Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DE SOFTWARE

Creación de API backend basado en Spring Boot para el desarrollo de un sistema de gestión de actividades deportivas

Alumno: Manuel Comesaña Couto

Tutor: Benjamín Sahelices Fernández

A mi abuela Carmen y a mi primo Pedro.

Agradecimientos

Quiero agradecer especialmente a mi madre y a mi padre por todo el esfuerzo que han realizado para brindarme infinidad de oportunidades.
A mis abuelos por ser un pilar sólido y fundamental en mi educación.
A mis tíos y a mi primo por el apoyo y preocupación diaria.

Resumen

El presente TFG propone desarrollar una API de gestión de reservas y un bot de Telegram. La API permitirá a los usuarios realizar reservas, consultar disponibilidad y obtener información detallada. El bot servirá como una interfaz adicional para interactuar con el sistema de reservas. Ambos componentes contarán con un sistema de registro y almacenamiento de historial. El objetivo es facilitar la realización y administración de reservas de manera eficiente y conveniente.

Abstract

The present Bachelor's Thesis proposes the development of a reservation management API and a Telegram bot. The API will allow users to make reservations, check availability, and obtain detailed information. The bot will serve as an additional interface to interact with the reservation system. Both components will feature a registration and history storage system. The goal is to facilitate the efficient and convenient process of making and managing reservations.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto

El padel se ha convertido en uno de los deportes con mas crecimiento de la actualidad. El numero de clubes en Europa a crecido un 181 %. Según un estudio realizado por Monitor Deloitte y de Playtomic pronostican que puede convertirse en uno de los deportes mas practicados en el mundo por delante del tenis, el numero de pistas de padel se ha triplicado desde el 2016. [1]

Además del creciente interés en el padel como deporte, también es importante destacar la necesidad de soluciones de software para apoyar a la industria del padel en su crecimiento. Con el aumento del número de clubes y pistas de padel en Europa y en todo el mundo, existe una creciente demanda de soluciones de software que ayuden a los jugadores, los organizadores y los propietarios de clubes a gestionar los eventos, las reservas y la administración en general.

1.2. Motivación

La idea de este proyecto, surgió al observar la dificultad para usuarios para la reserva de una simple pista. Además el coste elevado para los clubs de mantener una aplicación propia o contratar a empresas con su propio software, de forma que los responsables de los clubes de padel compartan la información de sus clientes y facturación con terceros.

A medida que los clubes se vuelven más grandes y complejos, gestionar la programación de pistas, las reservas y la administración en general se vuelve cada vez más difícil. Los propietarios de clubes pueden verse abrumados por la cantidad de reservas y solicitudes que reciben, lo que puede resultar en errores y confusiones. Los jugadores a menudo encuentran que reservar una pista de padel en su club local es complicado y requiere mucho tiempo, lo que puede ser un obstáculo para aquellos que desean jugar con regularidad.

Al proporcionar una plataforma centralizada para la gestión de reservas y la administración de clubes, la aplicación puede simplificar y automatizar muchos de los procesos que actualmente requieren tiempo y esfuerzo manual. Los propietarios de clubes pueden utilizar la aplicación para programar torneos y eventos, administrar los ingresos y gastos y tener una visión general del rendimiento de su club. Los jugadores pueden reservar pistas con facilidad, acceder a información sobre los horarios de los partidos y los torneos y mantener un registro de su historial de reservas.

La motivación para crear una aplicación de gestión y reservas de padel es la necesidad de simplificar y automatizar los procesos de administración de clubes y reservas, para que los propietarios de clubes puedan enfocarse en ofrecer una experiencia de alta calidad para los jugadores, y para que los jugadores puedan disfrutar del juego sin la frustración de un proceso complicado y engorroso de reserva de pistas.

Dada la existencia de un grupo de Telegram de 1400 usuarios entusiastas del pádel que utilizan para organizar partidos. El problema que resuelve es la automatización y simplificación del proceso de reserva de pistas, eliminando la necesidad de que un administrador del club actúe como intermediario entre los usuarios y la aplicación de reservas. Se implementara un bot de Telegram que los clientes utilizaran de Telegram para reservar una pista de pádel de manera rápida y sencilla. Podrán ver la disponibilidad de las pistas, seleccionar la fecha y hora deseada, e incluso buscar jugadores con un nivel similar para formar partidos. Esta funcionalidad permitirá a los usuarios encontrar compañeros de juego que compartan su pasión por el pádel y tengan un nivel de habilidad comparable.Al automatizar el proceso de reserva de pistas, el bot reduce la carga de trabajo de los administradores y gestores del club, al tiempo que brinda a los jugadores una forma más rápida y sencilla de organizar y confirmar sus participaciones en los partidos, ofrece funcionalidades adicionales, como notificaciones automáticas de cambios en las reservas, recordatorios de partidos y la capacidad de gestionar la lista de jugadores interesados en cada partida.

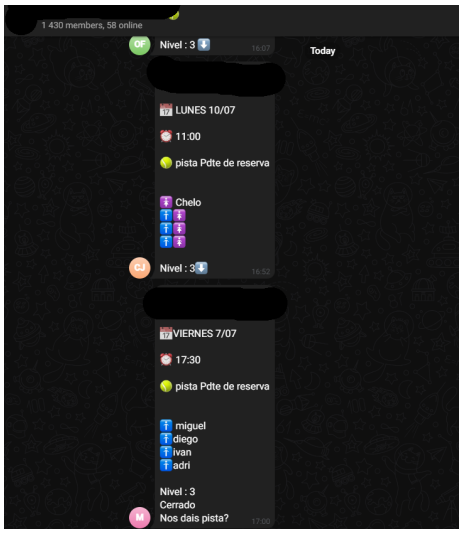

Figura 1.1: Captura del grupo de reservas de Telegram.

1.3. Alternativas

Uno de las mayores plataformas de gestión y reservas tanto de tenis como de padel es Playtomic.[2] Integra mas de 500 clubes en una sola plataforma. Entre sus funcionalidades se encuentran las siguientes:

- Comunicación
  - Chat de grupo
  - Envío y recepción de SMS
  - Gestor de contactos
- Ecommerce
  - Gestión de reservas
- Gestión de proyectos
  - Gestión de incidentes
  - Gestión de recursos
- Gestión del personal
  - Administración del personal
  - Control de ausencias
  - Estimación del tiempo
  - Gestión de tiempo de trabajo
  - Hoja de asistencia
  - Horarios flexibles
  - Seguimiento de la evolución
- Asistencia
  - Asistencia (teléfono, correo electrónico, chat)
  - Asistencia a la configuración
  - Base de conocimientos (tutoriales, demos)
  - Formación en linea (seminario web)
  - Gestor de cuentas dedicado

1.4. Nicho de mercado y usuarios objetivo

Tras hacer un estudio profundo con responsables de clubes de padel, se puede deducir que el principal objetivo de mercado son los clubes a nivel nacional y usuarios o socios activos vinculados a estas instalaciones.

Cada propietario podrá tener a su alcance toda la información y gestión de sus pistas y clientes. La interfaz en Telegram proporcionará a los propietarios de las pistas y a los clientes una forma fácil y accesible de acceder a toda la información y gestionar sus reservas de pistas y clientes, de una forma muy similar a la que ya conocian mediante la experiencia del grupo de Telegram. Para los propietarios, podrán ver y administrar el estado de las pistas, gestionar las reservas y recibir notificaciones relevantes. También podrán acceder a la información de los clientes y realizar un seguimiento de sus actividades.

Los usuarios objetivos para esta solución serán:

- Jugadores del grupo de Telegram: Aquellos que participan activamente en la organización de partidos y desean tener un proceso más eficiente para realizar reservas de pistas. El bot les permitiría crear y publicar partidos, añadir jugadores interesados y gestionar las reservas de forma automática.
- Administradores del club: Los encargados de verificar la disponibilidad de las pistas y confirmar las reservas en la aplicación correspondiente. El bot les proporcionaría una herramienta para agilizar este proceso al automatizar la comunicación con los jugadores y gestionar las reservas directamente.
- Gestores del club: Aquellos responsables de la gestión general del club y la asignación de las pistas. El bot les permitiría liberar su carga de trabajo al eliminar la necesidad de estar pendientes de la disponibilidad de las pistas específicas donde se llevarán a cabo los partidos, ya que el bot se encargaría de realizar las reservas de forma automática.

El desarrollo del bot de Telegram para este grupo de usuarios entusiastas del pádel y el club correspondiente permite automatizar el proceso de reserva de pistas, mejorar la experiencia de los jugadores y liberar al personal del club de tareas administrativas. Esto satisface una necesidad específica en el nicho de mercado al simplificar la organización de partidos y optimizar la gestión de las reservas de pistas.

1.5. Objetivos del proyecto

Esta aplicación API Rest tiene como objetivo principal proporcionar una plataforma completa y eficiente para la gestión de reservas y usuarios tanto para los clientes como para los propietarios de clubes. Es de vital importancia mantener una buena arquitectura que nos permita cumplir con los requisitos actuales y futuros, así como prepararnos para una futura implementación de la interfaz con React Native.

La aplicación consta de dos partes: la gestión de reservas y usuarios por parte del cliente, y la gestión del club por parte del propietario. La primera parte permite a los clientes realizar reservas de pistas de pádel de forma sencilla y rápida, además de buscar jugadores con niveles de habilidad similares para formar partidos. La interfaz intuitiva y amigable facilita la gestión de reservas y actividades relacionadas.

Por otro lado, la gestión del club brinda a los propietarios las herramientas necesarias para administrar las reservas de las pistas, controlar la disponibilidad y el estado de las mismas, y gestionar a los usuarios. La plataforma proporciona información detallada, informes y estadísticas relevantes, y notificaciones importantes para facilitar la operación y el crecimiento del club.

La importancia de mantener una buena arquitectura radica en la capacidad de adaptarse y ampliar las funcionalidades en el futuro sin comprometer la estabilidad del sistema. Además, se prevé una futura implementación de la interfaz utilizando React Native, lo que permitirá ofrecer una experiencia nativa y consistente en diferentes plataformas móviles.

Esta aplicación API Rest tiene como objetivo satisfacer las necesidades de los clientes y propietarios de clubes en la gestión de reservas y usuarios. Con una arquitectura sólida y una futura implementación con React Native, se busca proporcionar una solución completa y eficiente, brindando una experiencia intuitiva y amigable en el mundo del pádel.

1.5.1. Objetivos del desarrollo

El principal objetivo es el desarrollo de una herramienta de consulta que comunique las ofertas del propietario y sacie la demanda de los clientes. Se creará una interfaz cómoda para que el usuario reserve cualquier recurso de la forma mas factible posible. El propietario de las instalaciones será el único que maneje la información relevante de su club, datos de clientes facturación y ocupación.

1. Diseñar una arquitectura escalable y modular que permita la incorporación de nuevas funcionalidades y la adaptación a futuros requisitos del negocio.
2. Desarrollar una API Rest robusta y segura que garantice la integridad de los datos y proteja la privacidad de los usuarios.
3. Implementar un sistema de autenticación y autorización sólido para garantizar que solo los usuarios autorizados puedan acceder y realizar acciones en la plataforma.
4. Crear endpoints claros y bien documentados que faciliten la interacción con la API y permitan una fácil integración con otras aplicaciones o servicios.
5. Optimizar el rendimiento de la aplicación, minimizando los tiempos de respuesta y maximizando la eficiencia en el uso de los recursos del servidor.
6. Realizar pruebas exhaustivas y asegurar la calidad del código a través de pruebas unitarias, de integración y de rendimiento.

7. Implementar mecanismos de seguridad, como encriptación de datos y protección contra ataques comunes, para garantizar la seguridad de la información almacenada en la aplicación.

8. Establecer una estructura de base de datos eficiente y optimizada que permita un acceso rápido y preciso a la información relevante.

9. Facilitar la futura implementación de la interfaz con React Native, asegurando una comunicación fluida entre la API y la interfaz de usuario móvil.

10. Cumplir con los estándares de desarrollo de software, siguiendo buenas prácticas de programación, documentación clara y mantenimiento del código limpio y legible.

1.5.2. Objetivos de la aplicación

La aplicación de gestión de reservas de pistas de pádel es la solución perfecta para simplificar y optimizar el proceso de reserva de pistas, tanto para los entusiastas jugadores como para los propietarios de clubes deportivos. Con una interfaz intuitiva y funcionalidades avanzadas, esta aplicación permite a los usuarios reservar pistas de forma rápida y sencilla, mientras que los propietarios tienen un control completo sobre la gestión de sus instalaciones. La aplicación de gestión de reservas de pistas de pádel tiene como principales objetivos:

1. Proporcionar a los clientes una plataforma intuitiva y fácil de usar para gestionar sus reservas de pistas de pádel de forma rápida y sencilla.

2. Permitir a los propietarios de clubes deportivos tener un control completo sobre la gestión de sus instalaciones, incluyendo la disponibilidad de pistas, horarios y precios.

3. Facilitar la reserva de pistas entre usuarios que comparten la pasión por el pádel y tienen niveles de juego similares, fomentando así la formación de partidos equilibrados y emocionantes.

4. Ofrecer un sistema de autenticación seguro que garantice la privacidad y la protección de los datos personales de los usuarios.

5. Proporcionar información detallada sobre las pistas disponibles, incluyendo características y horarios de apertura, para que los usuarios puedan tomar decisiones informadas al realizar sus reservas.

