SkyStay

**PORTADA**

Logotipo

El contenido generado por IA puede ser incorrecto.

Eduardo Merino Fernández

**ÍNDICE (Provisional)**

**1. INTRODUCCIÓN**

1.1. Presentación del proyecto…………………………………………………………………4  
1.2. Motivación y justificación…………………………………………………………………..4  
1.3. Objetivos generales y específicos………………………………………………………..4  
1.4. Metodología de desarrollo y herramientas utilizadas…………………………………..5

**2. PLANIFICACIÓN Y ANÁLISIS**

2.1. Definición del problema……………………………………………………………………5  
2.2. Requisitos del sistema……………………………………………………………………..5  
2.2.1. Requisitos funcionales  
2.2.2. Requisitos no funcionales  
2.3. Análisis de viabilidad………………………………………………………………………..6  
2.3.1. Estudio de tecnologías  
2.3.2. Recursos necesarios  
2.3.3. Planificación temporal del desarrollo

**3. DISEÑO DE LA ARQUITECTURA DEL SISTEMA**

3.1. Modelo de arquitectura (cliente-servidor) ………………………………………………..7  
3.2. Diseño de la API REST………………………………………………………………….….7  
3.2.1. Definición de endpoints  
3.2.2. Seguridad y autenticación (JWT, roles de usuario)   
3.3. Modelado de la base de datos……………………………………………………………..8  
3.3.1. Diagrama entidad-relación (ER)   
3.3.2. Estructura y normalización de tablas  
3.4. Diseño del frontend…………………………………………………………………………..9

**4. IMPLEMENTACIÓN DEL BACKEND**

4.1. Configuración del entorno y dependencias (Spring Boot, Maven)……………………..10  
4.2. Creación del modelo de datos en MariaDB……………………………………………….10  
4.3. Implementación de controladores y servicios……………………………………………..11  
4.4. Seguridad y autenticación con Spring Security y JWT…………………………………...14  
4.5. Gestión de reservas y usuarios……………………………………………………………..  
4.6. Envío de correos electrónicos con notificaciones…………………………………………  
4.7. Generación de informes y estadísticas…………………………………………………….

**5. IMPLEMENTACIÓN DEL FRONTEND**

5.1. Configuración del entorno y dependencias (React) ………………………………………  
5.2. Desarrollo de la estructura de componentes……………………………………………….  
5.3. Conexión con la API REST…………………………………………………………………  
5.4. Gestión de sesiones y autenticación………………………………………………………  
5.5. Diseño de las vistas y experiencia de usuario……………………………………………..  
5.6. Implementación de funcionalidades principales……………………………………………88  
5.6.1. Consulta de disponibilidad y reservas……………………………………………………..88  
5.6.2. Sistema de fidelización (puntos y recompensas) ………………………………………88  
5.6.3. Gestión de reservas y cancelaciones…………………………………………………….88

**6. PRUEBAS Y DEPURACIÓN**

6.1. Pruebas unitarias (JUnit) ……………………………………………………………………..88  
6.2. Pruebas de integración (Postman, Swagger) ………………………………………………88  
6.3. Pruebas de carga y rendimiento……………………………………………………………..88  
6.4. Pruebas de usabilidad y accesibilidad………………………………………………………88  
6.5. Estrategias de depuración y resolución de errores………………………………………..88

**7. DESPLIEGUE Y MANTENIMIENTO**

7.1. Contenedorización con Docker………………………………………………………………88

7.2. Despliegue del Front…………………………………………………………………………..88  
7.3. Estrategia de mantenimiento y actualizaciones……………………………………………88

**8. RESULTADOS Y CONCLUSIONES**

8.1. Comparación entre objetivos iniciales y resultados obtenidos……………………………88  
8.2. Problemas encontrados y soluciones aplicadas……………………………………………88  
8.3. Valoración del rendimiento del sistema……………………………………………………..88  
8.4. Posibles mejoras y ampliaciones futuras……………………………………………………88

**9. BIBLIOGRAFÍA Y REFERENCIAS**

**10. ANEXOS**

10.1. Código fuente relevante…………………………………………………………………….88  
10.2. Documentación técnica de la API (Swagger) ……………………………………………88  
10.3. Manual de usuario y administrador……………………………………………………….88  
10.4. Planificación temporal detallada………………………………………………………….88

**1. INTRODUCCIÓN**

**1.1. Presentación del Proyecto**

**SkyStay** es una plataforma web desarrollada para gestionar reservas en una cadena hotelera. El sistema se divide en dos partes:

* **Frontend:** Interfaz web intuitiva para que los usuarios puedan consultar disponibilidad, gestionar reservas y acceder a funcionalidades como el sistema de fidelización.
* **Backend:** Encargado de procesar la lógica del negocio, gestionar los datos y mantener la comunicación con la base de datos mediante una API REST desarrollada en Spring Boot.

El sistema contempla tanto la gestión de reservas por parte de los clientes como funcionalidades administrativas destinadas a los gestores de los hoteles.

**1.2. Motivación y Justificación**

En un entorno cada vez más digitalizado, las empresas hoteleras demandan soluciones modernas que agilicen la gestión de sus servicios y mejoren la experiencia del cliente. SkyStay surge para cubrir esta necesidad ofreciendo:

* Una plataforma unificada y centralizada para cadenas hoteleras.
* Interfaz amigable para el cliente y funcionalidades avanzadas para administradores.
* Sistema de recompensas que mejora la fidelización.

El objetivo es crear un sistema eficaz, seguro y moderno que responda a las exigencias del sector hotelero actual.

**1.3. Objetivos Generales y Específicos**

**Objetivo General:** Desarrollar una plataforma completa de gestión de reservas hoteleras que integre frontend, backend y base de datos.

**Objetivos Específicos:**

1. Crear una API REST con Spring Boot para la gestión de reservas.
2. Implementar autenticación y autorización diferenciada para clientes y administradores.
3. Permitir reservas en tiempo real y gestión de recompensas.
4. Desarrollar un frontend responsive e intuitivo en React.
5. Incluir funcionalidades de cancelación y modificación de reservas.
6. Incorporar notificaciones por correo para eventos clave.
7. Asegurar la protección de los datos personales y transacciones.
8. Generar estadísticas de ocupación para los administradores.

