes, directivas y tuberías que pertenecen al módulo.
- Imports: Se utilizan para importar otros módulos que se necesitan en el módulo actual. Estos pueden ser módulos internos de la aplicación o módulos externos proporcionados por bibliotecas o paquetes.
- Providers: Aquí se definen los servicios y proveedores de datos que estarán disponibles en todo el módulo y sus componentes.
- Bootstrap: Se utiliza en el módulo raíz (AppModule) para indicar el componente principal que se iniciará al arrancar la aplicación.

4.4 CREACIÓN DE LA CARPETA SHARED

La carpeta “shared” en Angular es una convención comúnmente utilizada para almacenar archivos compartidos entre diferentes módulos y componentes de una aplicación. Es una práctica recomendada para organizar y reutilizar código de manera eficiente.

La principal ventaja de esta carpeta es que permite centralizar y mantener de forma ordenada el código que es utilizado en varios lugares de la aplicación. Esto facilita su gestión, actualización y reutilización, ya que los componentes y módulos pueden importar y utilizar estos archivos compartidos sin necesidad de duplicar el código en cada ubicación.

Además, al tener una ubicación específica para los archivos compartidos, se mejora la legibilidad y el mantenimiento del proyecto, ya que es más fácil encontrar y entender el propósito de estos archivos.

En nuestro caso, hemos utilizado para importar todos los módulos de Angular Material que hemos ido necesitando. Angular Material es una biblioteca de componentes y estilos visuales para Angular. Proporciona una amplia gama de componentes predefinidos que se pueden utilizar para crear interfaces de usuario modernas y atractivas de manera rápida y sencilla. La biblioteca Angular Material se basa en los principios del Material Design, una guía de diseño desarrollada por Google. El Material Design se centra en la usabilidad, la estética y la consistencia visual, lo que ayuda a crear aplicaciones con una apariencia y experiencia de usuario coherentes.

Tanto los módulos de Angular Material como los módulos adicionales importados son exportados por el “SharedModule”. Esto significa que otros módulos que importen el módulo podrán utilizar estos módulos y componentes sin necesidad de importarlos individualmente. En nuestro caso ha sido importado por “AppModule”, que es el módulo raíz, por lo que el “SharedModule” estará disponible para todos los componentes de la aplicación.

Además de los módulos y configuraciones mencionados anteriormente, también podríamos haber incluido en el “SharedModule” el componente “navBar” y la carpeta de servicios. Sin embargo, no se incluyeron porque son elementos específicos de ciertos módulos.

4.5 CREACIÓN DE LAS RUTAS O ROUTING

El enrutamiento o routing en Angular se refiere a la capacidad de definir y gestionar las rutas de navegación en una aplicación web. Es el proceso de asociar componentes y vistas con URL específicas, permitiendo la navegación entre diferentes páginas o secciones de la aplicación sin tener que recargar toda la página. Esto mejora significativamente la experiencia de usuario al permitir una transición suave entre secciones de la aplicación.

Además, el enrutamiento facilita el mantenimiento del estado de la aplicación. Cada vista tiene su propia URL, lo que permite compartir enlaces directos a secciones específicas y mantener el estado actual de la aplicación. Esto es especialmente útil en aplicaciones complejas donde los usuarios pueden volver atrás o compartir enlaces a páginas específicas.

Otra ventaja del enrutamiento es la separación de responsabilidades. Al definir rutas y componentes asociados, se establece una estructura clara y modular para la aplicación. Esto facilita el mantenimiento a largo plazo.

El enrutamiento en Angular también permite la carga dinámica de módulos o lo que también se conoce como “Lazy Loading”, lo que significa que los módulos se cargan bajo demanda según sea necesario. Esto mejora el rendimiento general de la aplicación al reducir los tiempos de carga inicial y optimizar el uso de recursos.

En términos de seguridad, el enrutamiento también es útil, ya que se pueden implementar mecanismos de autenticación y autorización para proteger determinadas rutas o funcionalidades de la aplicación. Esto ayuda a garantizar que solo los usuarios autorizados puedan acceder a ciertas partes de la aplicación.

En nuestra aplicación se tienen dos archivos de enrutamiento que trabajan juntos para proporcionar un enrutamiento completo. El archivo “app-routing.module.ts” maneja el enrutamiento principal de la aplicación, mientras que “inicio-routing.module.ts” se encarga del enrutamiento específico para el módulo “InicioModule”. Vamos a ver qué hace cada uno:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './components/login/login.component';
const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  {
    path: 'inicio',
    loadChildren: () =>
      import('./components/inicio/inicio.module').then((m) => m.InicioModule),
  },
  { path: '**', redirectTo: 'login', pathMatch: 'full' },
];

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

Lo primero que se deberá hacer es importar las clases “RouterModule” y “Routes” desde @angular/router.

La clase “**RouterModule**” es una clase que proporciona las funcionalidades principales para el enrutamiento en Angular. Permite configurar las rutas de la aplicación y proporciona directivas y servicios relacionados con el enrutamiento.

Por otro lado, la clase “Routes” es una interfaz que se utiliza para definir las rutas de una aplicación. Se trata de una matriz de objetos de ruta donde cada objeto de ruta contiene la configuración para una ruta específica.

A continuación, se define una constante llamada “routes” que contiene una matriz de objetos de tipo “Routes” y como hemos dicho, cada objeto de ruta representa una ruta específica de nuestra aplicación.

La primera ruta definida es una redirección vacía ('') que nos llevará al componente “LoginComponent” cuando la URL coincida exactamente con el patrón vacío. Esto significa que cuando los usuarios accedan al inicio de nuestra aplicación, serán redirigidos automáticamente a la página de inicio de sesión.

Luego, se define una ruta llamada “login” que se asocia al componente “LoginComponent”. Esto significa que cuando los usuarios accedan a la URL “/login”, se mostrará nuestro componente de inicio de sesión.

La siguiente ruta es “inicio”, donde se utiliza la técnica de carga diferida (lazy loading). Esta técnica mejora el rendimiento de nuestra aplicación porque solo se cargará el código específico de la página “inicio” cuando los usuarios la soliciten. No tendremos que cargar todo el código de la aplicación al principio, lo que reduce el tiempo de carga inicial y mejora la experiencia del usuario.

Por último, se define una ruta comodín ('**') que redirige a “login” cuando la URL no coincida con ninguna de las rutas definidas anteriormente. Esto significa que, si los usuarios intentan acceder a una URL no válida, serán redirigidos nuevamente a la página de inicio de sesión.