6. Brindar un servicio de notificaciones y recordatorios para informar a los usuarios sobre el estado de sus reservas, cambios de horario o cualquier otra información relevante.

7. Proporcionar estadísticas y análisis sobre las reservas realizadas, permitiendo a los propietarios de clubes tomar decisiones estratégicas para optimizar la utilización de las pistas y mejorar la experiencia de los usuarios.

8. Garantizar un alto nivel de disponibilidad y rendimiento de la aplicación, para que los usuarios puedan acceder y utilizar el servicio en cualquier momento y desde cualquier lugar.

9. Facilitar la futura integración con otras plataformas o servicios, permitiendo la expansión y colaboración con otras aplicaciones o sistemas relacionados con el pádel.

1.5.3. Estructura de la memoria

A continuación se presenta la estructura del documento:

1. **Introducción**: En este primer apartado se presentan las motivaciones y los objetivos del proyecto.
2. **Tecnologías y servicios**: En este segundo apartado se presenta el plan de proyecto para conseguir los objetivos, así como el plan de control y el seguimiento del proyecto.
3. **Planificación y desarrollo del proyecto**: En este tercer apartado se presenta la parte de planificación y metodologías empleadas.
4. **Análisis y diseño**: En este cuarto apartado se presenta el diseño de la arquitectura del sistema y se detallan los patrones de diseño utilizados.
5. **Implementación**: En este sexto apartado se presenta la implementación de la aplicación y se destacan las partes más características de la arquitectura.
6. **Pruebas**: En este séptimo apartado se presentan las pruebas de los end-points.
7. **Conclusiones y líneas futuras**: En este octavo apartado se presentan las conclusiones obtenidas al finalizar el proyecto y se plantean las líneas futuras de trabajo.
8. **Anexo A: Manual de Usuario**: Este anexo contiene el manual de la API y del bot de Telegram.
9. **Anexo B: Contenidos del CD-ROM**: Referencia al proyecto entregado.

Capítulo 2

Tecnologías y servicios

Para cumplir con los objetivos del proyecto, se utilizarán diversas tecnologías y servicios para desarrollar el API de la aplicación web de gestión y reservas para el club deportivo de padel. En particular, se utilizará el framework de desarrollo de aplicaciones web Java Spring Boot para desarrollar el API, ya que es una herramienta poderosa y flexible que permite desarrollar aplicaciones web eficientes y escalables de manera rápida y sencilla.

En cuanto a la base de datos, se utilizará MySQL, una base de datos relacional de código abierto y ampliamente utilizada. MySQL se utiliza con frecuencia en proyectos de este tipo debido a su fiabilidad, escalabilidad y compatibilidad con la mayoría de los lenguajes de programación.

Además, se utilizarán diversos servicios en la nube para alojar y ejecutar el API, así como para gestionar los datos del club y de los usuarios. Entre los servicios de la nube utilizados, se incluyen AWS (Amazon Web Services) y Google Cloud Platform.

El uso de estas tecnologías y servicios permitirá el desarrollo de un API robusto, seguro y escalable para la aplicación web de gestión y reservas del club deportivo de padel, mejorando la eficiencia y eficacia de la gestión del club y proporcionando una experiencia de reserva de pistas de padel más cómoda y agradable para los usuarios.

2.1. Git

Git es una herramienta de control de versiones de código abierto y gratuita. Su función principal es gestionar los cambios realizados en un proyecto de software y su configuración. Utilizando repositorios, Git almacena los archivos de un proyecto y permite dividirlo en ramas. Cada rama es una copia exacta del proyecto en un momento dado.

Git es un sistema distribuido, lo que significa que puede coordinar repositorios locales y remotos. Esto permite mantener una copia estática del proyecto de forma remota mientras se

realizan cambios y pruebas a nivel local. Una vez que se analiza el trabajo, las ramas pueden fusionarse para combinar los cambios con la versión anterior mediante la acción de "merge".

Para realizar un seguimiento de los cambios en el repositorio local, es necesario agregar los archivos modificados al área de stage y luego hacer un commit, que permite añadir un mensaje descriptivo de los cambios realizados. Los commits locales se sincronizan con el repositorio remoto mediante un push, actualizando la versión en el repositorio remoto y reflejando el historial de commits del repositorio local.

Proporciona un control de versiones efectivo al permitir el seguimiento de cambios, la gestión de ramas y la sincronización entre repositorios locales y remotos. Esto facilita la colaboración y la organización en el desarrollo de proyectos de software.

2.2. Spring framework

Spring Framework es un marco de desarrollo empresarial en Java que se basa en el patrón de diseño de Inversión de Control y Programación Orientada a Aspectos para simplificar el desarrollo de aplicaciones empresariales escalables y fáciles de mantener.

En cualquier aplicación multiplataforma la programación Back-End es un pilar base para la lógica de negocio. Es un framework de código abierto para desarrollar aplicaciones empresariales en Java. Fue creado por Rod Johnson en 2003 y ha sido ampliamente adoptado en la comunidad de desarrollo de Java debido a su enfoque modular, escalable y fácil de usar.

El marco Spring se basa en el patrón de diseño de Inversión de Control (IoC) y el patrón de diseño de Programación Orientada a Aspectos (AOP) para simplificar el desarrollo de aplicaciones empresariales en Java. Spring proporciona un conjunto de características y herramientas que permiten a los desarrolladores centrarse en la lógica de negocio en lugar de preocuparse por las tareas de bajo nivel, como la gestión de la conexión a la base de datos o la gestión de transacciones.

Algunas de las características clave de Spring Framework incluyen:

- IoC (Inversión de Control): permite que los objetos sean creados y gestionados por el contenedor de Spring, lo que facilita la inyección de dependencias.
- AOP (Programación Orientada a Aspectos): permite la separación de las preocupaciones transversales de una aplicación, como la seguridad y la gestión de transacciones.
- Integración con otros marcos y bibliotecas: Spring se integra con otros marcos y bibliotecas comunes, como Hibernate, Struts y JSF, lo que permite a los desarrolladores trabajar con sus herramientas favoritas.
- Soporte para desarrollo basado en pruebas: Spring proporciona soporte para pruebas unitarias y de integración, lo que permite a los desarrolladores probar su código de manera eficiente.

2.2.1. Spring Security

Spring Security es un poderoso marco de seguridad para aplicaciones Java. Proporciona una amplia gama de características de seguridad, incluyendo autenticación, autorización y gestión de roles.

Spring Security se basa en el patrón de Inversión de Control (IoC) de Spring Framework y utiliza una arquitectura modular que permite una fácil integración con otras bibliotecas y marcos de seguridad. Además, proporciona una amplia gama de proveedores de autenticación, incluyendo autenticación basada en formularios, autenticación basada en tokens o autenticación basada en LDAP (Lightweight Directory Access Protocol).

Es altamente personalizable, lo que le permite adaptarse a las necesidades específicas de una aplicación. Por ejemplo, puede configurar diferentes reglas de autorización para diferentes URLs, y puede personalizar los mensajes de error que se muestran a los usuarios en caso de que no puedan acceder a una página o recurso específico.

Es una herramienta esencial para garantizar la seguridad de las aplicaciones Java, especialmente aquellas que manejan información confidencial o de usuarios.

2.2.2. SpringBoot

Spring Boot es una elección popular para desarrollar una API REST debido a su simplicidad, configuración por convención, iniciadores preconfigurados, comunidad activa, integración con el ecosistema Spring y facilidad de despliegue. Estos aspectos combinados te permiten desarrollar rápidamente una API REST de alta calidad y centrarse en la lógica de negocio en lugar de preocuparse por configuraciones y tareas repetitivas.[3]

Spring Boot es un framework para el desarrollo de aplicaciones Java que facilita la creación de APIs RESTful de forma rápida y sencilla. Algunas de las ventajas por las que se escogido utilizar Spring Boot para crear un API son las siguientes:

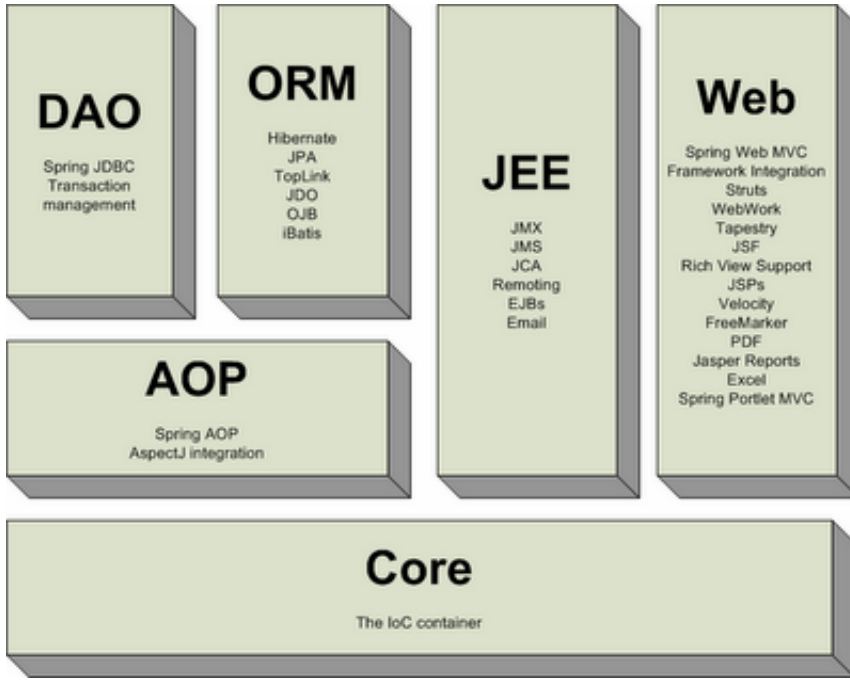

Figura 2.1: Principales módulos de spring.

- Facilita el desarrollo: ofrece una estructura de proyecto bien definida y una gran cantidad de librerías y herramientas que facilitan el desarrollo de una API RESTful.

- Reduce la configuración: reduce la cantidad de configuración necesaria para crear una aplicación, lo que permite a los desarrolladores centrarse en el desarrollo de la lógica de negocio.

- Integración con otros frameworks: se integra fácilmente con otros frameworks de Spring, como Spring Data, Spring Security, Spring Cloud, entre otros, lo que permite a los desarrolladores utilizar una amplia gama de herramientas para crear una API.

- Automatización: proporciona una serie de herramientas para la automatización de pruebas, construcción y despliegue, lo que facilita el proceso de desarrollo y reduce los errores.

- Escalabilidad: se puede utilizar para crear aplicaciones escalables y de alta disponibilidad, lo que es esencial para una API RESTful que maneja una gran cantidad de solicitudes.

- Documentación automática: incluye herramientas que permiten la generación automática de documentación de API, lo que facilita el mantenimiento y la comprensión de la API para los desarrolladores y usuarios.

En general, es una herramienta poderosa y flexible que facilita el proceso de creación de una API RESTful, lo que a su vez reduce el tiempo de desarrollo y mejora la eficiencia del proceso.

2.3. OpenAPI

OpenAPI es una especificación de API abierta y estandarizada que permite a los desarrolladores describir, documentar, probar y utilizar una API de manera más fácil y eficiente. Anteriormente conocida como Swagger, OpenAPI es un estándar de la industria utilizado por desarrolladores, empresas y organizaciones de todo el mundo para diseñar y construir APIs interoperables y fáciles de usar. La especificación OpenAPI se describe en un documento JSON o YAML que define los puntos finales, los parámetros, las respuestas, los esquemas de datos y otra información relevante para interactuar con una API de manera consistente y predecible.[4]

Las ventajas de OpenAPI son diversas y se pueden resumir en los siguientes puntos:

- Estandarización: OpenAPI proporciona un estándar común para describir y documentar una API, lo que facilita la integración y la colaboración entre equipos y empresas.
- Documentación automática: Al describir una API con OpenAPI, se puede generar automáticamente documentación interactiva para la API, lo que ahorra tiempo y reduce la posibilidad de errores en la documentación.
- Generación de código: Las herramientas de generación de código pueden utilizar la especificación OpenAPI para generar código cliente en una variedad de lenguajes de programación, lo que acelera el desarrollo de aplicaciones que consumen una API.
- Facilidad de prueba: OpenAPI proporciona una descripción completa de los puntos finales y parámetros de una API, lo que facilita la creación de pruebas automatizadas para asegurar que una API se comporta como se espera.
- Interoperabilidad: OpenAPI es un estándar abierto y ampliamente adoptado, lo que significa que las API diseñadas utilizando OpenAPI son más fáciles de integrar con otras API y herramientas que utilizan el mismo estándar.
- Facilidad de mantenimiento: Al describir una API con OpenAPI, los cambios y actualizaciones se pueden hacer en la especificación en lugar de en la documentación y el código, lo que simplifica el proceso de mantenimiento y reduce la posibilidad de errores.

