Andrea Castilla

[Nombre de la compañía]

[Fecha]

Proyecto Fin de Grado

Logotipo, nombre de la empresa

Descripción generada automáticamente

ÍNDICE

[INTRODUCCIÓN 2](#_Toc169174947)

[TECNOLOGÍAS 3](#_Toc169174948)

[FRONTEND 3](#_Toc169174949)

[BACKEND 5](#_Toc169174950)

[OTRAS 7](#_Toc169174951)

[ANÁLISISA 9](#_Toc169174952)

[DIAGRAMAS DE SECUENCIA 9](#_Toc169174953)

[DIAGRAMA DE CASO DE USO 12](#_Toc169174954)

[DIAGRAMA DE CLASES 13](#_Toc169174955)

[DIAGRAMA DE ARQUITECTURA 14](#_Toc169174956)

PROYECTO FIN DE GRADO

[Enlace al repositorio](https://github.com/acascoc098/BARteca-V2.git)

# INTRODUCCIÓN

Alguna vez has viajado a una ciudad y te apetecía tomar algo, pero te encuentras siempre el mismo problema, ¿dónde? Puedes tener amigos que te ayuden y te aconsejen, pero si no conoces a nadie que haya visitado esa ciudad o no conozca un sitio para \*tapear\* es difícil.

Este proyecto está enfocado en una web dirigida a una “***biblioteca***” de bares, donde puedes añadir bares, hacer reservas, calificar, etc.

Como una especie de *Booking* pero de bares, donde podemos ver un listado con los bares con información de estos, por ejemplo, su ciudad, nombre, contacto, valoración,… Y con acciones como son: **añadir un bar, eliminar un bar, hacer una reserva en ese bar**, etc.

Podríamos decir que es un foro o un blog donde las personas encuentran una interfaz donde encontrar un sitio para tomarse algo y reservar un hueco para cualquier momento que quieran ir o volver al mismo bar.

Además, se incluye una parte de \**registro*\* y *login* para usuarios, con esto conseguimos que no cualquiera pueda hacer todas las funciones que tiene alguien logueado, por ejemplo, crear una entrada de un bar.

# TECNOLOGÍAS

Este proyecto está dividido en dos partes, en una encontramos el frontend, con los componentes que hacen que la aplicación tenga funcionalidad y, por otra parte, encontramos el backend, donde gestionamos los **endpoints** y BBDD.

Veamos que encontramos en cada parte.

## FRONTEND

→ React

Es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario interactivas y reutilizables. Es una de las bibliotecas más populares y ampliamente utilizadas para el desarrollo de aplicaciones web.

Algunas de las características de *React* son :

1. Componentes reutilizables: *React* permite construir interfaces de usuario divididas en componentes reutilizables así cada componente encapsula su propio estado y lógica, lo que nos facilita su reutilización en diferentes partes de una aplicación.

1. Virtual DOM: *React* utiliza un modelo de representación virtual del DOM (Document Object Model). En lugar de actualizar directamente el DOM cada vez que cambia el estado de un componente, *React* compara el virtual DOM con el DOM real y realiza solo los cambios necesarios, lo que mejora el rendimiento y la eficiencia de la aplicación.

1. JSX: *React* utiliza **JSX** como una extensión de **JavaScript** para definir la estructura de los componentes y su apariencia. **JSX** combina **HTML** y **JavaScript** en un solo archivo, lo que facilita la escritura y comprensión del código.

La elección de *React* para mi proyecto viene dada a la comodidad con la que trabajo con este y con la variedad de opciones que puedes implementar con las librerías de componentes, de las que hablaré más adelante.

Además, cuento con *Axios*, una tecnología que implementamos en el grado y la cual nos ayuda a manejar las peticiones **HTTP**. A continuación, hablamos de esta parte.

→ Axios

Es una biblioteca de **JavaScript** que permite realizar solicitudes **HTTP** desde una aplicación web.

Veamos sus características:

1. Sintaxis sencilla: *Axios* nos permite de una manera clara y sencilla realizar peticiones **HTTP** a nuestro backend en este caso aunque puede ser cualquier dirección **HTTP**

2. Admite promesas: *Axios* permite el manejo de promesas debido a que utiliza promesas basadas en el estándar **Promise** de **JavaScript**

3. Compatibilidad con navegadores: Permite utilizar el navegador lo cual lo hace muy útil desde el punto de vista del cliente

→ PrimeReact

Es una librería de componentes de *React* esta nos facilita el uso de componentes generales y su diseño, como son botones, popups, inputs, etc. Con esto tenemos un desarrollo de interfaz más ágil y una creación de esta más cómoda.