Por su parte el archivo “**inicio-routing.module.ts**” es el archivo de enrutamiento específico para el módulo “InicioModule”. Como antes, también importamos las clases “RouterModule” y “Routes”. Además de éstas, en este caso será necesario importar funciones relativas a la seguridad. Estas son: importamos las funciones “canActivate” y “redirectUnauthorizedTo” del módulo “@angular/fire/auth-guard”.

Dentro de la clase, se definirá una constante llamada “redirectUnauthorizedToLogin”, que es una función que se utilizará para redirigir a los usuarios no autorizados al componente de inicio de sesión (login).

A continuación, definimos las rutas en la constante “routes”, que funciona de la misma manera que en la clase analizada anteriormente.

La primera ruta es una redirección vacía ('') que redirige a la ruta “principal” cuando los usuarios acceden a la URL base del módulo “InicioModule”.

Dentro del componente “InicioComponent”, que actúa como un contenedor para los componentes secundarios del módulo, definimos las rutas secundarias utilizando la propiedad children. Cada ruta secundaria tiene una ruta (path) y un componente asociado. Por ejemplo, la ruta “principal” utiliza el componente “PrincipalComponent” y tiene la protección de autenticación activada gracias a la función “canActivate”. Esta función comprobará si el usuario ha iniciado sesión correctamente y si no lo ha hecho bloqueará el acceso a estos componentes redirigiendolo a la página de “login” si intenta entrar. Otras rutas como “mis-reservas”, “reservar-clase”, “crear-usuario”, “**configuración**” y “crear-clase” también tienen sus componentes correspondientes y, en algunos casos, la protección de autenticación activada.

```
import …

const redirectUnauthorizedToLogin = () => redirectUnauthorizedTo(['login']);

const routes: Routes = [
   { path: '', redirectTo: 'principal', pathMatch: 'full' },
  {
    path: '',
    component: InicioComponent,
    children: [
      {
        path: 'principal',
        component: PrincipalComponent,
        ...canActivate(redirectUnauthorizedToLogin),
      },
      {
        path: 'mis-reservas',
        component: MisReservasComponent,
        ...canActivate(redirectUnauthorizedToLogin),
      },
                              .
                              .
                              ... resto de paths
    ],
  },
];

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

Por último, Importamos “RouterModule.forChild(routes)”. Al utilizar “forChild(routes)”, indicamos que estas rutas son rutas secundarias y pertenecen a un módulo secundario. Esto permite que el enrutador de Angular registre y administre las rutas específicas del módulo “InicioModule” junto con las rutas de otros módulos.

4.6 CREACIÓN DE LOS SERVICIOS

En Angular, los servicios son clases que se utilizan para encapsular la lógica y la funcionalidad compartida que puede ser utilizada en diferentes componentes de la aplicación. Los servicios permiten centralizar y reutilizar el código en toda la aplicación, lo que promueve la modularidad y la separación de responsabilidades.

Los servicios se crean con la finalidad de proporcionar una funcionalidad específica, como la comunicación con una API, el manejo de datos, la autenticación de usuarios, la manipulación del estado de la aplicación, entre otros. Estos servicios pueden ser inyectados en los componentes, directivas u otros servicios que los necesiten.

Al utilizar servicios, se promueve el principio de "Dependency Injection" (Inyección de Dependencias) de Angular. Esto significa que en lugar de que los componentes creen

instancias de los servicios que necesitan, Angular se encarga de proporcionar las instancias de los servicios necesarios a los componentes que los solicitan. Esto facilita la gestión de dependencias y permite una mayor flexibilidad y reutilización del código.

Además, los servicios pueden ser utilizados para compartir datos y comunicación entre componentes que no tienen una relación directa, a través de la creación de servicios compartidos o servicios de eventos.

En nuestro caso, se han utilizado tres servicios diferentes:

- "MostrarNavbarService": se encarga de mostrar la barra de navegación.
- "ClasesService": se encarga de la creación de las aulas y las reservas.
- "AuthService": se encarga de la autenticación.

4.6.1 MostrarNavbarService

Este servicio se utiliza para controlar la visibilidad de la barra de navegación en los componentes que lo requieran.

```
import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable } from 'rxjs';

@Injectable({providedIn: 'root'})
export class MostrarNavbarService {

  private mostrarNavBarSubject = new BehaviorSubject<boolean>(false);

  setMostrarNavBar(valor: boolean) {
    this.mostrarNavBarSubject.next(valor);
  }

  getMostrarNavBar(): Observable<boolean> {
    return this.mostrarNavBarSubject.asObservable();
  }
}
```

El servicio utiliza la clase "BehaviorSubject" de la biblioteca "RxJS" para almacenar el estado actual de la barra de navegación. "BehaviorSubject" es un tipo de Observable que mantiene y emite el último valor emitido a los nuevos suscriptores.

En el servicio, se crea una instancia privada de BehaviorSubject<boolean> llamada mostrarNavBarSubject con un valor inicial de false. Este valor representa el estado inicial de la barra de navegación, es decir, que no se muestra.

El servicio proporciona dos métodos públicos: setMostrarNavBar() y getMostrarNavBar().

- setMostrarNavBar() se utiliza para actualizar el valor de mostrarNavBarSubject con un nuevo valor booleano que indica si se debe mostrar o no la barra de navegación. Este método se llama desde otros componentes para cambiar el estado de la barra de navegación.

- getMostrarNavBar() devuelve un Observable que permite a los componentes suscribirse y recibir actualizaciones sobre el estado actual de la barra de navegación. Los componentes pueden utilizar este Observable para reaccionar y actualizar su interfaz de usuario en función del estado de la barra de navegación.

El servicio está decorado con @Injectable({providedIn: 'root'}), lo que significa que se proporcionará una instancia única y global del servicio en toda la aplicación. Esto permite que los componentes accedan al mismo servicio y compartan el estado de la barra de navegación de manera consistente.

4.6.2 ClasesService

Antes de poder desarrollar este servicio se deberá crear una base de datos en Firebase. También se necesitará instalar las librerías necesarias para darle a Angular las herramientas necesarias para poder interactuar con la base de datos.

Para ello seguiremos los siguientes pasos:

- Se accederá a Firebase y haremos login con la cuenta de Google que queramos utilizar.
- Se creará el proyecto y una base de datos desde la consola de Firebase.
- Iniciaremos la aplicación en la terminal situada dentro del proyecto de Angular con el comando “firebase init” y se seleccionará que queremos utilizar Firestore. Después lo enlazaremos con nuestro proyecto.
- Después se instalarán las librerías que permitirán lanzar acciones contra la base de datos. Se instalará con el siguiente comando “ng add @angular/fire”. Una vez instalado todo correctamente, se generará el siguiente archivo, el cual, tiene las credenciales que nos permitirán conectarnos con nuestra aplicación Firebase.