**1.4. Metodología de Desarrollo y Herramientas Utilizadas**

**Metodología:**  
Se empleará una metodología ágil basada en **iteraciones semanales** con entregables funcionales, revisiones constantes y pruebas continuas.

**Herramientas de Desarrollo:**

* **Backend:** Spring Boot 3.x
* **Frontend:** React
* **Base de Datos:** MariaDB
* **Control de versiones:** Git + GitHub
* **Pruebas:** JUnit (pruebas unitarias) + Postman (pruebas de API)
* **Documentación API:** Swagger
* **Despliegue:** Docker
* **IDE:** IntelliJ IDEA y Visual Studio Code

**2. PLANIFICACIÓN Y ANÁLISIS**

**2.1. Definición del Problema**

Las cadenas hoteleras que operan en múltiples localizaciones enfrentan dificultades al gestionar reservas de forma centralizada. Los sistemas tradicionales suelen carecer de integración, personalización o escalabilidad. SkyStay pretende solventar estas deficiencias proporcionando una solución web moderna y eficiente.

**2.2. Requisitos del Sistema**

**2.2.1. Requisitos Funcionales**

* Registro y autenticación de usuarios (clientes y administradores).
* Visualización de disponibilidad de habitaciones.
* Realización, cancelación y modificación de reservas.
* Canjeo de recompensas.
* Gestión de habitaciones, servicios y reservas por parte de los administradores.
* Envío automático de correos de confirmación y recordatorios.
* Generación de informes estadísticos para administradores.

**2.2.2. Requisitos No Funcionales**

* Sistema seguro con cifrado de datos sensibles.
* Accesibilidad desde distintos dispositivos (responsive design).
* Alta disponibilidad y rendimiento.
* Trazabilidad y registro de actividades.
* Despliegue en contenedores para facilitar la escalabilidad.

**2.3. Análisis de Viabilidad**

**2.3.1. Estudio de Tecnologías**

**Software:**

* Backend: Spring Boot
* Frontend: React
* Base de datos: MariaDB
* Despliegue: Docker
* Pruebas: JUnit, Postman
* Control de versiones: Git + GitHub
* Documentación: Swagger

**Hardware:**

* CPU: 4 núcleos o superior
* RAM: mínimo 8 GB
* Disco: mínimo 50 GB SSD
* Sistema operativo servidor: Windows

**2.3.2. Recursos Necesarios**

* Ordenadores de desarrollo con capacidad adecuada para ejecución local.
* Acceso a servidores para pruebas de despliegue.
* Licencias y entornos de desarrollo (IDE, herramientas de documentación, etc.).

**2.3.3. Planificación Temporal del Desarrollo**

| **Semana** | **Tarea** |
| --- | --- |
| 1 | Análisis de requisitos y diseño de arquitectura |
| 2-4 | Desarrollo del backend (Spring Boot) |
| 5-6 | Desarrollo del frontend (Angular) |
| 7-8 | Integración del backend con el frontend |
| 9-10 | Pruebas funcionales y corrección de errores |
| 11 | Despliegue en entorno real (Docker) |
| 12 | Documentación técnica y preparación de presentación final |

**3. DISEÑO DE LA ARQUITECTURA DEL SISTEMA**

**3.1. Modelo de Arquitectura (Cliente-Servidor)**

El sistema SkyStay adopta un modelo de arquitectura **cliente-servidor**, en el cual la lógica de presentación (frontend) y la lógica de negocio (backend) se encuentran desacopladas, permitiendo una comunicación eficiente y escalable mediante peticiones HTTP.

* **Cliente (Frontend):** Aplicación Angular accesible desde navegadores. Se encarga de la interacción con el usuario y el envío de solicitudes a la API.
* **Servidor (Backend):** API REST desarrollada en Spring Boot que gestiona la lógica de negocio, operaciones sobre la base de datos y aplica medidas de seguridad.
* **Base de Datos:** Servidor MariaDB que almacena de forma estructurada toda la información relacionada con usuarios, reservas, habitaciones, etc.

El flujo de información es bidireccional: el cliente realiza peticiones y el servidor responde con datos procesados en formato JSON.

**3.2. Diseño de la API REST**

La API REST de SkyStay sigue los principios RESTful para garantizar interoperabilidad, simplicidad y eficiencia. Todas las operaciones están organizadas en endpoints lógicos y cumplen con los verbos HTTP adecuados.

**3.2.1. Definición de Endpoints (ejemplos)**

| **Recurso** | **Método** | **Endpoint** | **Descripción** |
| --- | --- | --- | --- |
| Usuarios | POST | /api/auth/register | Registro de nuevos usuarios |
| Usuarios | POST | /api/auth/login | Autenticación (login) |
| Habitaciones | GET | /api/habitaciones | Obtener lista de habitaciones disponibles |
| Reservas | POST | /api/reservas | Crear una nueva reserva |
| Reservas | GET | /api/reservas/{id} | Consultar detalles de una reserva |
| Recompensas | PUT | /api/recompensas/canjear | Canjear puntos de fidelización |
| Admin/Informes | GET | /api/admin/informes | Obtener estadísticas e informes de ocupación |

Todos los endpoints devuelven respuestas en JSON y validan los datos recibidos mediante DTOs y validadores; también habrá otros métodos de las funcionalidades que se incluyan posteriormente en el proyecto, como el saldo del cliente o los puntos.

**3.2.2. Seguridad y Autenticación (JWT, Roles de Usuario)**

Para garantizar la seguridad, SkyStay implementa un sistema de autenticación y autorización basado en **JWT (JSON Web Tokens)**:

* Al iniciar sesión correctamente, el servidor genera un **token JWT** firmado que el cliente debe incluir en cada petición (Authorization: Bearer).
* La API verifica este token en cada solicitud protegida.
* Se definen **roles de usuario**:
  + ROLE\_CLIENTE: Puede consultar habitaciones, crear y modificar sus reservas, canjear puntos.
  + ROLE\_ADMIN: Puede acceder a paneles de gestión, informes, modificar disponibilidad y gestionar usuarios.

Se protege cada endpoint con anotaciones como @PreAuthorize y filtros personalizados en Spring Security.

También las contraseñas estarán encriptadas en la base de datos con Spring Security para así evitar filtraciones de datos importantes.