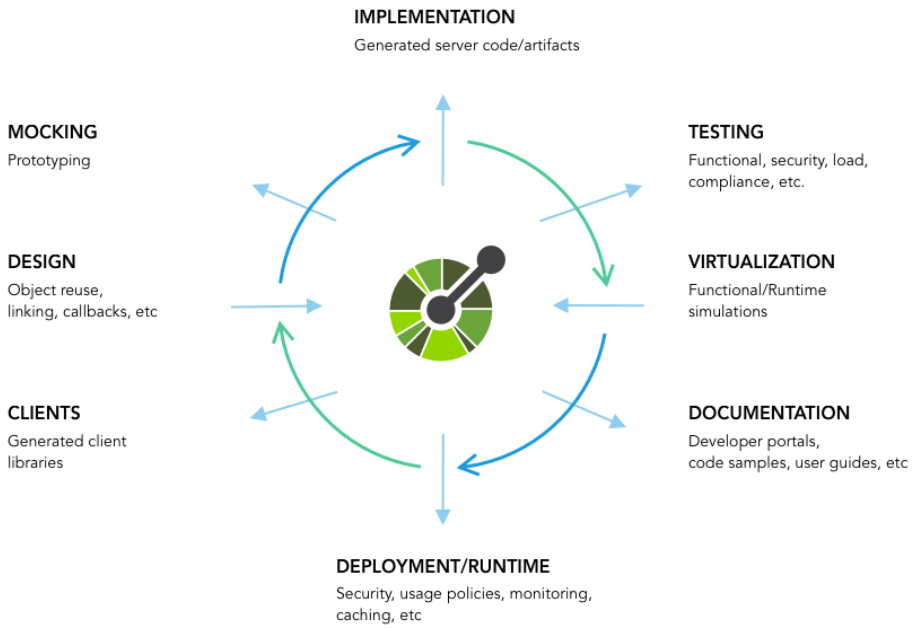

Figura 2.2: Beneficios de la especificación OpenAPI

2.4. Trello

Trello es una herramienta de gestión de proyectos en línea que utiliza un sistema de tarjetas organizadas en tableros para ayudar a los usuarios a visualizar y controlar sus tareas y proyectos. Cada tarjeta representa una tarea o un conjunto de tareas relacionadas, y se puede arrastrar y soltar entre columnas para indicar su estado (por ejemplo, "por hacer", "en proceso", "hecho", etc.). [5]

Además de las tarjetas, Trello también cuenta con otras características útiles, como etiquetas de color para identificar rápidamente las tareas importantes, comentarios para la comunicación entre los miembros del equipo y listas de comprobación para dividir las tareas en subtareas más manejables. También es posible adjuntar archivos, asignar tareas a miembros específicos del equipo y establecer fechas límite para las tareas.

Trello es ampliamente utilizado por equipos de desarrollo de software, equipos de marketing, equipos de proyectos y cualquier persona o equipo que necesite una herramienta visual y fácil de usar para gestionar tareas y proyectos. Además, Trello ofrece una variedad de integraciones con otras herramientas populares, como Slack, Google Drive y Jira, lo que la convierte en una herramienta altamente adaptable y personalizable.

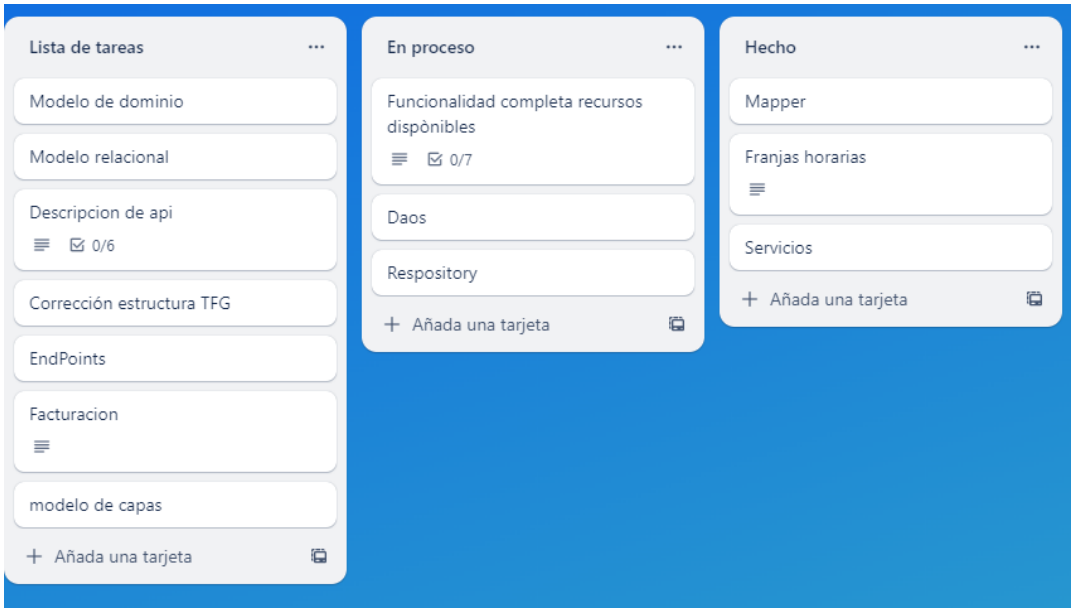

Figura 2.3: Ejemplo de uso de la herramienta trello

2.5. Draw.io

Draw.io es una herramienta en línea de creación de diagramas y gráficos que permite crear una gran variedad de diagramas, como diagramas de flujo, diagramas de red, diagramas UML, diagramas ER, organigramas, mapas mentales, entre otros. Es una herramienta de uso libre, es decir, se puede utilizar de manera gratuita sin necesidad de registrarse.

Draw.io ofrece muchas opciones para crear diagramas, incluyendo la capacidad de importar y exportar documentos de otros servicios de almacenamiento en línea, como Google Drive, Dropbox y OneDrive. También permite compartir y colaborar con otros usuarios en tiempo real.

2.6. MySql

MySQL es un sistema de gestión de bases de datos relacionales (RDBMS) de código abierto que utiliza SQL (Lenguaje de Consulta Estructurada) para gestionar y manipular datos. Es una de las bases de datos más populares del mundo y es utilizada por muchas organizaciones y sitios web.

MySQL es conocido por su velocidad, fiabilidad y facilidad de uso. Es compatible con muchos sistemas operativos, incluyendo Windows, Linux y Mac OS X. MySQL puede ser

utilizado para una variedad de aplicaciones, como sistemas de gestión de contenidos, sitios web de comercio electrónico y aplicaciones web. [6]

MySQL se utiliza a menudo en combinación con PHP, un lenguaje de programación popular para el desarrollo web. Juntos, MySQL y PHP forman una combinación poderosa para construir aplicaciones web dinámicas e interactivas.

Algunas de las características clave de MySQL incluyen:

- Soporte para múltiples motores de almacenamiento, incluyendo InnoDB, MyISAM y Memory.
- Soporte multiplataforma.
- Replicación de datos para alta disponibilidad y escalabilidad.
- Capacidades de búsqueda de texto completo.
- Funciones de seguridad, como encriptación y conexiones seguras.
- Soporte para procedimientos almacenados y disparadores.

MySQL es software libre y de código abierto, lo que significa que cualquiera puede descargarlo y usarlo sin pagar ninguna tarifa de licencia.

2.7. Maven

Maven es una herramienta popular de automatización de construcción utilizada principalmente para proyectos en Java. Ayuda a gestionar las dependencias del proyecto, los procesos de construcción y la documentación del proyecto. Maven utiliza un archivo de configuración llamado "pom.xml"(Project Object Model) que describe la estructura del proyecto, las dependencias y los complementos.

Maven simplifica el proceso de construcción al proporcionar un conjunto de convenciones y mejores prácticas para la organización del proyecto. También proporciona una amplia gama de complementos que se pueden usar para realizar tareas específicas de construcción, como la compilación, el empaquetado y la prueba del proyecto.

Utiliza repositorios de artefactos para almacenar y distribuir las dependencias de un proyecto. Los repositorios pueden ser locales (en la máquina del desarrollador), remotos (en un servidor de la red) o centrales (como Maven Central).

Maven es una herramienta muy popular en el desarrollo de software en Java y es ampliamente utilizada por los desarrolladores de software para automatizar el proceso de construcción y gestión de dependencias.

2.8. MySQL Workbench

Es una herramienta de software de gestión de bases de datos relacionales que se utiliza para diseñar, modelar, desarrollar y administrar bases de datos MySQL. La herramienta proporciona una interfaz gráfica para interactuar con la base de datos, lo que facilita la creación de tablas, la creación de relaciones entre tablas, la definición de índices, la creación de consultas y la visualización de resultados.

MySQL Workbench [7] también incluye características adicionales como la visualización de la estructura de la base de datos en un diagrama ER (Entity-Relationship), la posibilidad de realizar ingeniería inversa para generar un modelo de base de datos a partir de una base de datos existente, y la capacidad de generar informes detallados sobre el rendimiento de la base de datos.

MySQL Workbench es una herramienta de software muy útil para aquellos que trabajan con bases de datos MySQL, ya sea para la creación de nuevas bases de datos o para la administración y el mantenimiento de bases de datos existentes.

2.9. Telegram

Telegram es una aplicación de mensajería instantánea y plataforma de comunicación que permite a los usuarios enviar mensajes, realizar llamadas de voz y video, compartir archivos y crear grupos de chat. Fue lanzada en 2013 y se ha vuelto muy popular debido a su enfoque en la privacidad y seguridad de los datos. Algunas características clave de Telegram para este proyecto son:

Una función importante para este proyecto es que permite a los desarrolladores crear aplicaciones interactivas dentro de la aplicación. Los bots pueden proporcionar información, realizar acciones automatizadas y ofrecer servicios adicionales a los usuarios. Se puede acceder a Telegram desde múltiples dispositivos, incluyendo teléfonos móviles, tablets y ordenador. Las conversaciones se mantienen sincronizadas entre ambos dispositivo. Telegram ofrece una API abierta que permite a los desarrolladores crear sus propias aplicaciones y servicios integrados con la plataforma. Esto ha llevado al desarrollo de una amplia gama de bots y aplicaciones de terceros que amplían las funcionalidades de Telegram [8].

2.10. XAMPP

XAMPP es una solución todo en uno para configurar un entorno de servidor web local en tu máquina. Proporciona los componentes necesarios, como Apache, MySQL, PHP y Perl, para desarrollar y probar aplicaciones web de forma fácil y conveniente. Es una herramienta popular entre los desarrolladores que desean tener un entorno de desarrollo web local sin la necesidad de configurar cada componente por separado.

Las principales características y componentes de XAMPP [9] son:

- **Apache**: Es un servidor web de código abierto ampliamente utilizado. XAMPP incluye Apache como servidor web predeterminado, lo que te permite alojar tus aplicaciones y sitios web localmente.
- **MySQL**: Es un sistema de gestión de bases de datos relacional. XAMPP proporciona MySQL como el sistema de gestión de bases de datos predeterminado, lo que te permite crear y administrar bases de datos locales para tus aplicaciones web.
- **PHP**: Es un lenguaje de programación de código abierto especialmente diseñado para el desarrollo web. XAMPP viene con PHP preinstalado y configurado, permitiéndote ejecutar y probar fácilmente tus aplicaciones web basadas en PHP en tu entorno local.
- **Perl**: Es un lenguaje de programación de scripting ampliamente utilizado en el desarrollo web y otros campos. XAMPP incluye Perl para aquellos que necesitan utilizar este lenguaje en sus aplicaciones.
- **phpMyAdmin**: Es una herramienta de administración de bases de datos MySQL basada en web. XAMPP incluye phpMyAdmin, lo que te permite administrar tus bases de datos de forma visual y realizar tareas como crear tablas, ejecutar consultas SQL y realizar copias de seguridad.

Además de estos componentes principales, XAMPP también incluye otros complementos y utilidades que son útiles para el desarrollo web, como FileZilla (un cliente FTP) y Mercury Mail (un servidor de correo electrónico local para pruebas).

2.11. Postman

Postman [10] es una herramienta popular para probar y documentar API. Permite realizar solicitudes HTTP a diferentes puntos finales de una API, enviar diferentes tipos de datos (como JSON o formularios) y recibir las respuestas correspondientes. Postman proporciona una interfaz intuitiva y amigable para interactuar con las API, lo que facilita la prueba y depuración de las mismas.

Algunas características destacadas de Postman incluyen:

- **Interfaz de usuario intuitiva**: Postman tiene una interfaz fácil de usar que permite realizar solicitudes HTTP de manera rápida y sencilla. Puedes crear solicitudes personalizadas, especificar encabezados y parámetros, y ver las respuestas en formato legible.
- **Colecciones y entornos**: Postman te permite organizar tus solicitudes en colecciones, lo que facilita la gestión y ejecución de pruebas en diferentes escenarios. Además, puedes definir entornos para manejar variables de entorno y hacer pruebas en diferentes configuraciones.

- **Pruebas automatizadas**: Postman te permite escribir scripts de pruebas automatizadas usando JavaScript. Esto te permite realizar pruebas funcionales y de regresión para verificar que tu API funcione correctamente.
- **Generación de documentación**: Con Postman, puedes generar automáticamente documentación para tu API a partir de tus solicitudes y respuestas. Esto facilita la creación de documentación técnica para desarrolladores y usuarios de tu API.
- **Integraciones y colaboración**: Postman se integra con otras herramientas y servicios, como Git y herramientas de control de versiones, lo que facilita la colaboración entre miembros del equipo. También puedes compartir tus colecciones de Postman con otros usuarios y colaborar en tiempo real.

Figura 2.4: Logotipo de postman

2.11.1. IntelliJ IDEA

IntelliJ IDEA [11] es un IDE (Integrated Development Environment) creado por JetBrains que se utiliza principalmente para el desarrollo de software en lenguajes como Java, Kotlin, Groovy y Scala, entre otros. Este entorno de desarrollo integrado está disponible en versiones gratuitas y de pago.