## BACKEND

→ Spring

Spring es un framework de código abierto que da soporte al desarrollo de aplicaciones basadas en Java mediante el uso de objetos sencillos.

Tiene una estructura modular y gran flexibilidad para implementar diferentes tipos de estructura según las necesidades de la aplicación que vayamos a realizar.

Algunas de las características de *Spring* son:

1. Funciona sobre JVM

**¿Qué es JVM?:** La *JVM* es la encargada de traducir los códigos bytes de Java a las instrucciones nativas de la máquina del host permitiendo así el funcionamiento de la aplicación.

2. Estructura modular

El hecho de que sea modular permite que tenga una estructura muy flexible a la hora de trabajar, ya que al tener distintos módulos con distintas funcionalidades esto permite que el trabajo sea más organizado y a la hora de reutilizar dicho trabajo nos será más sencilla su exportación.

3. Nos permite hacer todo tipo de aplicaciones

- Aplicaciones que acceden a base de datos vía SQL

- Esquemas de seguridad clásica

- Aplicaciones escalables en el paso del tiempo

Me he decantado por el uso de *Spring* debido a la flexibilidad , viabilidad y seguridad que este nos ofrece.

→ Docker

*Docker* es una plataforma de código abierto que permite a los desarrolladores automatizar la implementación de aplicaciones dentro de contenedores de software. Estos contenedores son entornos ligeros, portátiles y autosuficientes que incluyen todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas, las dependencias y la configuración necesaria.

Sus características son:

1. Portabilidad: Los contenedores Docker pueden ejecutarse en cualquier sistema con Docker instalado (servidor local, nube pública, nube privada o entorno híbrido), garantizando que las aplicaciones funcionen de manera consistente en cualquier entorno.

2. Aislamiento: Cada contenedor ejecuta aplicaciones de manera aislada, evitando que los procesos dentro de un contenedor interfieran con los de otros, lo que mejora la seguridad y estabilidad de las aplicaciones.

3. Eficiencia: A diferencia de las máquinas virtuales, los contenedores comparten el mismo núcleo del sistema operativo y son más ligeros, permitiendo ejecutar más contenedores en la misma infraestructura sin sobrecargar el sistema.

4. Gestión de Dependencias: Docker empaqueta todas las bibliotecas y componentes necesarios dentro del contenedor, simplificando la gestión de dependencias y resolviendo problemas de "funciona en mi máquina" debido a diferencias en los entornos.

5. Ecosistema Extenso: Docker cuenta con un amplio ecosistema, incluyendo Docker Hub, un repositorio en línea donde los desarrolladores pueden compartir y obtener imágenes de contenedores preconfiguradas.

## OTRAS

→ MYSQL

*MySQL* es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto muy popular y ampliamente utilizado en el desarrollo de aplicaciones web y empresariales.

Sus características son:

1. Código Abierto: Aunque existen versiones comerciales, la versión comunitaria de *MySQL* es de código abierto y gratuita, lo que permite a los desarrolladores y organizaciones usarla sin costo.

2. Alto Rendimiento: *MySQL* está optimizado para un alto rendimiento en consultas y transacciones, lo que lo hace adecuado para aplicaciones de alta demanda.

3. Escalabilidad: Soporta bases de datos de tamaño pequeño a muy grande, y puede manejar un alto número de conexiones simultáneas.

4. Seguridad: Proporciona características avanzadas de seguridad, como cifrado **SSL**, autenticación segura y permisos de usuario granulares.

5. Compatibilidad: Funciona en múltiples plataformas, incluyendo Linux, Windows y macOS, y es compatible con muchos lenguajes de programación, como **PHP, Java, Python y C++.**

6. Soporte y Comunidad: *MySQL* tiene una amplia comunidad de usuarios y desarrolladores que contribuyen al desarrollo del software y ofrecen soporte a través de foros y documentación en línea.