El cliente para confirmar su cuenta tiene que verificar su correo electrónico con el link que se le proporcionará. Esto se realizará mediante una API de correo que tiene diferentes métodos para automatizar el envío de correos.

**3.3. Modelado de la Base de Datos**

El diseño de la base de datos se realiza respetando los principios de normalización para evitar redundancias y garantizar integridad referencial. Se emplea MariaDB como sistema gestor.

**3.3.1. Diagrama Entidad-Relación (ER)**

El sistema contiene, entre otras, las siguientes entidades principales:

* **Usuario:** ID, nombre, correo, contraseña (hasheada), rol, puntos.
* **Habitación:** ID, número, tipo, estado, precio por noche.
* **Reserva:** ID, usuario\_id, habitacion\_id, fecha\_entrada, fecha\_salida, estado.
* **Recompensa:** ID, descripción, puntos\_necesarios.
* **Check-in / Check-out:** ID, reserva\_id, timestamps asociados.

Las relaciones incluyen:

* Un usuario puede tener muchas reservas.
* Una habitación puede estar asociada a varias reservas a lo largo del tiempo.
* Una reserva puede permitir el canje de una o más recompensas.

El diagrama ER puede ser generado en herramientas como MySQL Workbench o dbdiagram.io.

HACER DIAGRAMA

**3.3.2. Estructura y Normalización de Tablas**

Las tablas están diseñadas siguiendo **hasta la 3ª Forma Normal (3NF)**:

* Cada tabla tiene una clave primaria única.
* Las claves foráneas aseguran relaciones entre entidades.
* No se permite la redundancia de datos.
* Se utilizan índices para optimizar consultas frecuentes (habitaciones disponibles, historial de reservas, etc.).

**3.4. Diseño del Frontend**

El frontend será una SPA (Single Page Application) desarrollada en Angular. La comunicación con el backend se realiza mediante HTTP y JSON.

**3.4.1. Wireframes y Prototipos**

Los diseños iniciales se elaboran como wireframes para visualizar la experiencia de usuario. Se incluyen las siguientes pantallas:

* **Página de inicio:** acceso a login y registro.
* **Página de registro:** Registro completo de cliente o admin.
* **Panel de cliente:** reservas actuales, historial, sistema de puntos.
* **Panel de administrador:** gestión de habitaciones, usuarios y reportes.
* **Formulario de reserva:** selección de fechas y servicios extra.
* **Confirmaciones y notificaciones:** mediante modales y alertas.

Herramientas utilizadas para diseño: **Figma** y **Adobe XD** (opcional).

**3.4.2. UX/UI: Principios Aplicados**

Se siguen principios de **Diseño Centrado en el Usuario (UCD)** y buenas prácticas de UI:

* **Consistencia:** estilos y componentes uniformes en toda la plataforma.
* **Feedback inmediato:** mensajes de éxito/error tras cada acción del usuario.
* **Accesibilidad:** contraste adecuado, navegación por teclado y etiquetas ARIA.
* **Responsive Design:** adaptable a móviles, tablets y escritorio.
* **Minimalismo funcional:** interfaz limpia, sin elementos innecesarios.

**4. IMPLEMENTACIÓN DEL BACKEND**

**4.1. Configuración del entorno y dependencias (Spring Boot, Maven)**

Para comenzar con el back, lo primero a realizar es tener un proyecto básico con todas las dependencias necesarias, en este caso en Maven debido a que vamos a utilizar dependencias en Spring y es mas sencillo de esta manera.  
En este proyecto se ha implementado mediante el SpringInizialicer que te permite elegir las dependencias que quieres que contenga tu proyecto (Spring Security, Spring Web, Mariadb, DevTools…), con esto ya tenemos la base del proyecto que vamos a realizar.

Para la configuración general del proyecto tenemos el archivo de application.properties que nos encontramos en resources dentro del src, aquí podremos configurar la base de datos que vamos a utilizar y diferentes utilidades que nos da Spring para probar y configurar un proyecto.

**4.2 Creación del modelo de datos en MariaDB**

Una parte muy relevante es la base de datos que vamos a necesitar, en este caso la seleccionada es MariaDB, debido a que necesitamos una base de datos relacional, porque queremos crear tablas diferentes con relaciones entre estas.

En nuestra base de datos tendremos diferentes tablas: Clientes, Administradores, Objetos, Reservas, Hoteles, Habitaciones, Recompensas… Cada una tiene su ID y sus claves foráneas que hace que se relacionen entre si. Esto es relevante porque mediante a nuestro Back que es una API Rest podremos consultar, modificar, crear y eliminar diferentes entradas en a base de datos.(CRUD)

**4.3 Implementación de controladores, recursos y entidades**

Para este proyecto el Back en Spring se ha distribuido de la siguiente manera:

-Primero tenemos las **entidades** que son básicamente los objetos de nuestro proyecto que queremos que Spring los convierta y los trate como tablas de nuestra base de datos con sus diferentes columnas. Aquí hay un ejemplo con el objeto reserva:

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Como vemos para que Spring lo tome como una tabla en la base de datos lo tenemos que declarar como @Entity, después en @Table (name) se indica el nombre de la tabla. Luego con el @Id indicamos la clave primaria de la tabla y que se autoincrementa. También indicamos las claves foráneas con el @ManyToOne y el @OneToMany, después ponemos los contructores y uno en vacío, los getters y setters y en este caso un método que calcula el precio de la reserva dependiendo de la diferencia de días y del precio del establecmiento.

-Después tenemos los **repositorios** que básicamente contienen los métodos que vamos a utilizar para hacer consultas a la base de datos y que nos devuelva información en forma de Objetos, booleanos… Gracias a Spring se pueden hacer consultas complejas mencionando los atributos del objeto y Spring detecta el propio nombre sin necesidad de hacer ningún método extra. Aquí un ejemplo con el repositorio del Cliente:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Como vemos hay que indicar el @Repositorio para indicarselo a Spring y hacer un extends del JpaRepository con el Objeto y el tipo de Primary Key.