```
export const environment = {
  firebase: {
    projectId: 'booking-classroom-4f209',
    appId: '1:686562888488:web:39c08509956b5556245e1d',
    storageBucket: 'booking-classroom-4f209.appspot.com',
    apiKey: 'AIzaSyBzcZXURuDguCRMaULGywvMOohFMsr5K8U',
    authDomain: 'booking-classroom-4f209.firebaseapp.com',
    messagingSenderId: '686562888488',
  },
};
```

Una vez hecho todo esto ya estaremos en condiciones de crear el servicio e interactuar con la base de datos. Este servicio constará de lo siguiente:

```typescript
@Injectable({
  providedIn: 'root',
})
export class ClasesService implements OnInit {
  //aulas: Aula[] = [];
  constructor(private firestore: Firestore, private authService: AuthService) {}
  ngOnInit() {
    //this.getAulas().subscribe((aulas) => (this.aulas = aulas));
  }

  addAula(aula: Aula) {
    const aulaRef = collection(this.firestore, 'aulas');
    return addDoc(aulaRef, aula);
  }

  deleteAula(aula: Aula) {
    const aulaRef = doc(this.firestore, 'aulas', aula.id);
    return deleteDoc(aulaRef);
  }

  getAulas(): Observable<Aula[]> {
    const aulaRef = collection(this.firestore, 'aulas');
    return collectionData(aulaRef, { idField: 'id' }) as Observable<Aula[]>;
  }

  addReserva(reserva: Reservas) {
    const reservasRef = collection(this.firestore, 'reservas');
    return addDoc(reservasRef, reserva);
  }

  deleteReserva(reserva: Reservas) {
    const reservaRef = doc(this.firestore, 'reservas', reserva.id);
    return deleteDoc(reservaRef);
  }

  getReservas(): Observable<Reservas[]> {
    const reservaRef = collection(this.firestore, 'reservas');
    return collectionData(reservaRef, { idField: 'id' }) as Observable<Reservas[]>;
  }
}
```

Así pues, este servicio se utiliza para interactuar con la base de datos Firestore de Firebase y realizar operaciones relacionadas con las aulas y las reservas.

El servicio utiliza la inyección de dependencias para obtener instancias de Firestore y AuthService a través de su constructor.

El servicio proporciona varios métodos para realizar operaciones CRUD en la base de datos:

- addAula(aula: Aula): Este método agrega un documento de aula a la colección "aulas".
- deleteAula(aula: Aula): Este método elimina un documento de tipo aula de la colección "aulas".
- getAulas(): Devuelve un Observable que emite un array de objetos Aula representando todas las aulas almacenadas en la colección "aulas".
- addReserva(reserva: Reservas): Sirve para agregar un documento de reserva a la colección "reservas".
- deleteReserva(reserva: Reservas): Este método elimina un documento de tipo reserva de la colección "reservas".
- getReservas(): Este método devuelve un Observable que emite un array de objetos Reservas representando todas las reservas almacenadas en la colección "reservas".

En resumen, estos métodos permiten a otros componentes de la aplicación realizar operaciones CRUD en la base de datos Firestore de manera sencilla y eficiente.

4.6.3 AuthService

Antes de crear este servicio, al igual que en el anterior, se deberá de realizar algunas acciones previas. Estas acciones son muy similares a las de antes por lo que no las detallaremos en profundidad. Lo más importante es instalar la librería necesaria para tener las herramientas necesarias relativas a la autenticación.

Este servicio proporciona métodos para registrar nuevos usuarios, iniciar sesión, cerrar sesión, recuperar contraseña y actualizar la información del perfil. También se encarga de obtener información sobre el usuario actual y su estado de conexión.

De todos ellos se van a analizar dos de ellos por ser un poco diferentes a los vistos hasta ahora:

```
registro(usuario: Usuario) {
  return createUserWithEmailAndPassword(
    this.auth,
    usuario.email,
    usuario.password
  ).then((userCredential) => {
    const user = userCredential.user;

    return updateProfile(user, {
      displayName: usuario.nombre + ' ' + usuario.apellidos,
    });
  });
}
```

El método “registro(usuario: Usuario)” se encarga de registrar un nuevo usuario en Firebase. Toma como parámetro un objeto Usuario que contiene la información necesaria para el registro, como el email y la contraseña.

En el cuerpo del método, se utiliza la función “createUserWithEmailAndPassword” de Firebase, que crea un nuevo usuario en la base de datos de autenticación de Firebase con el email y la contraseña proporcionados. Esta función devuelve una promesa que se resuelve en un objeto “userCredential” que contiene información sobre el usuario recién creado.

Luego, se accede a la propiedad “user” de “userCredential” para obtener el objeto de usuario correspondiente. A continuación, se utiliza la función “updateProfile” de Firebase para actualizar el perfil del usuario recién creado. Se proporciona un objeto con la propiedad “displayName”, que se forma concatenando el nombre y apellidos del usuario. La elección de utilizar “displayName” se debe a que al establecer el nombre completo del usuario como el displayName, se puede acceder fácilmente a esta información para mostrarla en la interfaz de usuario, por ejemplo, en la bienvenida o en la barra de navegación.

El otro método del que se va a hablar es “getAdministradores()”, el cual, se utiliza para obtener una lista de administradores desde la base de datos.