→ PlantUML

*PlantUML* es una herramienta de código abierto que permite crear diagramas a partir de texto escrito en un lenguaje de especificación simple y legible. Es especialmente popular entre desarrolladores y equipos de software porque facilita la creación rápida y sencilla de diagramas que se integran bien con el control de versiones y los flujos de trabajo de desarrollo ágil.

Características de *PlanUML*:

1. Simplicidad: Utiliza un lenguaje de marcado sencillo para definir diagramas, sin necesidad de herramientas complejas.

1. Variedad de Diagramas: Soporta diversos tipos de diagramas como secuencia, clases, casos de uso, actividades, entre otros.

1. Integración: Se integra fácilmente con IDEs, sistemas de control de versiones y herramientas de documentación.

1. Automatización: Puede ser utilizado en scripts de automatización y generar diagramas dinámicamente.

1. Exportación: Permite exportar diagramas en múltiples formatos gráficos como PNG, SVG, y PDF, lo que facilita su inclusión en documentación y presentaciones.

# ANÁLISIS

En estos diagramas mostramos cómo se comporta nuestra aplicación junto a la API.

## DIAGRAMAS DE SECUENCIA

En estos diagramas vemos cómo sería la secuencia sobre las diferentes acciones que se encuentra el usuario

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

## DIAGRAMA DE CASO DE USO

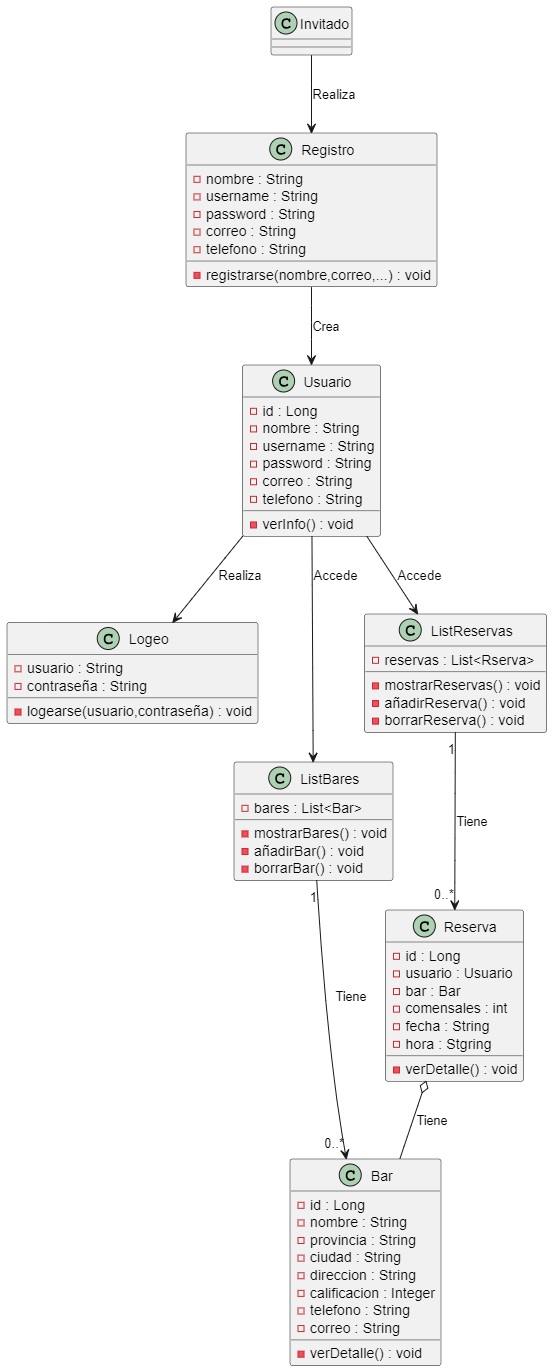
En este diagramaos encontramos los casos de uso respecto a los bares, las opciones que tiene un usuario al interactuar con la aplicación.

Diagrama

Descripción generada automáticamente

## DIAGRAMA DE CLASES

En este diagramas represento de forma general las diferentes clases y como se realcionan.



## DIAGRAMA DE ARQUITECTURA

En este diagrama muestro las diferentes tecnologías que utilizado para cada parte de la aplicación, tanto frontend como backend.