-Por último tenemos a los **controladores** que básicamente realizan las operaciones CRUD en la base de datos, es donde se conectará el front para hacer las diferentes peticiones para lanzar o recibir información a la API, sirve de intermediario entre el repositorio con la base de datos y el Front. Aquí un ejemplo del controlador del Cliente con las peticiones principales:

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Como se puede observar hay que declarar el @RestController para indicar que es un controlador de tipo Rest, y como vemos el resto son métodos cada uno con diferentes consultas, dependiento del tipo (Get, Post, Delete, Update) se pondrá un tipo u otro de mapping, con la ruta de la petición http. Cada una de estas peticiones se conectará al repositorio necesario y realizará modificaciones en la base de datos u obtendrá información determinada para cada petición.

Con esta base podremos realizar la API Rest y conectarla, en este caso lo que se realizo para la comprobación del funcionamiento fue la utilización de Postman para realizar las peticiones a la API y probar su correcto funcionamiento.

**4.4 Seguridad y autenticación con Spring Security**

Para asegurar los datos de los clientes y evitar las filtraciones de contraseñas, en este proyecto se ha utilizado Spring Security para encriptar y comparar las contraseñas, de tal manera que se se filtra la base de datos con información de los usuarios las contraseñas aparezcan encriptadas. Para esto se han realizado los siguientes métodos:

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Como se puede observar el método encode dentro del PasswordEncoder codifica la contraseña y el método matches es un boleano que indica si la contraseña coincide con la codificada lo que nos sirve perfectamente para un inicio de sesión del usuario.

-------Añadir mas cosas de seguridad aquí-----------------------------------

**4.5 Gestión de reservas y usuarios**

Para la gestión de reservas se han realizado diferentes métodos para hacer que un usuario no pueda reservar una habitación que este ocupada en ese momento, para ello se ha realizado un método boleano de esta ocupado y para poder reservar una habitación una de las condiciones es que ese método te devuelva false. En dicho método hay que introducir la habitación, el día de entrada y el de salida, lo que se hace es recorrerse estos días comprobando que no hay ningún día de estos que coincida con una reserva, si no coincide ningún día el método retornará false y si coincide retornará true. Con esto podemos comprobar fácilmente en un rango de días si la habitación esta o no ocupada.

Texto

El contenido generado por IA puede ser incorrecto.

**4.6 Envío de correos electrónicos con notificaciones**

Un añadido importante al proyecto es el de la API de correos electrónicos. Lo he realizado como una clase que implementa diferentes métodos: Un correo de confirmación al crear tu cuenta, un correo con el código de la recompensa canjeada, una confirmación al realizar una reserva de los datos de dicha reserva…

Esto se realiza gracias a Spring Mail con la que puedes automatizar los correos. Para esto necesitas una cuenta de correo electrónico con el permiso de aplicaciones.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Como vemos hay métodos que realizan diferentes funciones en este caso tendremos 3. El primero será el mail de confirmación que llegará a la cuenta de correo utilizada para registrarse y que sirve para confirmar el correo de dicha cuenta.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Como vemos este sería un ejemplo del correo que se le enviará cuando se registra, en dicho correo hay un link para confirmar la cuenta en forma de token, cuando se hace click sobre el gracias a una petición del back se confirma la cuenta y se verifica. Gracias a este sistema se asegura que la cuenta que se ha utilizado al registrarse es la cuenta del cliente.

Aparte del anterior tenemos un correo que se envía automáticamente al cliente cada vez que realiza una reserva en uno de los hoteles, a continuación, puede ver un ejemplo donde aparte de la confirmación de la reserva incluye información del hotel reservado, el día de entrada y de salida.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Por último tendremos un correo que se enviará cuando se canjea una recompensa con los puntos del cliente acumulados, en este correo se enviará el codigo para canjear esta recompensa, se ha decidido realizar de esta manera para que sea mas seguro y que solo en el correo del propietario se pueda ver el código de la recompensa.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**4.7 Generación de informes y estadísticas**

De cara al administrador es importante que este pueda tener toda la información que pueda para poder administrar correctamente el hotel que este encargado, por lo que una de sus funciones es generar diferentes informes que este puede imprimir o guardar como pdf con información relativa a dicho hotel, diferentes clientes, reservas o capacidad y disponibilidad de diferentes habitaciones. Estos informes estarán disponibles en el apartado del administrador mediante un botón podrás generar este archivo que será enviado directamente al correo del administrador adjuntado con un PDF.

Por lo que este proceso lo realizará el back en Java mediante una Api que genere el documento pdf y la envié por la api del correo adjuntando este archivo anteriormente creado.

----ADJUNTAR CAPTURAS-----------------------

**5. IMPLEMENTACIÓN DEL FRONTEND**

**5.1. Configuración del entorno y dependencias (React)**

Para el apartado visual se ha utilizado React debido a que hoy en día la web es un medio mucho mas sencillo y multiplataforma que una aplicación nativa. Sobre todo para el cliente, se ha observado que muchas veces al reservar productos o servicios en vez de un ordenador se utiliza el teléfono móvil u otros dispositivos como las tablets, por lo que se ha decidido realizar este proyecto en web y conectarse a la base de datos y a las diferentes funcionalidades mediante la Api Rest en Spring. Como Framework se ha decidido utilizar React debido a que hoy en día es de los más utilizados y con mas soporte. Y como IDE Visual Studio Code, que es el mejor para desarrollo en JavaScript y en general todo el desarrollo web con sus diferentes Frameworks.

Para empezar en Visual Studio Code en su terminal nos movemos a la carpeta de nuestro proyecto y tenemos que tener instalado npm y Node y luego realizar los siguientes comandos para crear una plantilla de nuestro proyecto.

Texto

El contenido generado por IA puede ser incorrecto.Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Cuando tengamos todo esto realizado podremos hacer un deploy de nuestro front para probar el funcionamiento en localhost:3030 y a partir de aquí tendremos nuestra estructura principal a la que más adelante se le añadirán dependencias como la de calendario para poder seleccionar las fechas de entrada y salida para el hotel, etc…

**5.2. Desarrollo de la estructura de componentes**

La estructura principal del Front es la siguiente:

Texto

El contenido generado por IA puede ser incorrecto.

Como se puede observar el la imagen anterior tenemos la estructura del proyecto:

**node\_modules**

* Carpeta generada automáticamente por npm install.
* Contiene todas las dependencias del proyecto.

**public**

* Archivos estáticos públicos accesibles directamente (por ejemplo, imágenes o favicon).
* index.html generalmente se encuentra aquí en un proyecto Vite o React.