```
getAdministradores(): Observable<Administradores[]> {
  const adminRef = collection(this.firestore, 'administradores');
  return collectionData(adminRef, { idField: 'id' }) as Observable<
    Administradores[]
  >;
}
```

Este método se va a utilizar para obtener la lista de emails existentes en la colección “administradores”. Será útil para ver si la persona autenticada se encuentra en esa lista o no y de esta manera otorgarle un tipo de acceso u otro.

Analicemos el código. En primer lugar, se crea una referencia a la colección de administradores utilizando el objeto “firestore” y el nombre de la colección 'administradores'.

A continuación, se utiliza la función “collectionData()” para obtener los datos de la colección de administradores. Esta función toma como argumento la referencia a la colección “adminRef” y un objeto de opciones opcional. En este caso, se pasa “{ idField: 'id' }” como opción para indicar que el campo identificador de cada documento de la colección se llamará 'id'. Esto permite que los documentos devueltos tengan un campo 'id' que corresponde a su identificador en la base de datos.

Por último, se utiliza “as Observable<Administradores[]>” para convertir los datos obtenidos en un objeto Observable que emite un array de objetos de tipo “Administradores”. Esto permite que otros componentes o servicios suscriban y reciban los datos de administradores de forma reactiva.

4.7 DESARROLLO DE LOS COMPONENTES

Después de creados todos los componentes se analizará cómo han sido desarrollados. Así pues, iremos uno por uno estudiando sus funcionalidades y características.

Hay que tener presente que en los componentes se repiten muchos elementos, como, por ejemplo, los formularios, por lo que no se analizarán todos los elementos del código, sino aquellas características más relevantes o las principales funcionalidades.

4.7.1 COMPONENTE DE LOGIN

Éste ha sido el primer componente funcional creado dentro de la aplicación. La manera de proceder ha sido siempre la misma en todos los componentes. Primero se ha creado la vista y después se le ha ido añadiendo funcionalidad.

Para crear la vista, se ha ido modificando la plantilla, es decir, “login.component.html”.

En este caso, se ha divido la pantalla en dos contendores que dividen la pantalla en dos partes.

En la parte izquierda se ha posicionado el logo de la aplicación con el siguiente código:

```
<!-- Columna izquierda -->
<div class="one">
  <div class="one-content"><img src="./assets/img/logo.svg" /></div>
</div>
```

En la parte derecha se ha situado la parte funcional. Ésta tiene una peculiaridad, y es que mediante el uso de directivas de Angular se mostrará la pantalla principal de login o el formulario para recordar la contraseña en función de si hemos pulsado en “Recuperar contraseña” o no.

Una directiva en Angular es una instrucción que se aplica a un elemento HTML y le proporciona un comportamiento específico. Las directivas permiten extender el lenguaje HTML y agregar funcionalidades adicionales a los elementos.

La directiva ngIf es una de las directivas más utilizadas en Angular. Permite mostrar u ocultar un elemento del DOM basado en una condición booleana. Si la condición es verdadera, el elemento se muestra en el DOM; de lo contrario, se oculta.

En este caso, la variable booleana se llama “recuperar”. Así pues, si es igual a true se mostrará la pantalla normal y si no la de recuperación de contraseña.

```
<div *ngIf="!recuperar" class="two-content">
  <!-- Contenido de la segunda columna -->
```

```
<div *ngIf="recuperar" class="two-content">
  <!-- Contenido de la segunda columna -->
```

Se analizará primero la parte del login y después la parte de la recuperación de la contraseña.

En la pantalla de login tenemos un formulario con dos cuadros de texto, los cuales pedirán al usuario que ingrese el email y la contraseña para poder acceder al sistema. Además, el propio cuadro de texto donde se ingresará la contraseña cuenta con un botón que permite que ésta sea visible o no. La parte del template que se corresponde con lo descrito es lo que se ve en el siguiente código:

```
<mat-form-field>
  <mat-label>Contraseña</mat-label>
  <input
    matInput
    formControlName="password"
    [type]="hide ? 'password' : 'text'"
  />
  <button
    mat-icon-button
    matSuffix
    (click)="hide = !hide"
    [attr.aria-label]="'Hide password'"
    [attr.aria-pressed]="hide"
    type="button"
  >
    <mat-icon>{{
      hide ? "visibility_off" : "visibility"
    }}</mat-icon>
  </button>
  <mat-error
    *ngIf="form.controls['password'].hasError('required')"
    >{{ getErrorMessageRequired() }}</mat-error
  >
</mat-form-field>
```

En este código se puede dividir en 5 partes diferenciadas.

- Etiqueta “<mat-form-field>”: Define un contenedor para un campo de formulario y sus elementos asociados, como etiquetas, entradas de texto, botones y errores.
- Etiqueta “<mat-label>”: Muestra una etiqueta descriptiva para el campo de contraseña.
- Etiqueta “<input>”: Especifica un campo de entrada de texto para la contraseña. Se utiliza la directiva “matInput” para aplicar estilos y comportamiento específicos de Angular Material.
    - A través de “formControlName”, se enlaza el campo de entrada a un control de formulario en el componente correspondiente.
    - “[type]” se utiliza para cambiar el tipo de entrada entre “password” y “text” según el valor de la variable “hide”.
- Botón “<button>”: Actúa como un interruptor para mostrar u ocultar la contraseña. Se muestra un ícono dependiendo del valor de la variable “hide”.
- Etiqueta “<mat-error>”: Muestra un mensaje de error si el campo de contraseña es inválido o está vacío.

Para que el formulario sea válido hay que rellenar obligatoriamente los dos campos y en el caso del email, éste debe ser válido. Esto se logra con el uso de los “Validators” que proporciona Angular. Los “Validators” los comentaremos más adelante cuando analicemos otro componente que haga un uso de validadores personalizados.

En este caso, si el formulario no es válido salta el error del “mat-error” comentado previamente y si es válido, el botón se activará gracias a otra directiva “ngIf” y al hacer click se llamara al método “onEntrar()”