Pantalla de un video juego

Descripción generada automáticamente con confianza media

# IMPLEMENTACIÓN

Aquí muestro algunos ejemplos de código a resaltar:

## BACKEND

Aquí vemos la clase donde tenemos la configuración para el backend:

    @Bean

    SecurityFilterChain filter(HttpSecurity http) throws Exception {

        return http

                .authorizeHttpRequests((requests) -> requests

                        .requestMatchers("/barteca/\*\*", "/usuario/\*\*", "/bar/\*\*", "/reserva/\*\*", "/login/\*\*").permitAll()

                )

                .exceptionHandling((exception) -> exception

                        .accessDeniedPage("/denegado"))

                .formLogin((formLogin) -> formLogin

                        .permitAll())

                .rememberMe(Customizer.withDefaults())

                .logout((logout) -> logout

                        .invalidateHttpSession(true)

                        .logoutSuccessUrl("/")

                        .permitAll())

                .csrf((csrf) -> csrf.disable())

                .cors(Customizer.withDefaults())

                .build();

    }

    @Autowired

    public void configure(AuthenticationManagerBuilder auth) throws Exception {

        auth.jdbcAuthentication()

                .dataSource(dataSource)

                .usersByUsernameQuery("select username, password from usuario where username = ?")

                .authoritiesByUsernameQuery("select username "

                                                + "from usuario "

                                                + "where username = ?");

    }

## FRONTEND

En la parte del frontend tenemos una clase para acceder a la api, para obtener los bares o reservas, por ejemplo:

    const URL = 'http://localhost:8080/barteca';

    const getBares = async (state) => {

    const token = localStorage.getItem('token'); // Obtener el token

    if (!token) {

        console.error('No token found, redirecting to login');

        window.location.href = "/";

        return;

    }

    try {

        const req = await axios.get(URL + '/bar', {

        headers: {

            'Authorization': `Bearer ${token}`

        },

        withCredentials: true

        });

        console.log(req);

        state(req.data);

    } catch (error) {

        console.error('Error fetching data:', error);

        if (error.response && error.response.status === 401) {

        window.location.href = "/";

        }

    }

    };

En esta clase implementamos además *axios*.

Además, nos encontramos con un **AuthProvider** donde vemos como se implementa el logout:

const logout = () => {

        localStorage.removeItem('token');

        localStorage.removeItem('usuario');

        dispatch({ type: 'LOGOUT' });

    };

Para la parte del registro tenemos un componente donde llamamos a la siguiente función enviando los datos necesarios para registrar un usuario:

const registerUser = async (data) => {

  try {

      const response = await axios.post(`${URL}/usuario`, data);

      console.log('Registro exitoso:', response.data);

  } catch (error) {

      console.error('Error registrando usuario:', error);

  }

};

Para la parte de un nuevo bar vemos lo siguiente:

const nuevoBar = async (data) => {

  try {

      const response = await axios.post(`${URL}/bar`, data);

      console.log('Registro exitoso:', response.data);

  } catch (error) {

      console.error('Error registrando usuario:', error);

  }

};

Y la misma estructura de la función la podemos encontrar para las reservas.

Por último, podemos ver la implementación de Email JS en el componente de la creación de reservas:

const sendEmail = (form) => {

        emailjs

          .sendForm('service\_xxoz70p', 'template\_tcjyzo6', form, {

            publicKey: 'cOCPBrPWdhn-blN8F',

          })

          .then(

            () => {

              console.log('SUCCESS!');

            },

            (error) => {

              console.log('FAILED...', error.text);

            },

          );

      };

Donde los datos de .*sendForm* los encontramos en la página, menos *form* que llevará la estructura de los datos que nos pide el email como es la fecha, hora y el número de comensales.

Además, vemos algunos componentes utilizados de la librería de **PrimeReact**, como son los *Speed Dial* para el menú flotante o el *ScrollTop* que vemos en cada página par al darle nos mueve al principio de esta:

```js

<Route path="/about"

element={

<div style={{ position: 'relative'}}>

        <About/>

        <Tooltip target=".speeddial-top-rigth .p-speeddial-action" />

-><SpeedDial model={items2} direction="down"

style={{ right: 0, bottom: 0 }}

className="speeddial-top-rigth rigth-0 top-0"

buttonClassName="p-button-help" />

    -><ScrollTop threshold={100} behavior="smooth" />

       </div>} />

```