**src**

* Carpeta principal del código fuente.

Dentro de src hay:

**assets**

* Contiene imágenes, fuentes o archivos estáticos que se usan dentro de los componentes.

**components**

* Aquí es donde están los **componentes reutilizables o las diferentes vistas** del proyecto, donde dependiendo de la acción del usuario este llevará a una u otra vista.
* Los archivos .jsx son componentes de React (escritos en JSX).

Archivos dentro de components:

* + Canjear.jsx: Una vista para el cliente donde este puede canjear puntos por diferentes recompensas.
  + Home.jsx: Vista de la página principal o landing.
  + Login.jsx: Vista para el inicio de sesión.
  + Profile.jsx: Vista del perfil de usuario.
  + Register.jsx: Vista para el registro de nuevos usuarios o administradores.

Aquí dependiendo de lo que necesitemos tendremos mas vistas para añadir opciones y funcionalidades a la web.

**src**

* App.jsx: Componente principal de la aplicación, donde se definen las rutas y se estructura la navegación en este caso donde te redirige a las diferentes rutas que se encuentran en la carpeta components.
* index.css: Estilos globales, puede contener resets o configuraciones base.
* main.jsx: Punto de entrada de la aplicación React. Aquí se renderiza el <App /> dentro del root del HTML.

**5.3. Conexión con la API REST**

Para realizar las conexiones con la API se utiliza fetch para poder comunicarse con la API y realizar las diferentes peticiones que necesitemos (POST, GET, DELETE, UPDATE).

Como ejemplo para ver su funcionamiento he utilizado la vista del Login del Cliente:

**1. Recogida de datos del formulario**

El estado user y password se actualizan con los datos introducidos por el usuario mediante los inputs:

Texto

El contenido generado por IA puede ser incorrecto.

**2. Envía las credenciales al servidor**

Cuando se envía el formulario (onSubmit), se llama a la función handleSubmit, que hace lo siguiente:

Una captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

- Aquí se utiliza fetch para hacer una **petición POST** a la URL /api/clientes/login de tu API.  
- En el body se envía un **JSON con el usuario y la contraseña**.  
- El Content-Type: application/json indica que se están enviando datos en formato JSON.

**3. Procesa la respuesta del servidor**

const success = await response.json();

- Aquí se esta esperando la **respuesta del servidor**.  
- Esta respuesta contiene algo que indique si el login fue exitoso, por ejemplo true o false.

**4. Si la autenticación es exitosa:**

**Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.**

Si el login es correcto:

* Se hace otra petición GET para obtener la **información completa del usuario** (/api/clientes/{user}).
* Se guarda en el estado padre usando setUser(userJson) (esto se pasa como prop desde un componente superior).
* Se redirige al usuario a la ruta /profile usando navigate.

**5. Si el login falla:**

- Muestra un mensaje de error si la autenticación no fue exitosa.

Y está es la manera que básicamente funcionan las peticiones en todas las vistas tanto para cambiar datos de la base de datos como para pedir cierta información y obtener los datos.

**5.4. Gestión de sesiones y autenticación**

En el caso de las sesiones se ha utilizado un método más seguro que las propias rutas con el usuario, en este caso cuando inicias sesión entras en la ruta perfil y dependiendo de con que usuario hayas iniciado sesión la web será de una u otra manera con os datos de cada usuario. Con el sistema actual cada vez que el usuario sale de la web no se mantiene su cuenta y tiene que volver a iniciar sesión por seguridad.

-----CAPTURA DE LA WEB CON EL DOMINIO /PERFIL

Para la autenticación lo que hace el front es conectarse con la API en el inicio de sesión y realizar dicha petición y dependiendo de si es correcta o no que acceda al usuario o no le sea disponible y pruebe de nuevo, también por seguridad habrá un bloqueo de 30 min de la cuenta cuando se realicen mas de 5 intentos seguidos de inicio de sesión. Esto último será trabajo únicamente del Front que no realizará dicha petición hasta pasado ese tiempo.

---CAPTURA INICIO SESIÓN Y INTENTOS

**5.5. Diseño de las vistas y experiencia de usuario**

En las webs es importante el diseño visual tanto como la funcionalidad, por lo que es importante hacer unas vistas y una web que llame la atención, que sea legible, intuitiva y que sea acorde al tema y al sector que pertenece.

En este proyecto en cada vista justo encima del HTML esta la parte de style que es el código CSS que aportan estos detalles que ayudan al usuario a aportar este diseño que se busca.

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

Este es un ejemplo de la utilización del CSS en el archivo, en el return va tanto el HTML como el CSS junto.

Aparte del CSS básico en este proyecto se utilizan diferentes librerías para aportar diseños mas personalizados y prediseños, el caso mas claro es el calendario de reservas, donde el cliente puede seleccionar un rango de fechas que es visible en el calendario para que sea mas sencillo el proceso de reserva de una habitación.

---CAPTURA DE LIBRERÍA DE CALENDARÍO Y MUESTRA EN IMAGEN DE LA WEB-----

Aparte de esto se ha intentado con el uso de colores sin mucho contraste y con el tipo de navegación intuitiva hacer esta web accesible para toda clase de público, y que todos los procesos que puedes realizar queden claros y que sean sencillos de realizar.

**5.6. Implementación de funcionalidades principales**

Algunas de las funcionalidades principales son las siguientes:

* Consulta de disponibilidad y reservas

Tanto el cliente como el administrador puede consultar puede comprobar la disponibilidad de una habitación en un periodo de tiempo determinado gracias a los métodos que se conectan con la API.

También en el caso de las reservas, el propio cliente puede ver las reservas que ha realizado y en el caso del administrador puede ver todas las reservas realizadas en el hotel al que está a cargo.

---------CAPTURAS DE PANTALLA DE RESERVAS----------------

* Sistema de fidelización (puntos y recompensas)

Un añadido importante para fortalecer la fidelización del cliente es el sistema de puntos y recompensas. Esto consiste en que el cliente cada vez que haga una reserva en la web se le acumularán puntos dependiendo del valor de la reserva realizada, estos puntos estarán vinculados a la cuenta y cuando acumulas una cantidad determinada puedes usarlos para canjear diferentes recompensas disponibles. Estas recompensas serán obtenidas por el cliente a través de un código que será enviado al correo del cliente en el momento que este canjee los puntos.