```
onEntrar() {
  this.authService
    .login(this.form.value)
    .then((response) => {
      console.log(response);
      this.loading = true;
      setTimeout(() => {
        this.mostrarNavbarService.setMostrarNavBar(true);
        this.router.navigate(['inicio']);
      }, 4000);
    })
    .catch((error) => {
      console.log(error);
      const snackBarRef = this.snackBar.open(
        'El usuario o la contraseña son incorrectos.'
      );
      setTimeout(() => {
        snackBarRef.dismiss();
      }, 4000);
    });
}
```

En este método, se necesita inyectar el servicio “authService” para acceder a su funcionalidad. El servicio se inyecta en el constructor del componente mediante la inyección de dependencias de Angular, lo que permite utilizar sus métodos y propiedades.

El servicio “authService”, como ya hemos explicado, contiene la lógica para autenticar al usuario utilizando el método “login()”. Al inyectar este servicio, podemos acceder a dicho método y pasarle el valor del formulario “form” para realizar la autenticación. En este caso, el valor de “form” será un email y una contraseña que son los argumentos que necesitaba nuestro método “login()” desarrollado en el servicio.

Al llamar a este método devolverá una promesa y cuando ésta se resuelve correctamente se ejecutará el código que hay dentro del "then” y si falla se ejecutará el “catch”.

Cuando se resuelve de manera correcta, básicamente, lo que ocurre es que se realiza una redirección a la página de inicio gracias a “(this.router.navigate(['inicio'])”. El servicio router es un servicio que proporciona Angular y que permite la navegación programática y dinámica en la aplicación. Por lo tanto, únicamente hay que inyectarlo en el constructor para poder hacer uso de él. Otra cosa que ocurre es que se hace uso de otro de nuestros servicios para poner el valor booleano de la barra de navegación a “true”, de manera que cuando seamos redirigidos ésta sea visible.

El bloque de código dentro de “catch()” se ejecutará si hay un error durante el inicio de sesión. Aquí se capturará el error y se mostrará un mensaje de error en la barra de notificaciones utilizando el servicio “snackBar” que proporciona Angular Material y se establecerá un tiempo de espera de 4000 milisegundos (4 segundos) antes de que se cierre automáticamente el mensaje de error.

Una vez estudiadas las partes más interesantes del login, se analizará cuando el usuario ha olvidado la contraseña y quiere recuperarla. Es decir, cuando se cumple la condición boleana de la que hablábamos anteriormente.

Cuando “recuperar=true”, se nos redirige a una vista nueva el la que hay un formulario con un único campo y se nos indica que introduzcamos el email. Una vez introducido, Firebase se encargará de enviarnos un correo electrónico en el que nos pedirá que hagamos click en un enlace para cambiar la contraseña.

4.7.2 BARRA DE NAVEGACIÓN

La barra de navegación es un elemento que aparecerá en todas las vistas excepto en la pantalla de acceso a la aplicación. Su utilidad será la de dar soporte al resto de elementos de la aplicación a través de tres botones.

- Botón “Home”: nos redirigirá a la pantalla principal en la que podremos encontrar todas las funcionalidades. Esta vez no se realizará de manera programática, si no, desde el propio template haciendo uso de la directiva de Angular “routerLink”.

```
<button
  routerLink="/inicio/principal"
  mat-icon-button
  class="boton-pequeño"
>
  <mat-icon>home</mat-icon>
</button>
```

- Botón “Usuarios”: al pulsar sobre este botón nos redirigirá a una nueva vista en la que tendremos un nuevo formulario en el que el usuario podrá cambiar la contraseña. Se le pedirá la nueva contraseña por dos veces y si éstas coinciden se actualizará con éxito.
- Botón “Logout”: tiene la función de cerrar la sesión del usuario en activo. Hará uso del servicio “AuthService” y su método “logout”.

Otra información que muestra la barra de navegación es el nombre del usuario que está conectado en ese momento.

Para ello, se enlazará la variable “usuario” del componente con la vista haciendo uso de la interpolación “{{ usuario }}”. Así, en el constructor del componente tendremos el siguiente código para que cuando se cree el componente, automáticamente se muestre el dato del usuario.

```
this.usuario = this.authService.getUsuario()?.displayName;
```

4.7.3 COMPONENTE CREAR CLASE

En este componente se establece una de las funcionalidades principales de la aplicación, como es la creación de los espacios disponibles. Además, hay que añadir que, a este componente, únicamente tendrán acceso los usuarios que tengan rol administrador.

Para analizarlo, empezaremos viendo la parte del formulario, que, a su vez nos servirá para introducir los conceptos de formulario reactivo y de “Validator”. En esta parte será donde el usuario administrador introducirá los datos para crear una nueva clase.

Los formularios utilizados en nuestra aplicación son formularios reactivos. Éstos son una forma de manejar y validar los datos de los formularios de manera programática. En lugar de depender únicamente del estado del DOM, los formularios reactivos utilizan objetos y funciones para representar el estado y las acciones relacionadas con el formulario. Además, ofrecen varios beneficios, como una mayor flexibilidad, facilidad de mantenimiento y la capacidad de realizar validaciones personalizadas.

Ilustración 11. Formulario para crear una nueva clase

En la imagen se puede observar el resultado final de nuestro formulario. Para conseguir este resultado, lo primero se ha realizado, como siempre, es crear la parte visual y después la parte lógica. Sin embargo, se analizarán las dos partes simultáneamente a medida que se avanza por el código para evitar tener que ir de adelante a atrás continuamente.

Para crear un formulario en Angular, se hace igual que en html, se utiliza la etiqueta “form”.

```
<!-- Contenido del formulario -->
  <form [formGroup]="form" class="login-wrapper">
```

La parte más interesante de esta etiqueta es la directiva de Angular [formGroup]=”form” Ésta es una directiva de enlace de datos utilizada para asociar un objeto FormGroup (en este caso, llamada “form”) a un elemento del formulario en el template. Es decir, en nuestro código TypeScript existirá un objeto de tipo FormGroup llamado “form”.

FormBuilder es una clase proporcionada por Angular que simplifica la creación y configuración de formularios reactivos en Angular. Se utiliza para construir instancias de FormGroup y ofrece métodos y utilidades para agregar controles individuales al formulario, asignar validadores y realizar otras configuraciones relacionadas con la validación y el manejo de datos del formulario.

```
export class CrearClaseComponent {
  form: FormGroup;

  constructor(private fb: FormBuilder, private clasesServices: ClasesService) {
    this.form = this.fb.group({
      // Definir los campos del formulario y sus validaciones
    });
  }
}
```

Los siguientes elementos a analizar de los formularios son los campos o controles. Éstos serán los encargados de recibir la información por parte del usuario.

En la parte del template se crean de la siguiente manera:

```
<mat-form-field>
  <mat-label>Aula | Ejemplo: A01</mat-label>
  <input matInput formControlName="aula" />
  <mat-hint align="start">Letra y 2 dígitos</mat-hint>
  <mat-error *ngIf="form.controls['aula'].hasError('required')">
    {{ getErrorMessage() }}
  </mat-error>
  <mat-error *ngIf="form.controls['aula'].hasError('pattern')">
    {{ "Letra en mayúscula + 2 digitos" }}
  </mat-error>
</mat-form-field>
```

La parte que nos interesa aquí es la relacionada con la construcción del formulario. Así pues, el código proporcionado que enlaza el campo creado con la lógica del TypeScript es: formControlName= “aula”. Esto es otra directiva de Angular, se utilizará tantas veces como campos tenga el formulario. En el caso que nos ocupa, nuestro formulario quedaría de la siguiente manera:

```
this.form = this.fb.group({
      aula: [null, [Validators.required, Validators.pattern('[A-Z][0-9]{2}')]],
      aforo: [null, Validators.required],
      hora_inicial: [null, Validators.required],
      hora_final: [null, [Validators.required, this.validateHoraFinal]],
      dia: [null, Validators.required],
      ordenadores: [null, Validators.required],
      proyector: [null, Validators.required],
    });
```

Aquí se ve, como el primer elemento del objeto es “aula”. El nombre se corresponde con el que aparece en el formControlName de la plantilla. Por lo tanto, tenemos que cada elemento del objeto se referencia con un campo de la plantilla a través de su nombre. También vemos que todos los campos se crean con valor “null” por defecto.

Este valor cambiará cuando el usuario haga click en el botón “Crear” el cual está asociado a un método de la lógica del componente. Entre otras cosas, este método tendrá acceso al valor de cada campo en ese momento utilizando el siguiente código: “this.form.value”.