IntelliJ IDEA ofrece una serie de características destacadas, como la navegación inteligente del código, la refactorización de código, la detección de errores en tiempo real, la integración con sistemas de control de versiones como Git y SVN, la capacidad de desarrollar aplicaciones para diferentes plataformas, y la gestión de dependencias, entre otras.

Además, IntelliJ IDEA cuenta con una gran comunidad de usuarios y una amplia documentación y soporte en línea. Es utilizado por muchos desarrolladores y empresas de todo el mundo para desarrollar software de alta calidad.

Entre los plugins más utilizados se encuentran Spring Boot Tools, que proporciona una integración completa con Spring Boot para la creación y el mantenimiento de aplicaciones Spring Boot, Spring Initializr Java Support, que proporciona plantillas para la creación de aplicaciones Spring Boot y Spring Boot Dashboard, que proporciona una interfaz gráfica para la gestión y el monitoreo de aplicaciones Spring Boot.

2.12. Swagger Editor

Swagger Editor es una herramienta en línea gratuita que proporciona un entorno de desarrollo y prueba para diseñar y visualizar especificaciones OpenAPI (anteriormente conocidas como Swagger). Permite crear y editar archivos YAML o JSON que describen los endpoints, las operaciones, los modelos de datos y otra información relacionada con la API.

Las principales funcionalidades de Swagger Editor son:

- **Diseño de la API**: Permite definir los endpoints, los parámetros de solicitud, las respuestas esperadas y otros detalles de la API utilizando una interfaz interactiva y amigable. Swagger Editor ayuda a mantener una estructura y formato adecuados en la especificación OpenAPI.
- **Validación de la especificación**: Verifica automáticamente la sintaxis y la estructura del archivo OpenAPI para asegurarse de que cumpla con la especificación. Esto ayuda a detectar posibles errores o inconsistencias en el diseño.
- **Visualización de la API**: Genera una documentación interactiva basada en la especificación OpenAPI. Permite explorar y probar los endpoints directamente desde el editor, facilitando la comprensión y prueba de la API.
- **Realización de pruebas**: Permite enviar solicitudes HTTP a los endpoints desde el editor para verificar el comportamiento de la API. Esto permite probar y depurar la API antes de implementarla en un entorno de producción.
- **Exportación de la especificación**: Permite exportar la especificación OpenAPI en formato YAML o JSON para su uso posterior. Esta especificación se puede utilizar para generar código, automatizar pruebas o compartirla con otros desarrolladores.

Capítulo 3

Planificación y desarrollo del proyecto

3.1. Planificación

Como el cliente (el club de padel) se enfrenta a este reto de modernización tecnológica, es fácil que cambien sus prioridades durante el desarrollo del software ya que es un campo desconocido y muy diferente del contexto empresarial. Este puede querer añadir nuevas funcionalidades o cambiar de parecer sobre alguna ya implantada. Esta es una de las razones por la que la metodología a seguir sera una metodología ágil, SCRUM [12], que adaptara a las expectativas del cliente. Es una metodología óptica para la prevención y paliación de riesgos.
Otra de las principales razones por que se elige este modelo es que el desarrollo del producto se hará por partes que hará que el desarrollo sea incremental e iterativo en el que pueda participar el cliente mostrando sus opiniones y cambios. Se busca que el proyecto resultante sea de la mayor calidad posible, dejando en un segundo plano la calidad de los procesos y la documentación exhaustiva.
Para que haya una revisión permanente de los objetivos alcanzados los sprints serán de 15 días y estará fielmente relacionado con sus objetivos. Al final de cada interacción quincenal se examinara al detalle las metas alcanzadas, las fallidas y los aspectos a mejorar.
En SCRUM hay definidos tres roles

- Prouct Owner Es el que interpreta el modelo de negocio del cliente. Es el que decide que requisitos elaborar cuales no y de que forma se desarrollaran con el fin de maximizar los beneficios.
- Scrum Master es el líder del equipo y la voz frente al product Owner, esta implicado codo con codo con el equipo en fase de producción, planifica al detalle los quince días velando por el cumplimiento del programa eliminando impedimentos

- Equipo de desarrollo.

Para que los sprints consigan el mayor éxito posible se establecerán tres pautas a seguir en cada uno de ellos. Primero previa planificación, una reunión donde se define la meta del sprint la cual no será modificable.

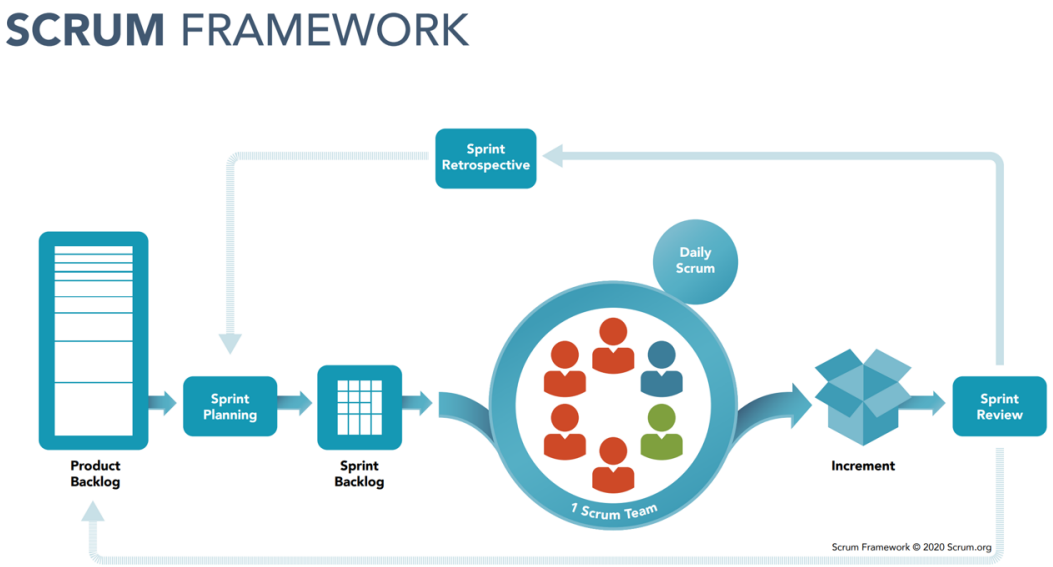

Figura 3.1: Scrum roles, Eventos y Artefactos [13]

3.1.1. Sprints

La metodología de desarrollo ágil de software ha demostrado ser muy efectiva en proyectos de software, permitiendo una rápida entrega de valor al cliente a través de iteraciones cortas y frecuentes llamadas sprints. En este caso, se propone una metodología de implementación de una aplicación de gestión de pistas de pádel, dividida en sprints de dos semanas con una revisión de código y objetivos al final de cada interacción. Cada sprint está enfocado en la implementación de un conjunto específico de funcionalidades, lo que permite una gestión más efectiva de los cambios y errores en el modelo de negocio y API. De esta manera, se logra una mayor adaptabilidad a los requisitos del cliente y se puede mantener una alta calidad del software entregado.

Sprint 1

- Definición de requisitos y análisis de funcionalidades
- Diseño de la arquitectura y la base de datos de la aplicación

- Creación de la estructura básica del proyecto (clases, paquetes, etc.)
- Modelado de la de la estructura de datos con API first

Sprint 2

- Implementación de la funcionalidad de registro y login de usuarios.
- Creación del API con las funciones principales.
- Implementación de la funcionalidad para la gestión de usuarios y su información personal.
- Realización de pruebas unitarias y revisión de código.

Sprint 3

- Implementación de la funcionalidad para la gestión de pistas de pádel y su reserva.
- Implementación de la funcionalidad para la gestión de partidos y torneos.
- Realización de pruebas unitarias y revisión de código.

Sprint 4

- Implementación de la funcionalidad para la generación de informes y estadísticas.
- Realización de pruebas unitarias y revisión de código.

Sprint 5

- Implementación de la funcionalidad para la gestión de perfiles de jugadores y su historial.
- Implementación de la funcionalidad de un bot de Telegram.
- Realización de pruebas unitarias y revisión de código.

Sprint 6

- Pruebas de despliegue.
- Implementación de la funcionalidad para la gestión de permisos y roles de usuario.
- Realización de pruebas unitarias y revisión de código.

Sprint 7

- Realización de pruebas de caja negra y pruebas de aceptación del bot con usuarios reales.
- Ajuste de la aplicación y corrección de errores.
- Revisión final del código y documentación del proyecto.

3.2. Metolologia

Domain-Driven Design (DDD)[14] es una metodología de desarrollo de software que se centra en comprender y modelar el dominio del problema para crear soluciones efectivas y de alta calidad. DDD promueve la colaboración estrecha entre expertos del dominio y desarrolladores, y se basa en la idea de que el diseño del software debe reflejar con precisión el lenguaje y las reglas del negocio.

El enfoque Api First[15] se refiere a diseñar y desarrollar la interfaz de programación de aplicaciones (API) antes de implementar la lógica subyacente. Con este enfoque, se pone un énfasis especial en la experiencia del usuario y en garantizar que la API sea intuitiva, eficiente y cumpla con los requisitos del negocio. Api First también implica obtener retroalimentación temprana de los usuarios y clientes para iterar y mejorar continuamente la API.

Ambas metodologías, DDD y Api First, comparten el objetivo de construir software que se ajuste a las necesidades del negocio y sea fácil de mantener y evolucionar. Al combinar DDD con el enfoque Api First, se pueden obtener beneficios adicionales al lograr una comprensión profunda del dominio y al asegurar una interfaz de API bien diseñada y validada.

A continuación, se explorarán las ventajas y desventajas de utilizar ambas metodologías en el desarrollo de la aplicación de gestión y reservas de una API.

Ventajas:

1. Comunicación y alineación: DDD facilita la comunicación y alineación entre los equipos de desarrollo y los expertos del dominio, lo que ayuda a comprender y modelar de manera precisa el negocio. El enfoque Api First asegura que la interfaz de la API esté diseñada y validada en base a las necesidades del negocio, lo que mejora aún más la comunicación y la alineación entre todas las partes involucradas.
2. Diseño centrado en el dominio y la interfaz: DDD permite modelar el dominio del problema de manera profunda, mientras que el enfoque Api First garantiza un diseño sólido de la interfaz de la API. Al combinar ambas metodologías, se puede lograr un diseño coherente y eficiente tanto a nivel de dominio como de interfaz.
3. Flexibilidad y adaptabilidad: DDD permite una arquitectura modular y flexible, lo que facilita la adaptación del software a medida que cambian los requisitos del negocio. El enfoque Api First permite obtener retroalimentación temprana de los usuarios y

clientes, lo que permite ajustar y mejorar la API de acuerdo con las necesidades en constante evolución.

Desventajas:

1. Complejidad adicional: La combinación de DDD y el enfoque Api First puede aumentar la complejidad inicial del proyecto, ya que ambas metodologías requieren un mayor esfuerzo de comprensión y diseño.

2. Posible sobrediseño: Si no se gestiona adecuadamente, la combinación de ambas metodologías puede llevar a un exceso de diseño y complejidad innecesaria en el software.

La combinación de Domain-Driven Design (DDD) y el enfoque Api First puede ser beneficiosa para el desarrollo de la aplicación de gestión y reservas de una API, ya que permite una comprensión profunda del dominio y un diseño sólido de la interfaz. Sin embargo, es importante gestionar adecuadamente la complejidad y evitar un sobrediseño para asegurar la eficiencia y mantenibilidad del software.

3.3. Gestión de Riesgos

En el proceso de desarrollo de una aplicación de gestión de padel pueden surgir diversos riesgos que afecten al tiempo, coste y calidad del proyecto. A continuación, se presentan algunos de los riesgos más comunes junto con su probabilidad, causa, consecuencia, impacto, estrategia de mitigación y plan de contingencia correspondiente.

Riesgo	**Cambios en los requisitos del cliente**
Probabilidad	Media
Consecuencia	Retraso en el proyecto o aumento de costos debido a la necesidad de rehacer el trabajo.
Causa	Falta de comunicación efectiva con el cliente o cambios en las necesidades del negocio.
Impacto	Moderado a alto
Estrategia	Establecer canales de comunicación claros con el cliente y realizar revisiones regulares de los requisitos del proyecto para garantizar que se satisfagan sus necesidades.
Plan de contingencia	Tener un proceso de cambio de control bien definido para evaluar los cambios en los requisitos, determinar su impacto en el proyecto y tomar decisiones informadas.

Cuadro 3.1: Riesgo de cambios en los requisitos del cliente

Riesgo	**Fallos en la infraestructura**
Probabilidad	Baja a media
Causa	Problemas técnicos o de mantenimiento en la infraestructura de TI utilizada para la aplicación.
Consecuencia	Retraso en el proyecto, pérdida de datos o interrupción del servicio.
Impacto	Moderado a alto
Estrategia	Establecer una estrategia de gestión de la infraestructura de TI sólida, que incluya la realización de pruebas regulares de la infraestructura y la implementación de medidas de seguridad adecuadas.
Plan de contingencia	Tener un plan de recuperación ante desastres para garantizar la continuidad del negocio en caso de fallo de la infraestructura.