* Gestión de reservas y cancelaciones

**6. PRUEBAS Y DEPURACIÓN**

**6.1. Pruebas unitarias (JUnit)**

Las pruebas son una fase tan importarte como el propio desarrollo ya que gracias a estas podemos comprobar el correcto funcionamiento del software antes de que lo haga el cliente final, y con esto poder detectar errores de diferentes tipos y corregirlos.

En este caso una de las pruebas mas relevantes son las pruebas unitarias,

Las pruebas unitarias se han centrado en los componentes del backend desarrollados en Spring Boot. Se han utilizado test con **JUnit 5**, orientados a verificar la lógica interna de los servicios, controladores y repositorios. Cada método crítico ha sido cubierto por casos de prueba que contemplan tanto comportamientos esperados como condiciones límite y errores esperados. El objetivo principal ha sido garantizar la fiabilidad del núcleo de la lógica de negocio.

**6.2. Pruebas de integración (Postman, Swagger)**

Para validar el correcto funcionamiento de la **API REST**, se realizaron pruebas de integración mediante **Postman**, simulando solicitudes típicas del flujo de uso: creación, modificación, cancelación y consulta de reservas, así como autenticación y gestión de recompensas. Las colecciones de Postman permitieron automatizar pruebas para diferentes escenarios.

Además, **Swagger** se integró en el proyecto para documentar y explorar de forma interactiva los endpoints de la API, lo cual facilitó tanto el desarrollo como la detección temprana de inconsistencias entre los contratos de la API y su implementación.

**6.3. Pruebas de carga y rendimiento**

Se efectuaron pruebas básicas de carga utilizando herramientas como **Apache JMeter**, con el fin de evaluar el comportamiento del sistema bajo múltiples peticiones simultáneas, simulando picos de usuarios accediendo al sistema para consultar disponibilidad o realizar reservas.

Estas pruebas se orientaron especialmente al backend, para asegurar que la API REST pudiera responder de forma eficiente sin provocar caídas ni tiempos de espera excesivos. Se identificaron cuellos de botella y se realizaron optimizaciones en las consultas a la base de datos y la configuración del entorno de ejecución.

**6.4. Pruebas de usabilidad y accesibilidad**

Desde el frontend, desarrollado en **React**, se realizaron pruebas de usabilidad con usuarios reales que simularon distintos perfiles (cliente y administrador). Se recopilaron observaciones sobre la claridad de la interfaz, la navegación, y la disposición de elementos clave como botones y formularios.

Asimismo, se validaron aspectos de **accesibilidad**, como el uso de etiquetas adecuadas, contraste de colores y navegación mediante teclado, siguiendo las pautas WCAG (Web Content Accessibility Guidelines). Estas pruebas contribuyeron a mejorar la experiencia de usuario final y garantizar la inclusión.

**6.5. Estrategias de depuración y resolución de errores**

Durante el desarrollo se aplicaron distintas estrategias de depuración tanto en frontend como en backend:

* En el backend, se utilizó el sistema de logs de Spring Boot con niveles diferenciados (INFO, DEBUG, ERROR), lo que permitió localizar problemas de forma precisa.
* En el frontend, se aprovechó la consola del navegador para inspeccionar errores de red y fallos de renderizado.
* Se recurrió a **debuggers integrados** en IntelliJ IDEA y Visual Studio Code para analizar el flujo de ejecución.
* Los errores detectados durante las pruebas fueron registrados en un sistema de control de incidencias, y se priorizó su corrección antes de cada entrega.

Estas estrategias permitieron mantener una alta calidad del software en cada iteración, favoreciendo una integración continua fluida y reduciendo el riesgo de errores en producción.

-------------HACER CAPTURAS Y REVISAR EL PUNTO 6 ENTERO----------------------------

**7. DESPLIEGUE Y MANTENIMIENTO**

**7.1. Contenedorización con Docker**

Para la API Rest he querido utilizar Docker, que es una plataforma que permite empaquetar una aplicación con todas sus dependencias dentro de un contenedor. Este contenedor se puede ejecutar en cualquier máquina que tenga Docker instalado, asegurando que la aplicación funcione igual sin importar el entorno.

Esto es muy relevante porque en el caso de esta aplicación lo que se necesita es poder hacer un deploy fácilmente en cualquier tipo de entorno. Como en esta aplicación se necesitan diferentes servicios se ha optado por realizarlo con un Docker Compose que básicamente define todos los servicios en un archivo y los levanta en diferentes contenedores simúltaneamente con el comando: “Docker-compose up”

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Como se puede observar en las imágenes del Docker Compose, tenemos determinados diferentes servicios que va a ejecutar Docker en diferentes contenedores, en este caso el contenedor api y el contenedor mariadb donde se van a encontrar la API Rest en un .jar y la base de datos respectivamente.

Aparte del Docker Compose necesitamos un archivo Dockerfile para que se cree una imagen y se copie el fichero jar del proyecto JAVA y que lo ejecute en Docker.

Texto

El contenido generado por IA puede ser incorrecto.

**7.2. Despliegue del Front**

Para el Front se ha utilizado una plataforma para poder subir a la web la aplicación y que se pueda acceder mediante cualquier dispositivo (Ordenadores, móviles, tablets…), en este caso se ha utilizado Netlify. Para esto se han seguido unos pasos:

*1. Preparar el proyecto React*

* Hay que asegurarse de que el proyecto **compila** con:

bash

CopiarEditar

npm run build

Esto generará una carpeta build/ (en React) con los archivos listos para producción.

*2. Sube tu proyecto a GitHub*

* El proyecto tiene que estar en un repositorio online.

*3. Entra en Netlify*

* Ve a: https://app.netlify.com/
* Inicia sesión (puedes usar tu cuenta de GitHub).

*4. Importa tu repositorio*

* Clica en “Add new site” → “Import an existing project”.
* Conecta tu cuenta de GitHub y elige el repo de tu frontend.