```
<button type="button" (click)="onCrear()">Crear</button>
```

Como en el caso del botón de acceso en el “login” y como en muchos otros, el botón sólo se activará si el formulario es válido. En este caso, para que sea válido tiene que cumplir con una serie de diferentes “Validators”.

En Angular, los validadores son funciones que se utilizan para validar la entrada de datos en formularios. Los validadores permiten verificar si los valores ingresados por el usuario cumplen con ciertos criterios predefinidos, como la obligatoriedad, el formato o la longitud.

Angular proporciona un conjunto de validadores predefinidos que se pueden usar directamente, como required, minLength, maxLength, email, entre otros.

Además de los validadores predefinidos, también es posible crear validadores personalizados que se ajusten a requisitos específicos de la aplicación. Los validadores personalizados son funciones que toman el control del formulario como entrada y devuelven un objeto que indica si la validación es exitosa o no.

En nuestro formulario tenemos validadores de los dos tipos, personalizados y predefinidos.

Los predefinidos son “required”, lo que significa que el usuario debe introducir un valor obligatoriamente y pattern, que, en este caso, verifica que el nombre del aula esté compuesto por la primera letra en mayúscula seguida de dos números.

En cuanto, al personalizado tenemos al validador llamado “validateHoraFinal”, el cual comprueba que la hora inicial de la clase es menor que la de fin.

```
validateHoraFinal(control: FormControl) {
  if (control.parent) {
    const horaInicialControl = +control.parent.get('hora_inicial')?.value;
    if (horaInicialControl) {
      const horaInicial = horaInicialControl;
      const horaFinal = +control.value;

      if (horaFinal <= horaInicial) {
        return { horaFinalMenor: true };
      }
    }
  }
  return null;
}
```

Una vez el formulario es correcto ya podemos hacer click en el botón para crear la clase. Este botón está asociado a un método llamado “onCrear()”. Lo destacable de este método es que

tiene un método que comprueba si la clase que se va a crear se solapa con otra que ya esta creada en el sistema, cosa que no tendría sentido. El método es el siguiente:

```
compruebaSolapamiento(aula: Aula): boolean {
  const aulasSolapadas = this.aulas.filter((_aula) => {
    if (_aula.aula === aula.aula && _aula.dia === aula.dia) {
      if (
        (_aula.hora_inicial >= aula.hora_inicial &&
          _aula.hora_inicial < aula.hora_final) ||
        (_aula.hora_final > aula.hora_inicial &&
          _aula.hora_final <= aula.hora_final) ||
        (_aula.hora_inicial <= aula.hora_inicial &&
          _aula.hora_final >= aula.hora_final)
      ) {
        return true;
      }
    }
    return false;
  });

  return aulasSolapadas.length > 0;
}
```

El método recibe el aula del formulario como parámetro. Después filtra el array de aulas creadas comparando cada aula del array con la que se ha pasado por parámetro para ver si se produce solapamiento. En caso de encontrar alguno lo almacena en la constante. Una vez comprueba todas las aulas devuelve la longitud del array declarado como constante y si es cero, es que no hay solapamiento.

A parte del formulario para la creación de clases, en este componente también podemos encontrar una tabla donde se muestran todas las aulas disponibles creadas en el sistema.

```
<table mat-table [dataSource]="dataSource" class="mat-elevation-z8">
  <!-- Columna Aula -->
  <ng-container matColumnDef="aula">
    <th mat-header-cell *matHeaderCellDef>Aula</th>
    <td mat-cell *matCellDef="let aula">{{ aula.aula }}</td>
  </ng-container>

  <!-- Columna Día -->
  <ng-container matColumnDef="dia">
    <th mat-header-cell *matHeaderCellDef>Dia</th>
    <td mat-cell *matCellDef="let aula">
      {{ obtenerNombreDia(aula.dia) }}
    </td>
  </ng-container>
```

La tabla es otro de los elementos de Angular Material. Podemos ver el uso de la sintaxis “[dataSource]”. Esto es un ejemplo de property binding en Angular. El property binding se utiliza para establecer el valor de una propiedad de un elemento HTML o de un componente Angular en función de una expresión en el componente.

En el caso de “[dataSource]”, se está enlazando la propiedad “dataSource” del componente con el atributo dataSource de un elemento HTML, como una tabla. Esto permite establecer

dinámicamente el origen de datos de la tabla utilizando la propiedad “dataSource” del componente.

La propiedad del componente “dataSource” almacenará todas las aulas que hay creadas en ese momento en la aplicación. Para ello hace uso del siguiente método:

```
cargarClases() {
  this.clasesServices.getAulas().subscribe((aulas) => {
    this.aulas = aulas;
    this.aulas.sort((a, b) => {
      const nombreA = a.aula.toUpperCase();
      const nombreB = b.aula.toUpperCase();
      const numeroA = parseInt(a.aula.substring(1));
      const numeroB = parseInt(b.aula.substring(1));