Cuadro 3.2: Riesgo de fallos en la infraestructura

Riesgo	**Problemas de seguridad**
Probabilidad	Media
Causa	Falta de atención a la seguridad de la aplicación durante su desarrollo o implementación.
Consecuencia	Pérdida de datos, compromiso de la privacidad de los usuarios o daño a la reputación de la empresa.
Impacto	Alto
Estrategia	Establecer prácticas sólidas de seguridad durante todo el ciclo de vida del desarrollo de la aplicación, incluyendo la realización de pruebas de seguridad regulares y la implementación de medidas de seguridad adecuadas.
Plan de contingencia	Tener un plan de respuesta a incidentes de seguridad que permita detectar rápidamente los problemas de seguridad, aislarlos y solucionarlos de manera oportuna.

Cuadro 3.3: Riesgo de problemas de seguridad

Riesgo	**Fallos en la integración de sistemas**
Probabilidad	Media
Causa	Problemas técnicos o de compatibilidad entre los sistemas que deben integrarse.
Consecuencia	Retraso en el proyecto o interrupción del servicio.
Impacto	Moderado a alto
Estrategia	Establecer un plan de integración sólido y realizar pruebas rigurosas de integración antes de implementar la aplicación en producción.
Plan de contingencia	Tener un plan de contingencia para problemas de integración que permita detectar rápidamente los problemas, aislarlos y solucionarlos de manera oportuna.

Cuadro 3.4: Riesgo de fallos en la integración de sistemas

3.4. Presupuesto

El costo por hora en el desarrollo de software puede variar según varios factores, como la experiencia del ingeniero, la complejidad del proyecto, la ubicación geográfica y la demanda del mercado. En este caso, estableceremos un costo de 15 euros por hora como una estimación promedio para un ingeniero de software con experiencia moderada en el mercado español. [16]

Para el presupuesto inicial, se estima un total de 300 horas trabajadas en las diferentes tareas. Cada tarea se ha evaluado individualmente para determinar la cantidad de horas necesarias para completarla de manera efectiva.

Necesitaríamos conocer el tiempo de vida útil estimado del ordenador y su valor residual al final de dicho período. Si el ordenador tiene una vida útil de 5 años y un valor residual estimado de 200 euros al final de ese período.

Para calcular la depreciación anual, podemos utilizar el método de depreciación lineal, que distribuye el costo del ordenador de manera uniforme a lo largo de su vida útil.

Depreciación anual = (Costo del ordenador - Valor residual) / Vida útil

Supongamos que el costo total del ordenador es de 1000 euros y la suma de los periféricos (2 pantallas 120 euros cada una, teclado 98 euros y ratón 75 euros) suman un total de 1423 euros y utilizamos una tasa de depreciación anual del 20 % .

Depreciación anual = (1413 euros - 200 euros) / 5 = 242.6 euros por año

Para calcular la depreciación en las 300 horas utilizadas, podemos dividir la depreciación anual entre las horas de trabajo en un año y luego multiplicarlo por las horas utilizadas.

Depreciación por hora = Depreciación anual / Horas de trabajo en un año

Supongamos que hay 1800 horas de trabajo en un año.

Depreciación por hora = 242.6 euros / 1800 horas = 0.1348 euros/hora

Depreciación total del ordenador en las 300 horas utilizadas = Depreciación por hora * Horas utilizadas.

Depreciación total del ordenador y periféricos = 0.1348 euros/hora * 300 horas = 40.44 euros.

Elevar el presupuesto para contingencias es una práctica recomendada en la gestión de proyectos. Ayuda a garantizar la estabilidad financiera del proyecto y proporciona margen de maniobra para abordar situaciones imprevistas. Esta reserva adicional se sitúe generalmente en el 20 % del presupuesto total del proyecto.

Existen varios motivos por los que se recomienda elevar el presupuesto para contingencias:

Riesgos y cambios imprevistos: Durante la ejecución del proyecto, pueden surgir riesgos y eventos inesperados que requieran acciones correctivas o cambios en el alcance. La reserva adicional permite cubrir estos riesgos y mitigar su impacto financiero.

Cambios en los requisitos: A medida que se avanza en el proyecto, es posible que los requisitos iniciales cambien o evolucionen. Esto puede implicar ajustes en el diseño, desarrollo o implementación del software, lo cual puede requerir recursos adicionales y, por lo tanto, un aumento en el presupuesto.

Problemas técnicos: Es posible que surjan problemas técnicos durante la implementación del software, como conflictos de integración, fallas en la infraestructura o dificultades en la codificación. La reserva adicional ayuda a cubrir los costos adicionales asociados con la resolución de estos problemas.

Aumento de costos: Los costos de recursos, herramientas o servicios pueden fluctuar durante la duración del proyecto. La reserva adicional proporciona flexibilidad para hacer frente a posibles incrementos en los costos y evitar desviaciones significativas en el presupuesto.

3.4.1. Presupuesto inicial

Análisis y relevamiento de requerimientos: Se estima que tomará 40 horas para comprender a fondo los requisitos del proyecto y recopilar información relevante.

Diseño de la arquitectura del software: Se estima que tomará 20 horas para diseñar una arquitectura sólida y escalable que cumpla con los requisitos del proyecto.

Documentación de los requisitos funcionales y no funcionales: Se estima que tomará 30 horas para documentar de manera precisa y completa los requisitos del sistema.

Elaboración de diagramas de flujo y diagramas de clases: Se estima que tomará 25 horas para crear diagramas claros y comprensibles que representen la estructura y el flujo del software.

Creación de documentación de prueba y casos de uso: Se estima que tomará 15 horas para elaborar una documentación exhaustiva de las pruebas y los casos de uso del sistema.

Tiempo de implementación: Se estima que tomará 190 horas para implementar y desarrollar el software según los requisitos especificados.

El total de horas trabajadas en el presupuesto inicial es de 300 horas.

300 horas * 15 euros/hora = 4500 euros

Recurso	**Precio**
Herramientas	40,44€
Equipo de desarrollo	4500€
Total	4540,44€
Total con reserva de contingencias	5448,53€

Cuadro 3.5: Presupuesto inicial

3.4.2. Presupuesto final

Análisis y relevamiento de requerimientos: Se mantiene la estimación de 40 horas para asegurar una comprensión precisa de los requisitos actualizados.

Diseño de la arquitectura del software: Se mantiene la estimación de 20 horas para garantizar la coherencia y la calidad del diseño.

Documentación de los requisitos funcionales y no funcionales: Se mantiene la estimación de 30 horas para reflejar los cambios y las mejoras en los requisitos.

Elaboración de diagramas de flujo y diagramas de clases: Se mantiene la estimación de 25 horas para asegurar la claridad en la representación visual del software actualizado.

Creación de documentación de prueba y casos de uso: Se mantiene la estimación de 15 horas para documentar las pruebas y casos de uso actualizados.

Tiempo de implementación: Se estima que tomará 210 horas, reflejando las tareas adicionales y los ajustes requeridos durante la implementación.

Revisiones y modificaciones de la documentación: Se estima que tomará 10 horas para realizar las revisiones y modificaciones necesarias en la documentación del proyecto.

El total de horas trabajadas en el presupuesto final es de 330 horas.

Sumando el costo de las horas de implementación del presupuesto final (330 horas a 15 euros por hora), el cálculo del presupuesto quedaría de la siguiente manera:

Presupuesto Final (incluyendo horas de implementación y revisión de documentación): 330 horas * 15 euros/hora = 4950 euros

Recurso	**Precio**
Herramientas	40,44€
Equipo de desarrollo	4950€
Total	4990,44 €
Total con reserva de contingencias	5988,53€

Cuadro 3.6: Presupuesto final

Capítulo 4

Análisis y diseño

4.1. Análisis

A continuación, se realiza un análisis detallado de los diferentes actores involucrados en el proceso de gestión de reservas de pádel. Se identifican los roles de los administradores, los usuarios y los encargados de las instalaciones. Se define la funcionalidad específica y los privilegios asociados a cada uno de estos roles, para asegurar una experiencia de usuario personalizada y segura.

Otro aspecto clave del análisis es la identificación de los datos necesarios para el correcto funcionamiento del sistema. Se definen las entidades relevantes, como las reservas, los recursos, y los usuarios, y se analizan las relaciones y restricciones entre ellas.

4.1.1. Actores

A continuación se presenta la tabla que describe los códigos, usuarios y descripciones:

Código	Usuario	Descripción
U-01	Usuario registrado	Tienen acceso a todas las funcionalidades, incluyendo la visualización de información sobre las pistas, horarios y precios, la realización de reservas y la participación en partidos.
U-02	Usuario no registrado	Persona que aún no se ha registrado en la aplicación. Solo puede acceder a promociones y al registro de usuario. No puede acceder a información general sobre las pistas, horarios y precios o realizar reservas.
U-03	Usuario de Telegram	Usuarios que utilizan la aplicación a través de la plataforma de Telegram. No hace falta que estén registrados en la aplicación, pueden realizar reservas, publicar y unirse a partidos.
U-04	Administrador	Es una persona encargada de gestionar y supervisar las operaciones del club de pádel. Tiene acceso a funciones administrativas avanzadas, como la gestión de pistas, horarios, precios, confirmación de reservas y comunicación con los usuarios. El administrador del club se asegura de que todas las reservas se realicen correctamente y resuelve cualquier problema relacionado con las pistas o el proceso de reserva.
U-05	Entrenador	Son profesionales encargados de impartir clases de pádel y asesorar a los usuarios en su desarrollo en el deporte. Pueden tener acceso a información específica sobre las reservas y horarios de las pistas para organizar sus sesiones de entrenamiento de manera efectiva.
U-06	Socio	Son usuarios registrados en el club de pádel y tienen acceso a beneficios adicionales, como descuentos especiales, reservas preferenciales o participación en eventos exclusivos. Los socios pueden realizar reservas de pistas y aprovechar las instalaciones del club de acuerdo con las políticas y regulaciones establecidas.

Cuadro 4.1: Usuarios participantes en el sistema.

4.1.2. Requisitos Funcionales

Para mejorar la estructuración y la visión de los objetivos se han identificado los requisitos funcionales específicos para cada componente del sistema: la API y el bot de Telegram. Ambos bloques de requisitos están basados en las funcionalidades requeridas para cada componente. A continuación, se presentan los requisitos funcionales para la API de reservas y gestión de un club deportivo de pádel.

Código	Requisito	Descripción
RF-001	Registro de usuarios	La aplicación debe permitir a los usuarios registrarse en el sistema proporcionando información personal, como nombre, correo electrónico y contraseña.
RF-002	Inicio de sesión	Los usuarios registrados deben poder iniciar sesión en la aplicación con su correo electrónico y contraseña.
RF-003	Visualización de perfil de usuario	El usuario registrado podrá consultar sus datos.
RF-003	Modificación de perfil de usuario	El usuario registrado podrá modificar sus datos.
RF-003	Pre-registro de socio	El usuario registrado podrá solicitar su sociedad a partir de un formulario.
RF-003	Baja de socio	El socio solicita su baja a través del sistema, también, el administrador tiene la capacidad de dar de baja a un socio
RF-003	Alta de socio	El administrador tiene la capacidad de dar de alta a un socio que previamente lo haya solicitado.
RF-003	Visualización de pistas de padel disponibles	La aplicación debe mostrar una lista de las pistas de padel disponibles para la reserva, número y horarios disponibles.
RF-003	Visualización de entrenamientos disponibles	Los usuarios podrán consultar los entrenamientos disponibles para un determinado día, nivel y pista.
RF-004	Reserva de pistas de padel	Los usuarios deben poder reservar una pista de padel específica para un horario determinado y realizar el pago correspondiente.
RF-005	Cancelación de reservas	Los usuarios deben poder cancelar una reserva de pista de padel previamente realizada.
RF-006	Visualización del historial de reservas	Los usuarios deben poder ver un registro de todas las reservas de pistas de padel realizadas anteriormente.
RF-009	Administración de horarios de las pistas	El administrador de la aplicación debe poder configurar los horarios de las pistas de padel disponibles para la reserva.
RF-009	Dar de alta festivos o fechas significativas	El administrador de la aplicación puede seleccionar días en el calendario para crear festivos locales, eventos etc...
RF-010	Administración de pistas de padel	El administrador de la aplicación debe poder agregar, modificar o eliminar pistas de padel.

Cuadro 4.2: Requisitos de la API de gestión y de reserva de pistas de padel

Algunos de los requisitos de funcionales del bot de Telegran serán iguales o tendrán mucha

similitud con los de la API ya que estarán basados en la misma lógica de negocio.

Código	Requisito	Descripción
RF-T01	Obtener información sobre las pistas disponibles	Los usuarios pueden obtener información en tiempo real sobre las pistas de padel disponibles para reservas. Se muestra una lista de las pistas para los próximos cinco días, con horarios y disponibilidad. Los usuarios pueden seleccionar una pista y proceder con la reserva, ya sea de forma privada o pública.
RF-T02	Acceder a las reservas existentes	Los usuarios pueden ver una lista de las reservas que han realizado. Se muestran detalles como la fecha, la hora y la pista reservada. También se puede mostrar el estado de la reserva y permitir modificaciones o cancelaciones.
RF-T03	Acceder a información general	Los usuarios pueden obtener información general sobre el sistema de gestión de reservas de pádel. Aquí se proporcionan detalles sobre el funcionamiento del sistema, las políticas de reservas, las tarifas y las reglas de uso de las instalaciones.
RF-T04	Obtener información sobre los entrenamientos	Los usuarios pueden acceder a información sobre los entrenamientos de pádel ofrecidos en el centro. Se proporciona información sobre los tipos de entrenamientos disponibles, los horarios, los niveles de habilidad requeridos y los instructores asignados.
RF-T04	Acceder a información sobre las promociones	Los usuarios pueden obtener información sobre las promociones vigentes en el centro de pádel. Se muestran detalles sobre los descuentos, las ofertas especiales y cualquier otra promoción disponible. Esto permite a los usuarios aprovechar beneficios adicionales al realizar sus reservas.