*5. Configura el build*

* **Build command:** npm run build
* **Publish directory:** build
* (Netlify detecta esto automáticamente si es un proyecto React)

*6. Despliegue*

* Dale a “Deploy site”.
* En unos segundos estará online con una URL tipo https://nombredetusitio.netlify.app

**7.3. Estrategia de mantenimiento y actualizaciones**

Una parte muy relevante de un proyecto es el mantenimiento y las actualizaciones que se dan a lo largo de su tiempo de vida.

En cuanto al **mantenimiento** de la aplicación se harán diversas pruebas funcionales a lo largo del tiempo que aseguren el correcto funcionamiento de la aplicación. Así cuando se detecte un error o problema se pueda solucionar rápida y correctamente. Habrá diferentes backups tanto de la API Rest como del Front para si por cualquier problema una versión no funcione se pueda cambiar a otra anterior fácilmente.

En cuanto a las **actualizaciones,** sobre todo las posteriores, se centrarán sobre todo en mejorar la usabilidad por parte del usuario y en diferentes añadidos y mejoras que seran detallados a continuación:

* *Posibilidad de chat con amigos*:

Una de las actualizaciones mas importantes que se quiere implementar en un futuro es la posibilidad de chatear con los amigos agregados. Esto será una tarea relativamente sencilla debido a que se utilizaran peticiones a la API con un nuevo objeto mensaje que enviara y recibirá información mediante GET y POST, y el usuario podrá enviar mensajes directos al amigo que desee.

* *Eventos especiales o temporales*:

Otra posible actualización será la de añadir diversos eventos dependiendo de la época del año, y estos eventos vendrán acompañados de diversos descuentos y sorteos entre los usuarios.

* *Reseñas*:

Un añadido bastante relevante que hacer en el futuro sería el de las reseñas, donde cada cliente puede votar de 1 a 5 estrellas el hotel donde se ha hospedado y dependiendo de esto los clientes al elegir hotel podrán ver su puntuación media para poder comprobar la satisfacción de clientes anteriores. Esto se haría de manera sencilla añadiendo la propiedad puntuación al objeto hotel y por medio de peticiones de los clientes que esta fuera variando.

**8. RESULTADOS Y CONCLUSIONES**

**8.1. Comparación entre objetivos iniciales y resultados obtenidos**

Inicialmente se pusieron unos objetivos, aunque realistas a la vez que algo optimistas, debido a todas las funcionalidades que quise introducir en un primer momento, como la seguridad, las recompensas canjeadas o los amigos; todo esto fue realizado correctamente pero al intentar abarcar tanto no se pulió al detalle cada apartado debidamente, por lo que me di cuenta que es mejor centrarse en ciertas acciones y pulirlas correctamente que intentar abarcar mucho y no lograr el objetivo al detalle.

Aun así tanto el objetivo general como los objetivos específicos se lograron correctamente:

**Objetivo General:** Desarrollar una plataforma completa de gestión de reservas hoteleras que integre frontend, backend y base de datos. ✅

**Objetivos Específicos:**

1. Crear una API REST con Spring Boot para la gestión de reservas. ✅
2. Implementar autenticación y autorización diferenciada para clientes y administradores. ✅
3. Permitir reservas en tiempo real y gestión de recompensas. ✅
4. Desarrollar un frontend responsive e intuitivo en React. ✅
5. Incluir funcionalidades de cancelación y modificación de reservas. ✅
6. Incorporar notificaciones por correo para eventos clave. ✅
7. Asegurar la protección de los datos personales y transacciones. ✅
8. Generar estadísticas de ocupación para los administradores. ❌

Como se puede observar todos los objetivos has sido superados excepto el de estadísticas de ocupación que tiene acceso el administrador, que esto será uno de los añadidos posteriores en las actualizaciones.

Por lo que se puede concluir que los objetivos han sido en su mayoría superados satisfactoriamente, incluso realizando mas funciones de las previamente buscadas como todas las funciones que tiene la API de mail.

**8.2. Problemas encontrados y soluciones aplicadas**

**-------CAMBIAAAAAAAAAAAAAAAAAAAAAAAARRRRRRRRR**

Durante el desarrollo del proyecto se han encontrado diversos problemas de diferentes tipos, tanto a nivel de backend como de frontend. A continuación, se detallan algunos de los más relevantes junto con las soluciones aplicadas:

*1. Ciclo de referencias entre entidades al devolver datos JSON*

Problema*:*  
Al realizar una petición que devolvía información de un cliente junto con sus reservas, se produjo el error:

HttpMessageNotWritableException: Could not write JSON: Document nesting depth (1001) exceeds the maximum allowed (1000)

Este error fue provocado por una relación de bucle entre las entidades Cliente y Reserva, donde cada entidad hacía referencia a la otra, generando una recursividad infinita durante la serialización a JSON.

Solución:  
Se utilizaron las anotaciones @JsonManagedReference y @JsonBackReference para romper el ciclo de serialización, controlando qué parte de la relación se serializa y cuál no. Alternativamente, en algunos casos, se empleó @JsonIgnore cuando no era necesario incluir la propiedad en la respuesta.

En otros el método que se utilizo fue cambiado por ejemplo para en vez de contener el objeto completo solo contenga String por ejemplo con el usuario del Cliente para que así no genere la recursividad.

*2. Datos mostrados incorrectamente en React por error en el acceso a propiedades*

Problema*:*  
En el frontend, al intentar mostrar los amigos de un usuario, se accedía incorrectamente a amigo.body asumiendo que amigo era un objeto. Sin embargo, amigo era simplemente un string, lo que causaba que no se mostrara nada o apareciera undefined.

Solución:  
Se corrigió el código eliminando el acceso erróneo a .body, y se utilizó directamente el valor del string amigo dentro del componente:

<div key={amigo}>{amigo}</div>

3. Persistencia de la autenticación al recargar la página

Problema:  
Inicialmente, al recargar la aplicación React, el usuario autenticado perdía su sesión, ya que el token de autenticación no se almacenaba ni verificaba correctamente.

Solución:  
Se implementó el almacenamiento del token JWT en localStorage y se añadió un mecanismo en el frontend para verificar su validez al iniciar la app. Además, se configuró un filtro en Spring Boot para autenticar las peticiones mediante el token almacenado.

*4. Errores de sincronización entre frontend y backend tras cambios en el modelo*

Problema:  
Después de realizar modificaciones en las entidades del backend (por ejemplo, agregar nuevos campos a Reserva), las peticiones desde React empezaron a fallar debido a diferencias entre lo que el frontend enviaba y lo que el backend esperaba.