      if (nombreA < nombreB) {
        return -1;
      } else if (nombreA > nombreB) {
        return 1;
      } else {
        return numeroA - numeroB;
      }
    });
    this.dataSource = new MatTableDataSource<Aula>(this.aulas);
    if (this.dataSource && this.paginator) {
      this.dataSource.paginator = this.paginator;
    }
  });
}
```

En el método se llama la método “getAulas()” del servicio “clasesServices” y realiza las siguientes acciones:

- Llama al método “getAulas()” del servicio “clasesServices”, el cual devuelve un Observable que emite un array de objetos aulas.
- Se suscribe al Observable utilizando el método “subscribe()” para recibir los datos emitidos.
- En la función de callback del “subscribe()”, se asignan los datos recibidos a la propiedad “this.aulas” del componente.
- Se realiza una clasificación de las aulas en función de su nombre y número.
- Se crea una instancia de “MatTableDataSource” con las aulas clasificadas y se asigna a la propiedad “this.dataSource”.
- Si existe un paginador “(this.paginator)” y la fuente de datos “(this.dataSource)” está definida, se asigna el paginador a la fuente de datos para habilitar la funcionalidad de paginación en la tabla.

4.7.4 COMPONENTE RESERVAR CLASES

Este componente es muy similar al anterior. A la izquierda tendremos un formulario prácticamente idéntico al del componente “Crear Clases” que servirá para buscar una clase libre que se ajuste a las necesidades del usuario, tanto en tiempo, como en el equipamiento

que tiene la clase. A la derecha se mostrará la información en función de los datos seleccionados en el formulario. Esta vez no será en forma de tabla si no a través de otro elemento de Angular Material llamado: “mat-card”. Son como tarjetas donde se mostrará el aula con el horario y un botón para tener la posibilidad de reservar la clase.

Cuando se rellena el formulario de manera válida y se le pulsa al botón para buscar un aula libre, ponemos en marcha la principal funcionalidad de la aplicación. También ha sido el mayor desafío de todo el desarrollo del proyecto.

El objetivo es que, cuando un usuario busca una clase, el sistema verifique si en la fecha seleccionada por el usuario, así como en las horas y demás condiciones establecidas, existe alguna clase disponible.

El problema surge porque las clases se crean para días genéricos, por ejemplo, los lunes de 9:00 a 10:00, pero a la hora de reservar se selecciona la clase para una fecha concreta, por ejemplo, el 15 de junio. Así pues, la aplicación debería de ser capaz de ver si en la fecha seleccionada por el usuario el aula está disponible o no. Si no está no la debería de mostrar, pero si el resto de los lunes del calendario siempre y cuando no esté ocupada.

Para resolver este problema se creó la función “verAulas()”. En ella lo primero que se hace es filtrar las aulas a través del formulario y almacenarlas en una constante llamada “aulasFiltradas”. Si no se encuentra ninguna lanzaremos un mensaje diciendo que no hay aulas disponibles para esa fecha.

Después se llama al método “getAulasDisponibles()” pasándole la fecha que el usuario ha elegido en el formulario como parámetro. El método recorrerá el array aulasFiltradas, en el cual ya se han cribado las horas en las que el usuario desea realizar la reserva y que habíamos obtenido previamente, comparando cada aula con las aulas almacenadas en la colección “reservas” (previamente obtenida con ClasesService) para ver si coincide la misma aula en la misma fecha.

```
getAulasDisponibles(fecha: any) {
  return this.aulasFiltradas.filter((aula) => {
    for (const reserva of this.reservas) {
      if (reserva.aula.id === aula.id && reserva.fecha === fecha) {
        return false; // Aula reservada en la fecha especificada
      }
    }
    return true; // Aula no reservada en la fecha especificada
  });
}
```

```
verAulas() {
  this.mostrarElemento = false;

  // Obtener los valores del formulario
  const filtro = this.form.value;
  this.formValue = this.form.value;

  // Realizar el filtrado de las aulas
  const aulasFiltradas = this.aulas.filter((aula) => {
    //retorna true si el aula cumple con las condiciones, false en caso contrario

    if (filtro.fecha && aula.dia !== filtro.fecha.getDay()) {
      return false;
    }

    if (filtro.hora_inicial && aula.hora_inicial < filtro.hora_inicial) {
      return false;
    }

    if (filtro.hora_final && aula.hora_final > filtro.hora_final) {
      return false;
    }

    if (filtro.aforo && aula.aforo < filtro.aforo) {
      return false;
    }

    if (
      filtro.ordenadores &&
      filtro.ordenadores === 'Si' &&
      aula.ordenadores !== filtro.ordenadores
    ) {
      return false;
    }

    if (
      filtro.proyector &&
      filtro.proyector === 'Si' &&
      aula.proyector !== filtro.proyector
    ) {
      return false;
    }

    // Si el aula pasa todas las comprobaciones, se incluye en las aulas filtradas
    return true;
  });
  if (aulasFiltradas.length === 0) {
    this.mostrarElemento = true;
    this.snackBar.open(
      'No hay ningún aula disponible para la fecha indicada',
      '',
      {
        duration: 3000,
      }
    );
  }

  // Hacer algo con las aulas filtradas (por ejemplo, asignarlas a una propiedad del componente)
  this.aulasFiltradas = aulasFiltradas;
  const aulasDisponiblesFiltradas = this.getAulasDisponibles(
    this.form.controls['fecha'].value.getTime()
  );
  this.aulasDisponiblesFiltradas = aulasDisponiblesFiltradas;
  this.form.reset();
```

4.7.5 COMPONENTE MIS RESERVAS

En este componente el usuario tendrá la oportunidad de ver las reservas que ha realizado, así como, borrar aquellas en las que ya no esté interesado.

La vista mostrará una tabla en la que en cada fila aparecerá cada aula reservada. Cada reserva también tendrá un botón que servirá para cancelar la misma.

El código más interesante de este componente es el siguiente:

```
verAulas() {
  this.mostrarElemento = true;

  // Realizar el filtrado de las aulas retorna true si el aula cumple con las condiciones, false en caso contrario
  const reservasFiltradas = this.reservas.filter((reserva) => {
    const emailUsuario = this.authService.getEmailUsuario();
    if (reserva.email !== emailUsuario) {
      return false;
    }
    return true;
  });
  if (reservasFiltradas.length === 0) {
    this.mostrarElemento = false;
  }