Cuadro 4.3: Requisitos funcionales del bot de Telegram

4.2. Requisitos no funcionales

Autorizacion y autorizacion ver para poner en modo tabla En el desarrollo de aplicaciones modernas, los requisitos no funcionales juegan un papel crucial para garantizar el éxito y la eficiencia del sistema. Estos requisitos se centran en aspectos clave como el rendimiento, la seguridad, la escalabilidad, la fiabilidad y la documentación, que son fundamentales para ofrecer una experiencia de usuario óptima y mantener la calidad del sistema a lo largo del tiempo.

En el contexto de una aplicación REST desarrollada en Spring Boot, se han identificado

una serie de requisitos no funcionales clave que deben ser considerados para asegurar el cumplimiento de los estándares de rendimiento, al abordar estos requisitos de manera adecuada, se crea una base sólida para el éxito y la satisfacción de los usuarios.

Rendimiento:

- La aplicación debe tener un tiempo de respuesta máximo de X segundos para las solicitudes.
- La aplicación debe ser capaz de escalar horizontalmente para manejar un aumento en la carga de trabajo sin afectar el rendimiento.
- Se debe implementar una estrategia de caché para minimizar el tiempo de respuesta y reducir la carga en los recursos del servidor.
- Las consultas de base de datos deben ser optimizadas para garantizar un acceso eficiente a los datos y minimizar la carga en el sistema.

Seguridad:

- Se debe implementar un mecanismo seguro de autenticación y autorización para proteger los end-points y permitir el acceso solo a usuarios autorizados.
- La aplicación debe estar protegida contra ataques comunes, como inyecciones SQL, XSS y CSRF.
- Todos los datos sensibles que se transmiten entre la aplicación y los clientes deben estar encriptados.

Fiabilidad:

- La aplicación debe ser tolerante a fallos y contar con mecanismos de recuperación para minimizar el impacto de errores y fallas.
- Debe haber herramientas de monitorización y registro configuradas para detectar y diagnosticar problemas de forma proactiva.

Documentación:

- Se debe proporcionar una documentación completa y clara de la API, que incluya descripciones detalladas de los end-points, los parámetros y las respuestas esperadas.

4.3. Casos de uso

En este apartado, abordaremos la especificación de los casos de uso y sus diagramas. Aunque todos los actores forman parte del sistema global, que es la aplicación, presentaremos los diagramas de casos de uso de forma separada por actor para facilitar la comprensión y el seguimiento de las funcionalidades ofrecidas por la aplicación, brindando una visión más clara de cómo interactúan los diferentes actores con el sistema en función de sus roles y responsabilidades.

Los requisitos de usuario se agruparán en función de los roles del usuario y las funcionalidades proporcionadas por la API y el bot de Telegram. Esto nos permitirá identificar claramente qué acciones están disponibles para cada tipo de usuario y cómo interactúan con la aplicación. Para cada actor, se presentarán los casos de uso correspondientes a su rol y las acciones que puede realizar en el sistema. Se proporcionarán descripciones detalladas de cada caso de uso, junto con sus diagramas correspondientes, que ilustrarán las interacciones entre el actor y el sistema.

4.3.1. Diagrama de caso de uso

A continuación se muestra el diagrama de casos de uso, el cual ha sido realizado mediante la información obtenida en los pasos previos. En el diagrama se observan los distintos casos de uso que pueden realizar cada uno de los actores.

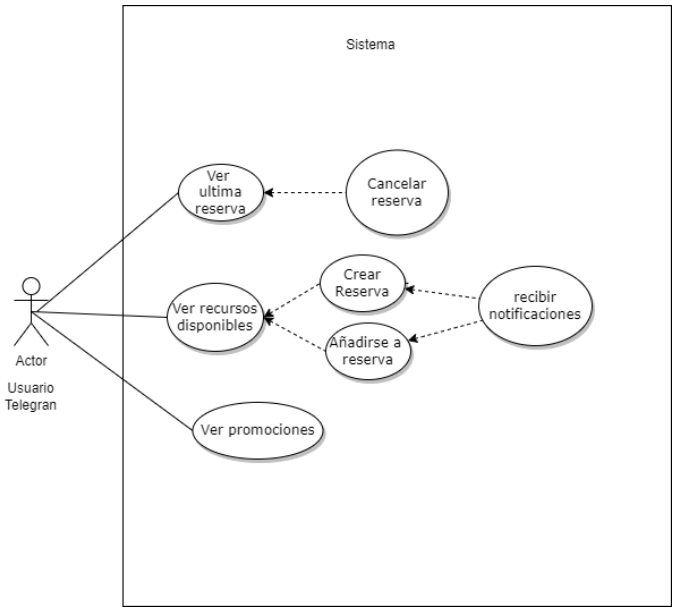

Figura 4.1: Diagrama caso de uso usuario Telegram

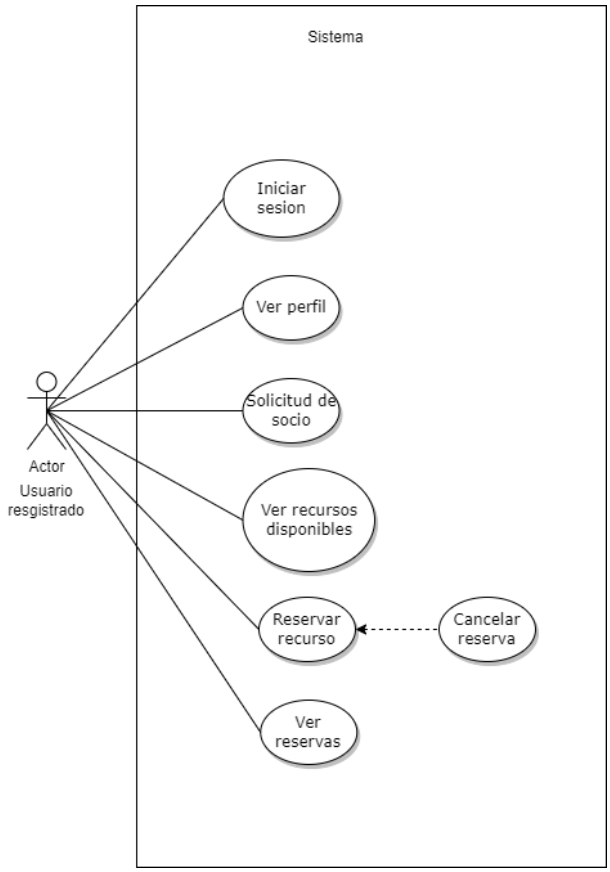

Figura 4.2: Diagrama caso de uso usuario Registrado

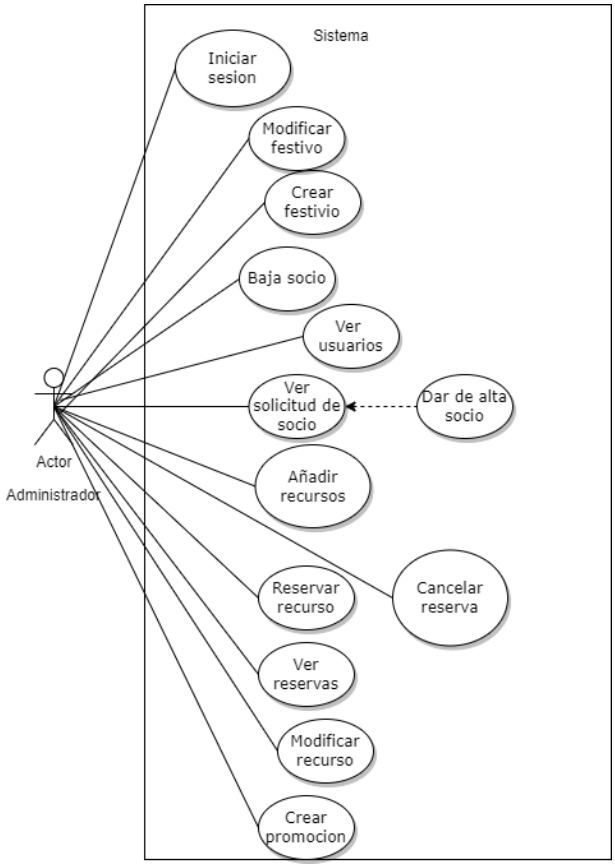

Figura 4.3: Diagrama caso de uso administrador

4.3.2. Descripción de casos de uso

La descripción de los casos de uso viene definida en la documentación de los endpoint explicando el formato de entrada de salida así como ejemplos de uso. Si que hay que explicar los casos de uso del bot de Telegram

UC-001	**Obtener información sobre los recursos disponibles**
Autor	Manuel Comesaña
Descripción	Los usuarios pueden obtener información en tiempo real sobre las pistas de pádel disponibles para reservas. Se muestra una lista de las pistas para los próximos cinco días, con horarios y disponibilidad. Los usuarios pueden ver las reservas públicas que necesitan algún jugador.
Precondición	iniciar el bot
Secuencia normal	1. El usuario accede a la sección de "Pistas disponibles". 2. El usuario selecciona un día de los próximos cinco días. 4. El sistema muestra los detalles de la pista disponibles en ese día. Nivel, participantes etc.
Postcondición	
Excepciones	2.a. No hay pistas disponibles para los próximos cinco días. 4.a. El usuario cancela la selección de la pista y vuelve a la lista de pistas disponibles.

Cuadro 4.4: Caso de Uso: Obtener información sobre las pistas disponibles

UC-001	**Realizar reserva de recurso libre**
Autor	Manuel Comesaña
Descripción	Los usuarios pueden seleccionar una pista que se encuentre vacía y proceder con la reserva, ya sea de forma privada o pública.
Precondición	El el usuario ha listado las pistas.
Secuencia normal	1. El usuario selecciona una pista disponible sin jugadores anotados. 2. El sistema muestra los detalles de la pista seleccionada y ofrece la opción de reservarla de forma privada o pública. 3. El usuario selecciona el tipo de reserva (privada o pública). 4. El usuario selecciona el tipo de partido (masculino, femenino o mixto). 5. El usuario establece un nivel para el partido y confirma la reserva. 6. El sistema registra la reserva y muestra una confirmación al usuario.
Postcondición	Se realiza una reserva de la pista seleccionada según la preferencia del usuario.
Excepciones	4.a. El usuario cancela la selección de la pista y vuelve a la lista de pistas disponibles. 6.a. Se produce un error al realizar la reserva.

Cuadro 4.5: Caso de Uso: Realizar reserva de recurso libre

UC-002	**Unirse a un partido**
Autor	Manuel Comesaña
Descripción	Los usuarios pueden seleccionar una pista que se encuentre ya con jugadores tipo de partido y con un nivel determinado .
Precondición	El el usuario ha listado las pistas.
Secuencia normal	1. El usuario selecciona una pista disponible sin jugadores anotados. 2. El sistema muestra los detalles de la pista seleccionada y ofrece la opción de reservarla de forma privada o pública. 3. El usuario selecciona el tipo de reserva (privada o pública). 4. El usuario selecciona el tipo de partido (masculino, femenino o mixto). 5. El usuario establece un nivel para el partido y confirma la reserva. 6. El sistema registra la reserva y muestra una confirmación al usuario.
Postcondición	Se realiza una reserva de la pista seleccionada según la preferencia del usuario.
Excepciones	4.a. El usuario cancela la selección de la pista y vuelve a la lista de pistas disponibles. 6.a. Se produce un error al realizar la reserva.

Cuadro 4.6: Caso de Uso: Unirse a un partido

UC-003	**Acceder a las reservas existentes**
Autor	Manuel Comesaña
Descripción	Los usuarios pueden ver una lista de las reservas que han realizado. Se muestran detalles como la fecha, la hora y la pista reservada. También se puede mostrar el estado de la reserva y permitir modificaciones o cancelaciones.
Precondición	
Secuencia normal	1. El usuario accede a la sección de "Mis Reservas". 2. El sistema muestra una lista de las reservas realizadas por el usuario. 3. El usuario puede ver los detalles de cada reserva, incluyendo la fecha, la hora, participantes y la pista reservada. 4. El usuario puede cancelar una reserva existente.
Postcondición	Se realizan las modificaciones o cancelaciones correspondientes a las reservas existentes según la acción realizada por el usuario. Y se notifica a los demás usuarios
Excepciones	2.a. El usuario no tiene reservas realizadas. 4.a. El usuario cancela la modificación de la reserva y vuelve a la lista de reservas existentes. 5.a. El usuario cancela la cancelación.Selecciona otra opción en el menú.

Cuadro 4.7: Caso de Uso: Acceder a las reservas existentes

UC-004	**Acceder a información sobre las promociones**
Autor	Manuel Comesaña
Descripción	Los usuarios pueden obtener información sobre las promociones vigentes en el centro de pádel. Se muestran detalles sobre los descuentos, las ofertas especiales y cualquier otra promoción disponible. Esto permite a los usuarios aprovechar beneficios adicionales al realizar sus reservas.
Precondición	Ninguna
Secuencia normal	1. El usuario accede a la sección de "Promociones". 2. El sistema muestra una lista de las promociones vigentes en el centro de pádel. 3. El usuario puede ver los detalles de cada promoción.
Postcondición	Ninguna
Excepciones	2.a. No hay promociones vigentes en el momento.