Solución:  
Se actualizaron los DTOs, los controladores y los servicios en el backend para adaptarse a los nuevos modelos. Asimismo, se ajustaron los formularios y las peticiones del frontend para mantener la coherencia entre cliente y servidor.

Estos problemas y sus respectivas soluciones han sido fundamentales para consolidar el funcionamiento correcto y estable de la aplicación, permitiendo que tanto la lógica del negocio como la experiencia de usuario final sean robustas y coherentes.

**8.3. Valoración del rendimiento del sistema**

Una parte relevante del proyecto era el rendimiento del sistema, debido a que es una aplicación relativamente compleja que se tiene que conectar a diversos puertos y realizar diferentes conexiones entre el Front, el Back y la Base de datos.

Uno de los problemas que se encontraron fue a la hora de por ejemplo enviar los correos con la API, porque dependiendo de la conexión de donde se encuentre corriendo el Docker con la API, esta realiza las operaciones en un mayor o menor tiempo. Para esto la única solución que se ha encontrado es una actualización que se realizará en un futuro que es correr el Docker de la API en la nube, para que este este siempre desplegada y no haya problemas con el lugar donde este conectada.

Por el resto, se puede observar que el rendimiento es bueno y fluido que es lo que se buscaba en un primer momento, para esto se realizaron pruebas de funcionamiento de la web para comprobar la fluidez tanto en versión móvil como escritorio. Gracias tambien a Spring vemos que las consultas y las conexiones con la Base de Datos se realizan de marera fluida y eficiente. Esto ayuda a que todo el programa funcione de una manera más rápida y sin fallos.

**8.4. Posibles mejoras y ampliaciones futuras**

Como se puede observar y como se ha mencionado anteriormente este proyecto es la primera versión pero el propósito es que se vaya ampliando en el futuro con nuevas funcionalidades y características que se detallan a continuación:

**9. BIBLIOGRAFÍA Y REFERENCIAS**

**Spring Boot Documentation**  
*Spring Framework Reference Documentation.*  
Disponible en: <https://docs.spring.io/spring-boot/docs/current/reference/html/>

**Spring Boot + Spring Data JPA**  
*Guide to Spring Data JPA.*  
Baeldung, 2023.  
Disponible en: https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

**API REST con Spring Boot – Curso Práctico**  
*Desarrollo de servicios REST con Spring Boot y Maven.*  
Código Mentor, 2022.  
Disponible en: https://www.codigomentor.dev/posts/api-rest-con-spring-boot

**Documentación oficial de React**  
*React – A JavaScript library for building user interfaces.*  
Disponible en: https://reactjs.org/docs/getting-started.html

**JWT – Autenticación con tokens**  
*Introduction to JSON Web Tokens.*  
Auth0, 2024.  
Disponible en: https://auth0.com/learn/json-web-tokens/

**Spring Security y JWT**  
*Implementing JWT Authentication on Spring Boot APIs.*  
Dev.to, 2023.  
Disponible en: https://dev.to/keyurparalkar/jwt-authentication-in-spring-boot-3fjj

**Comunicación entre React y Spring Boot**  
*Consuming a RESTful Web Service with React.*  
Spring.io Guides.  
Disponible en: <https://spring.io/guides/tutorials/react-and-spring-data-rest/>

**Postman**  
*Herramienta para pruebas de API REST.*  
Disponible en: <https://www.postman.com/>

**10. ANEXOS**

**10.1. Código fuente relevante**

**10.2. Documentación técnica de la API (Swagger)**

**-------TUTORIAL DE USO SWAGGER---------**

**¿Qué es Swagger?**

**Swagger** (ahora parte del conjunto de herramientas **OpenAPI**) es una herramienta que permite **documentar, visualizar y probar** tu API REST de forma interactiva desde una interfaz web.  
Con Swagger, puedes ver todos tus endpoints disponibles, los métodos (GET, POST, PUT, DELETE), los parámetros, las respuestas y hasta probarlos directamente.

**¿Por qué usar Swagger en tu proyecto?**

* Documentación automática de tu API.
* Interfaz web para probar endpoints sin Postman.
* Ayuda a otros desarrolladores a entender cómo consumir tu API.
* Profesionaliza y mejora la presentación de tu proyecto.

**Cómo integrar Swagger en tu proyecto Spring Boot**

**✅ Paso 1: Agrega las dependencias en tu pom.xml**

Si usas Maven, añade lo siguiente:

xml

CopiarEditar

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.2.0</version>

</dependency>

Para Gradle:

groovy

CopiarEditar

implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.2.0'

**✅ Paso 2: Ejecuta tu proyecto**

Cuando arranques la app (localhost:8080), Swagger estará disponible automáticamente en:

bash

CopiarEditar

http://localhost:8080/swagger-ui.html

o

bash

CopiarEditar

http://localhost:8080/swagger-ui/index.html

Desde esa interfaz podrás:

* Ver todos tus endpoints (por ejemplo: /api/clientes, /api/login)
* Ver parámetros requeridos, tipos de respuesta, códigos HTTP.
* Probar llamadas directamente (enviando parámetros JSON).

**✅ Paso 3: Personaliza la documentación (opcional)**

Puedes añadir anotaciones a tus controladores y métodos para enriquecer la documentación.

Ejemplo:

java

CopiarEditar

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

@RestController

@RequestMapping("/api/clientes")

@Tag(name = "Clientes", description = "Operaciones relacionadas con los clientes")

public class ClienteController {

@GetMapping

@Operation(summary = "Obtener todos los clientes", description = "Devuelve una lista de todos los clientes registrados.")

public List<Cliente> getAllClientes() {

return clienteService.findAll();

}

@PostMapping

@Operation(summary = "Crear un cliente", description = "Registra un nuevo cliente con los datos enviados en el cuerpo de la solicitud.")

public Cliente createCliente(@RequestBody Cliente cliente) {

return clienteService.save(cliente);

}

}

**✅ Paso 4: Generar documentación en formato JSON o YAML**

Tu documentación completa de OpenAPI se puede obtener también en:

bash

CopiarEditar

http://localhost:8080/v3/api-docs

Esto te permite exportarla si la necesitas para otra herramienta.