  this.reservasFiltradas = reservasFiltradas;
}
```

Este método se encarga de filtrar la colección “reservas” almacenada en un array gracias al servicio “ClasesServices”. Para ello, se obtiene el email del usuario autenticado en ese momento y se busca aquellas reservas que tengan ese email almacenado.

El nuevo array llamado “reservasFiltradas” será el “dataSource” de la tabla del template, mostrando únicamente las reservas del usuario.

4.8 DESPLIEGUE

El despliegue de una aplicación se refiere al proceso de ponerla en funcionamiento y hacerla accesible para su uso en un entorno de producción. Durante el despliegue, se implementan y configuran todos los componentes necesarios para que la aplicación esté disponible y sea accesible para los usuarios finales. La aplicación hará uso de los servicios que proporciona Firebase para este propósito.

Firebase proporciona un servicio de alojamiento y despliegue de aplicaciones web llamado Firebase Hosting. Firebase Hosting es un servicio de hosting en la nube que permite alojar y servir aplicaciones web estáticas de forma rápida y sencilla.

En primer lugar, se debe configurar Firebase para el proyecto de Angular, lo cual implica crear un proyecto en la consola de Firebase o, en nuestro caso, elegir el que ya ha sido creado y establecer las credenciales de acceso en el proyecto de Angular.

A continuación, se prepara la aplicación Angular generando los archivos estáticos optimizados mediante el comando “ng build”. Estos archivos se guardan en una carpeta llamada “dist”.

Luego, se instala el “Firebase CLI”, que es una herramienta de línea de comandos para interactuar con Firebase. Esto se realiza ejecutando el comando “npm install -g firebase-tools”.

Después se inicia sesión en la cuenta de Firebase desde la terminal utilizando el comando “firebase login”.

A continuación, se inicializa el proyecto de Firebase dentro del directorio del proyecto de Angular mediante el comando “firebase init”. Durante este proceso, se configura Firebase Hosting como la función a utilizar y se establecen las opciones adecuadas, como el proyecto de Firebase y la carpeta “dist” que contiene los archivos estáticos.

Una vez configurado, se puede desplegar la aplicación ejecutando el comando “firebase deploy”. Esto subirá los archivos estáticos al servidor de Firebase Hosting y generará una URL para acceder a la aplicación.

Finalmente, los usuarios podrán acceder a la aplicación desplegada utilizando la URL proporcionada por Firebase.

En nuestro caso, la URL generada es la siguiente: “booking-classroom-4f209.web.app”.

5 EVALUACIÓN Y CONCLUSIONES FINALES

Durante el desarrollo del proyecto “Booking Classroom”, hemos logrado implementar un sistema de reserva de aulas que brinda a los usuarios la posibilidad de encontrar y reservar aulas disponibles de manera eficiente. A lo largo de este proceso, nos hemos enfrentado a diversos desafíos que nos han permitido adquirir conocimientos y habilidades en el desarrollo de aplicaciones con tecnologías como Angular y Firebase.

Una de las principales fortalezas de la aplicación radica en su capacidad para filtrar y mostrar únicamente las aulas disponibles que cumplen con los criterios especificados por los usuarios. Esto simplifica la búsqueda de aulas adecuadas, ahorrando tiempo y evitando la frustración de revisar opciones no disponibles.

Durante el desarrollo, nos hemos enfrentado a desafíos técnicos que nos han impulsado a ampliar nuestro conocimiento en el desarrollo con Angular y aprovechar al máximo sus capacidades para crear interfaces de usuario dinámicas y funcionales. Además, hemos utilizado Firebase como plataforma de desarrollo y alojamiento, aprovechando sus servicios de backend y base de datos en tiempo real para gestionar las reservas de forma eficiente y mantener los datos actualizados.

Mirando hacia el futuro, existen varias mejoras que podrían implementarse en la aplicación. En primer lugar, sería beneficioso incorporar un sistema de notificación por correo electrónico para informar a los usuarios sobre el éxito de sus reservas. Esto brindaría una confirmación adicional y garantizaría a los usuarios que sus reservas se han realizado con éxito.

Otra mejora importante sería la implementación de una lista de espera. Los usuarios podrían indicar su interés en una clase ocupada y recibir una notificación por correo electrónico si la clase se vuelve disponible debido a una cancelación o cambio en el horario. Esto les brindaría la oportunidad de reservar la clase deseada y maximizar el aprovechamiento de las aulas.

Además, se podría introducir una funcionalidad de selección visual en un calendario. En lugar de mostrar solo una lista de resultados de búsqueda, se podría implementar un calendario interactivo donde los usuarios puedan ver de manera intuitiva la disponibilidad de las aulas y seleccionar los horarios deseados directamente en el calendario. Esto proporcionaría una experiencia más visual y facilitaría la planificación de las clases.

También, se podría introducir la capacidad de hacer reservas recurrentes. Esto permitiría a los usuarios programar reservas periódicas para una serie de fechas y horarios determinados. Esta funcionalidad sería especialmente útil para aquellos usuarios que necesiten reservar un aula de forma regular, como profesores o grupos de estudio.

6 REFERENCIAS

1. Guía de instalación de Angular: https://docs.angular.lat/guide/setup-local
2. Formularios reactivos en Angular: https://angular.io/guide/reactive-forms
3. Tutorial sobre la carga perezosa (Lazy Loading) en Angular - Punto 85: https://www.tutorialesprogramacionya.com/angularya/detalleconcepto.php?punto=85&codigo=85&inicio=80
4. Validación personalizada de confirmación de contraseña en Angular: https://dev.to/jdgamble555/angular-confirm-password-validation-custom-validator-3pkl
5. Tutorial sobre tablas y paginación con Angular Material (mat-table y mat-paginator) - Punto 46: https://www.tutorialesprogramacionya.com/angularya/detalleconcepto.php?punto=46&codigo=46&inicio=40
6. Documentación de módulos de Angular: https://docs.angular.lat/guide/frequent-ngmodules
7. Tutorial paso a paso sobre cómo crear un login con Firebase en Angular: https://www.youtube.com/watch?v=8VTxuIvMTIc
8. Tutorial paso a paso sobre CRUD con Angular y Firebase (Firestore): https://www.youtube.com/watch?v=t_YSrxj0wGY
9. Tutorial sobre autenticación con Firebase en Angular: https://www.youtube.com/watch?v=I808gGh9RTU
10. Tutorial de Angular Material: https://www.youtube.com/watch?v=rWOwTVSMfPw
11. Curso "Angular - The Complete Guide (Edición 2023)" en Udemy: https://www.udemy.com/course/the-complete-guide-to-angular-2/
12. Tutorial de despliegue de Angular - Vídeo 46 del curso Angular: https://www.youtube.com/watch?v=pIX3fTnCLkU&list=PLU8oAlHdN5BnNAe8zXnuBNzKID39DUwcO&index=46
13. Documentación de Firebase: https://firebase.google.com/docs?hl=es-419
14. Biblioteca de componentes de interfaz de usuario Angular Material: https://material.angular.io/
15. OpenAI. "ChatGPT". Chatbot desarrollado por OpenAI. [En línea]. Disponible en: https://openai.com/.