Cuadro 4.8: Caso de Uso: Acceder a información sobre las promociones

4.4. Diseño

El diseño de un proyecto con varios módulos Maven interconectados se ha escogido con el objetivo de lograr un sistema modular, escalable y de fácil mantenimiento. La utilización de este enfoque arquitectónico brinda numerosos beneficios en términos de estructura, segregación de responsabilidades y flexibilidad.

El modelo de arquitectura de capas se ha seleccionado por su capacidad para organizar el sistema en capas lógicas, cada una con responsabilidades específicas. Esta segregación de responsabilidades permite un desarrollo más ordenado y facilita futuras modificaciones o mejoras en el sistema. Además, este modelo proporciona una clara división entre la capa de presentación, la capa de lógica de negocio y la capa de acceso a datos, lo cual simplifica la comprensión y el mantenimiento del código.

La elección del modelo de microservicios se ha basado en la necesidad de contar con una arquitectura altamente escalable y modular. Al descomponer el sistema en microservicios independientes, cada uno con su propia lógica de negocio y comunicándose a través de interfaces definidas, se logra una mayor flexibilidad y se facilita el despliegue y la escalabilidad individual de cada componente. Esto es especialmente útil en proyectos grandes y complejos, donde diferentes equipos pueden trabajar de manera independiente en cada microservicio.

Además, la adopción del patrón de inyección de dependencias ha sido una elección estratégica para fomentar la modularidad y la reutilización de componentes. Al utilizar este patrón, se establece una clara separación entre las interfaces y las implementaciones, lo que facilita la sustitución de componentes y la realización de pruebas unitarias de manera aislada. Esto contribuye a un código más mantenible, extensible y fácilmente testeable.

Se ha escogido este diseño basándose en los beneficios que ofrece en términos de modularidad, escalabilidad y mantenibilidad del sistema. El modelo de arquitectura de capas, el enfoque de microservicios y el uso de inyección de dependencias brindan una estructura

sólida y flexible que se adapta a los requisitos del proyecto. Estos modelos de diseño permiten una mayor organización del código, una gestión más eficiente de cambios y mejoras, y una arquitectura escalable y modular que facilita el desarrollo y el mantenimiento a largo plazo.

4.4.1. API

Como ya se comento anteriormente se opta por la metodología API First en conjunto con OpenAPI para diseñar y desarrollar API. API First es un enfoque en el que la prioridad es la fase de diseño y especificación de la API antes de cualquier implementación o codificación. Permíteme explicarte por qué esta metodología es beneficiosa y cómo se alinea con OpenAPI.

En esta fase vamos a definir previamente las estructuras de datos, los formatos de entrada/respuesta y los end points con el objetivo de garantizar que se cumplan todos los requisitos funcionales.

Se definirá un endpoit especifico para la gestión del club que se llamara "backoffice". Esta ruta solo sera accesible por los usuarios que tengan rol administrador y se encuentren autenticados y autorizados. Los end points se agruparan mediante etiquetas o tags acorde a su funcionalidad. Para ordenar las operaciones y filtrar las funciones se generan las siguientes etiquetas.

- **Booking**: Comprende todas las funciones relacionadas con las reservas de los recursos.
- **Resources**: Engloba las operaciones de gestión de los recursos. Solo accesible desde backoffice.
- **User**: Incluye las operaciones relativas a los usuarios o a la gestión de los mismos.
- **Festive**: Solo accesible desde backoffice, para la gestión del calendario. Se utiliza para el manejo de días festivos y eventos especiales.
- **BackOffice**: Es una ruta especifica para los administradores. Engloba las operaciones de gestión.

Muchas operaciones estarán categorizadas con varias etiquetas, como por ejemplo el los endpoints /booking y backoffice/booking ambas operaciones POST crean una reserva, pero la de backoffice esta destinada a los administradores que permitirá añadir mas campos a la entrada o incluso crear varias reservas en una sola petición.

A continuación se detallan los end-points mas relevantes de la API (se recomienda verlos con mas detalle como se indica en el apéndice A: Manual de uso para front-end):

Endpoint:	/booking
Método:	POST
Etiquetas:	Booking
Nombre de la operación:	createBooking
Parámetros de entrada:	inputBooking
Respuestas	
Código	**Descripción**
200	Éxito
401	No autorizado
405	Excepción de validación

Cuadro 4.9: Endpoint: /booking

Endpoint:	/backoffice/booking
Método:	POST
Etiquetas:	Booking, Backoffice
Nombre de la operación:	backOfficeAddBooking
Parámetros de entrada:	inputBooking
Respuestas	
Código	**Descripción**
200	Éxito
405	Excepción de validación

Cuadro 4.10: Endpoint: /backoffice/booking

Endpoint:	/booking/findByDate
Método:	GET
Etiquetas:	Booking
Nombre de la operación:	findBookingsByDates
Parámetros de entrada:	startDate (query), endDate (query)
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	Valor de fecha inválido
401	Error de autenticación
405	Excepción de validación

Cuadro 4.11: Endpoint: /booking/findByDate

Endpoint:	/booking/findByType
Método:	GET
Etiquetas:	Booking
Nombre de la operación:	findBookingsByType
Parámetros de entrada:	type (query)
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	Tipo de reserva inválido
401	Error de autenticación
405	Excepción de validación

Cuadro 4.12: Endpoint: /booking/findByType

Endpoint:	/booking/bookingId
Método:	GET
Etiquetas:	Reserva
Nombre de la operación:	getBookingById
Parámetros de entrada:	bookingId (ruta, obligatorio)
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	ID no válido
401	Operación no autorizada
404	Reserva no encontrada

Cuadro 4.13: Endpoint GET: /booking/bookingId

Endpoint:	/booking/bookingId
Método:	PUT
Etiquetas:	Reserva
Nombre de la operación:	updateBooking
Parámetros de entrada:	bookingId (ruta, obligatorio)
Cuerpo de la solicitud:	inputBooking
Respuestas	
Código	**Descripción**
200	OK
400	ID de reserva no válido
401	No autorizado
404	Reserva no encontrada
405	Excepción de validación

Cuadro 4.14: Endpoint PUT: /booking/bookingId

Endpoint:	/booking/bookingId
Método:	DELETE
Etiquetas:	Reserva
Nombre de la operación:	deleteBooking
Parámetros de entrada:	bookingId (ruta, obligatorio)
Respuestas	
Código	**Descripción**
204	Reserva eliminada exitosamente
400	ID de reserva no válido
401	No autorizado
404	Reserva no encontrada

Cuadro 4.15: Endpoint DELETE: /booking/bookingId

Endpoint:	/booking/bookingId
Método:	GET
Etiquetas:	Booking
Nombre de la operación:	getBookingById
Parámetros de entrada:	bookingId (path)
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	ID no válido
401	Error de autenticación
404	Reserva no encontrada

Cuadro 4.16: Endpoint: /booking/bookingId

Endpoint:	/booking/bookingId
Método:	PUT
Etiquetas:	Booking
Nombre de la operación:	updateBooking
Parámetros de entrada:	bookingId (path)
Cuerpo de la solicitud:	Información actualizada de la reserva
Respuestas	
Código	**Descripción**
200	OK
400	ID de reserva no válido
401	No autorizado
404	Reserva no encontrada
405	Excepción de validación

Cuadro 4.17: Endpoint: /booking/bookingId (PUT)

Endpoint:	/booking/bookingId
Método:	DELETE
Etiquetas:	Booking
Nombre de la operación:	deleteBooking
Parámetros de entrada:	bookingId (path)
Respuestas	
Código	**Descripción**
204	Reserva eliminada con éxito
400	ID de reserva no válido
401	No autorizado
404	Reserva no encontrada

Cuadro 4.18: Endpoint: /booking/bookingId (DELETE)

Endpoint:	/resource/id
Método:	GET
Etiquetas:	Resource
Nombre de la operación:	getResource
Parámetros de entrada:	id (path)
Respuestas	
Código	**Descripción**
200	Operación exitosa
404	Recurso no encontrado

Cuadro 4.19: Endpoint: /resource/{id}

Endpoint:	/backoffice/resource
Método:	PUT
Etiquetas:	Resource, Backoffice
Nombre de la operación:	createResource
Cuerpo de la solicitud:	Información actualizada del recurso
Respuestas	
Código	**Descripción**
200	Operación exitosa
405	Entrada no válida

Cuadro 4.20: Endpoint: /backoffice/resource (PUT)

Endpoint:	/backoffice/resource
Método:	POST
Etiquetas:	Resource, Backoffice
Nombre de la operación:	createResourceDTO
Cuerpo de la solicitud:	Información del nuevo recursoDTO
Respuestas	
Código	**Descripción**
200	Operación exitosa
405	Entrada no válida

Cuadro 4.21: Endpoint: /backoffice/resource (POST)

Endpoint:	/resource/findByDate
Método:	GET
Etiquetas:	Resource, Booking
Nombre de la operación:	getResourceDTOsByDate
Parámetros	
Nombre	**Descripción**
start_date	Fecha de inicio del período de disponibilidad de resourceDTOs (en formato ISO 8601)
end_date	Fecha de finalización del período de disponibilidad de resourceDTOs (en formato ISO 8601)
Respuestas	
Código	**Descripción**
200	Operación exitosa

Cuadro 4.22: Endpoint: /resource/findByDate (GET)

Endpoint:	/user
Método:	POST
Etiquetas:	User
Nombre de la operación:	createUser
Cuerpo de la solicitud:	Objeto de usuario creado
Respuestas	
Código	**Descripción**
200	Operación exitosa

Cuadro 4.23: Endpoint: /user (POST)

Endpoint:	/user/login
Método:	GET
Etiquetas:	User
Nombre de la operación:	loginUser
Parámetros	
Nombre	**Descripción**
email	El nombre de usuario para iniciar sesión
password	La contraseña para iniciar sesión en texto claro
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	Nombre de usuario/contraseña no válido

Cuadro 4.24: Endpoint: /user/login (GET)

Endpoint:	/user/logout
Método:	GET
Etiquetas:	User
Nombre de la operación:	logoutUser
Parámetros:	Ninguno
Respuestas	
Código	**Descripción**
default	Operación exitosa

Cuadro 4.25: Endpoint: /user/logout (GET)

Endpoint:	/user/username
Método:	GET, PUT, DELETE
Etiquetas:	User
Nombre de la operación:	getUserByName (GET), updateUser (PUT), deleteUser (DELETE)
Parámetros	
Nombre	**Descripción**
username	El nombre que se debe recuperar. Utilice user1 para realizar pruebas.
Respuestas	
Código	**Descripción**
200	Operación exitosa (GET)
default	Operación exitosa (PUT, DELETE)
400	Nombre de usuario no válido
404	Usuario no encontrado

Cuadro 4.26: Endpoint: /user/username (GET, PUT, DELETE)

Endpoint:	/user/username/bookings
Método:	GET
Etiquetas:	User, Booking
Nombre de la operación:	getBookingsUser
Parámetros	
Nombre	**Descripción**
username	El nombre que se debe recuperar. Utilice user1 para realizar pruebas.
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	Nombre de usuario no válido
404	Usuario no encontrado

Cuadro 4.27: Endpoint: /user/username/bookings (GET)

Endpoint:	/festive/idfestive
Método:	GET
Etiquetas:	Festive
Nombre de la operación:	getFestivebyId
Seguridad:	bearerAuth
Parámetros	
Nombre	**Descripción**
idfestive	ID del evento festivo
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	ID de evento festivo no válido
404	Evento festivo no encontrado

Cuadro 4.28: Endpoint: /festive/idfestive (GET)

Endpoint:	/festive/date
Método:	GET
Etiquetas:	Festive
Nombre de la operación:	getFestivebyDate
Parámetros	
Nombre	**Descripción**
date	Fecha del evento festivo
Respuestas	
Código	**Descripción**
200	Operación exitosa
400	ID de evento festivo no válido
404	Evento festivo no encontrado

Cuadro 4.29: Endpoint: /festive/date (GET)

Endpoint:	/backoffice/festive/idfestive
Método:	PUT, DELETE
Etiquetas:	Festive, Backoffice
Nombre de la operación:	updateFestive (PUT), deleteFestive (DELETE)
Parámetros	
Nombre	**Descripción**
idfestive	ID del evento festivo a eliminar
Cuerpo de la solicitud	
Descripción	Actualizar un evento festivo existente en el sistema
Contenido	application/json
Esquema	FestiveDTO
Respuestas	
Código	**Descripción**
default	Operación exitosa

Cuadro 4.30: Endpoint: /backoffice/festive/idfestive (PUT, DELETE)

Cuadro 4.31: Esquema: PlayerPosition

Campo	**Descripción**
`type`	string
`description`	Posición favorita del jugador

Cuadro 4.32: Esquema: ResourceDTOType

Campo	**Descripción**
`type`	string
`description`	Tipo de reserva
`enum`	PADEL, ARENA, GYM, ECOWASH

Cuadro 4.33: Esquema: BookingType

Campo	**Descripción**
`type`	string
`description`	Tipo de reserva
`enum`	TOURNAMENT, TRAINING, GAME, MATCH-TELEGRAM, LEAGUE

Cuadro 4.34: Esquema: User

Campo	Descripción
id	integer (int64)
username	string
firstName	string
lastName	string
email	string (email)
password	string (password)
phone	string
userPosition	Referencia a PlayerPosition
level	integer
description	string
startTime	string (date-time)
endTime	string (date-time)
sport	string
role	string
memberClub	Referencia a MemberClub

Cuadro 4.35: Esquema: ResourceDTO

Campo	Descripción
id	integer (int64)
number	integer (int32)
name	string
resourceDTOType	Referencia a ResourceDTOType
timeSlot	integer
startTimeSlot	string (time)
endTimeSlot	string (time)
basePrice	number (double)
daysInAdvance	integer (int32)

Cuadro 4.36: Esquema: InputResourceDTO

Campo	Descripción
number	integer (int32)
name	string
resourceDTOActivity	string
timeSlot	integer
startTimeSlot	string (time)
endTimeSlot	string (time)
basePrice	number (double)
daysInAdvance	integer (int32)

Cuadro 4.37: Esquema: User

Campo	Descripción
id	integer (int64)
username	string
firstName	string
lastName	string
email	string (email)
password	string (password)
phone	string
userPosition	Referencia a PlayerPosition
level	integer
description	string
startTime	string (date-time)
endTime	string (date-time)
sport	string
role	string
memberClub	Referencia a MemberClub

Cuadro 4.38: Esquema: MemberClub

Campo	Descripción
startDate	string (date-time)
endTDate	string (date-time)
dni	string
onHold	boolean

Cuadro 4.39: Esquema: Coach

Campo	Descripción
startDate	string (date-time)
endTDate	string (date-time)
dni	string
onHold	boolean

Cuadro 4.40: Esquema: UserBooking

Campo	Descripción
id	integer (int64)
username	string
firstName	string
lastName	string
email	string (email)
phone	string
userPosition	Referencia a PlayerPosition

Cuadro 4.41: Esquema: BookingDTO

Campo	Descripción
idBooking	integer (int64)
resourceDTO	Referencia a ResourceDTO
startDate	string (date-time)
endDate	string (date-time)
ownerUser	Referencia a UserBooking
price	number (double)
guests	Referencia a UserBooking
bookingType	Referencia a BookingType
bookingVisibility	Referencia a BookingVisibility

Cuadro 4.42: Esquema: InputBooking

Campo	Descripción
resource	integer (int64)
startDate	string (date-time)
endDate	string (date-time)
user	integer (int64)
bookingType	Referencia a BookingType
guests	array de string
bookingVisibility	Referencia a BookingVisibility
level	integer

Cuadro 4.43: Esquema: FestiveDTO

Campo	Descripción
festiveId	integer (int64)
date	string (date-time)
description	string

Cuadro 4.44: Esquema: AuthenticationResponse

Campo	Descripción
accessToken	string
refreshToken	string

4.4.2. Arquitectura API

En esta fase de diseño, se definirán las estructuras de paquetes en este caso para hacer que sean lo mas independientes posibles se opta por una estructura de módulos Maven, para así permitir añadir librerías de terceros u otros módulos presentes en el proyecto. Se detallara

la estructura de cada modulo a groso modo para comprender como se destruyen los paquetes en cada componente.

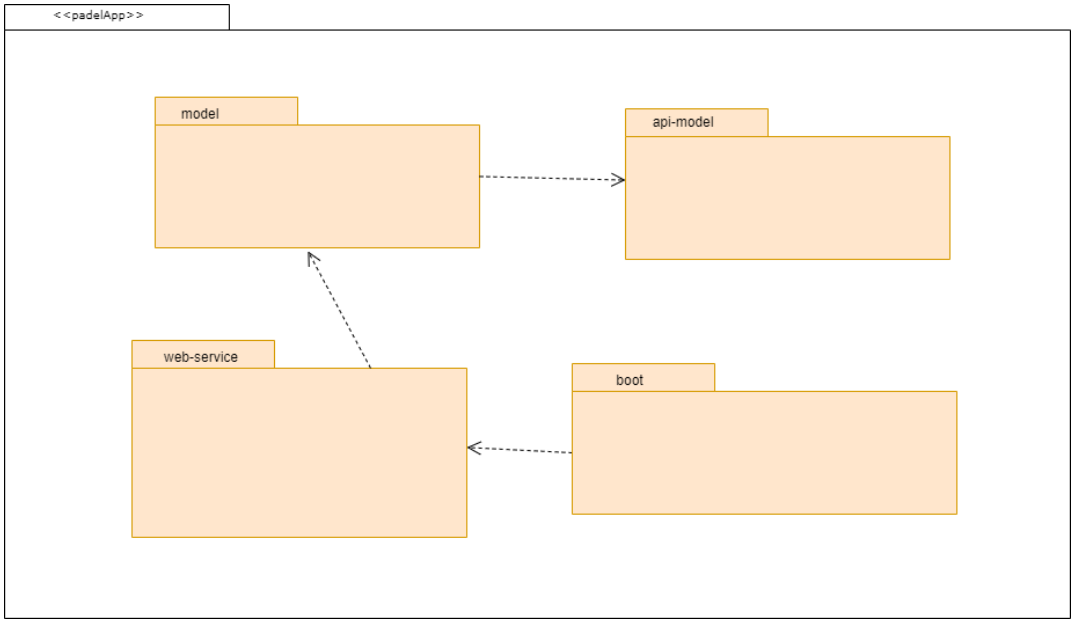

Figura 4.4: Diagrama de estructura de módulos

- **Módulo API Model:** En esta capa se encuentran las interfaces y la definición de la API en modo de API first. Aquí se definen los contratos y las estructuras de datos que serán utilizados por el resto del sistema. Las interfaces representan los puntos de entrada y salida de la API, especificando los métodos y los datos necesarios para interactuar con el sistema. Esta capa promueve una separación clara entre la interfaz de la API y su implementación, facilitando la modularidad y la comunicación entre componentes.

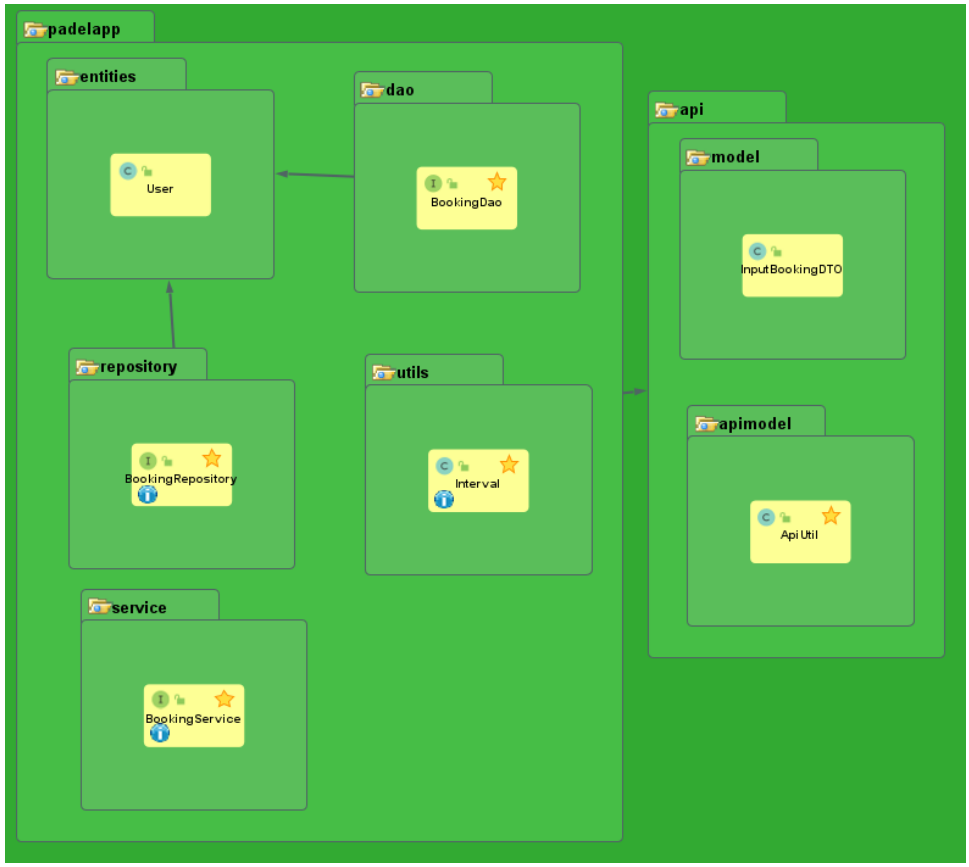

Figura 4.5: Diagrama de estructura de módulos

- **Módulo Model:** La capa Model implementa las interfaces definidas en la capa API Model y contiene la lógica de negocio del sistema. Aquí se procesan los datos, se realizan las operaciones y se aplican las reglas de negocio necesarias. Esta capa se encarga de la lógica interna del sistema y se comunica con las capas superiores e inferiores a través de las interfaces definidas en la capa API Model. También puede incluir la interacción con fuentes de datos externas, servicios web u otros sistemas.

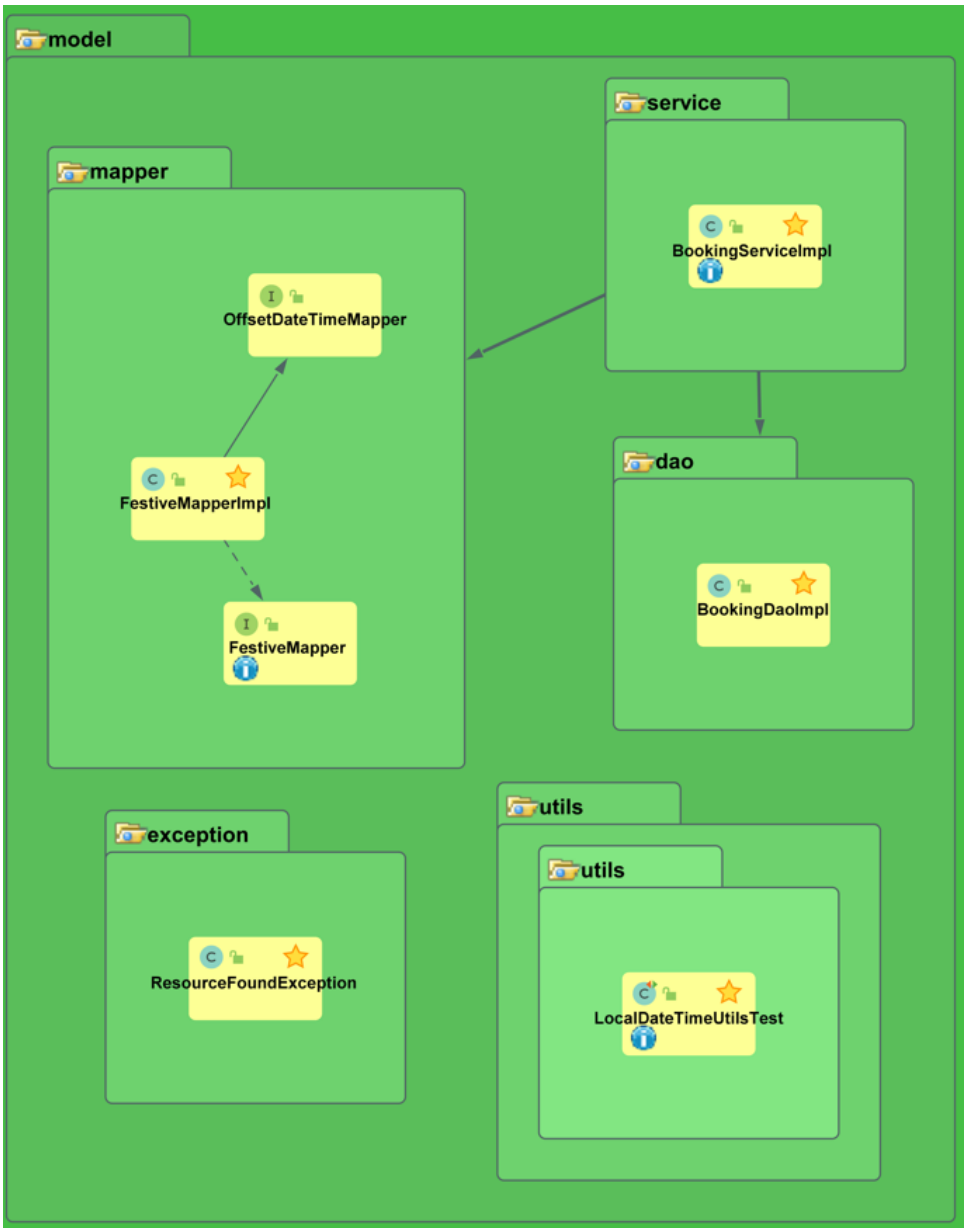

Figura 4.6: Diagrama de estructura de módulos

- **Módulo Web Service:** En esta capa se encuentran los controladores y la seguridad de acceso. Los controladores reciben las peticiones del cliente y se encargan de enrutarlas a la lógica de negocio correspondiente en la capa Model. Aquí se definen los puntos de entrada del sistema, como las URL y los métodos HTTP. Además, se implementa la seguridad de acceso para garantizar que solo los usuarios autorizados puedan acceder a los recursos protegidos del sistema. Esta capa hace uso de los servicios proporcionados

por la capa Model para realizar las operaciones solicitadas por los clientes.

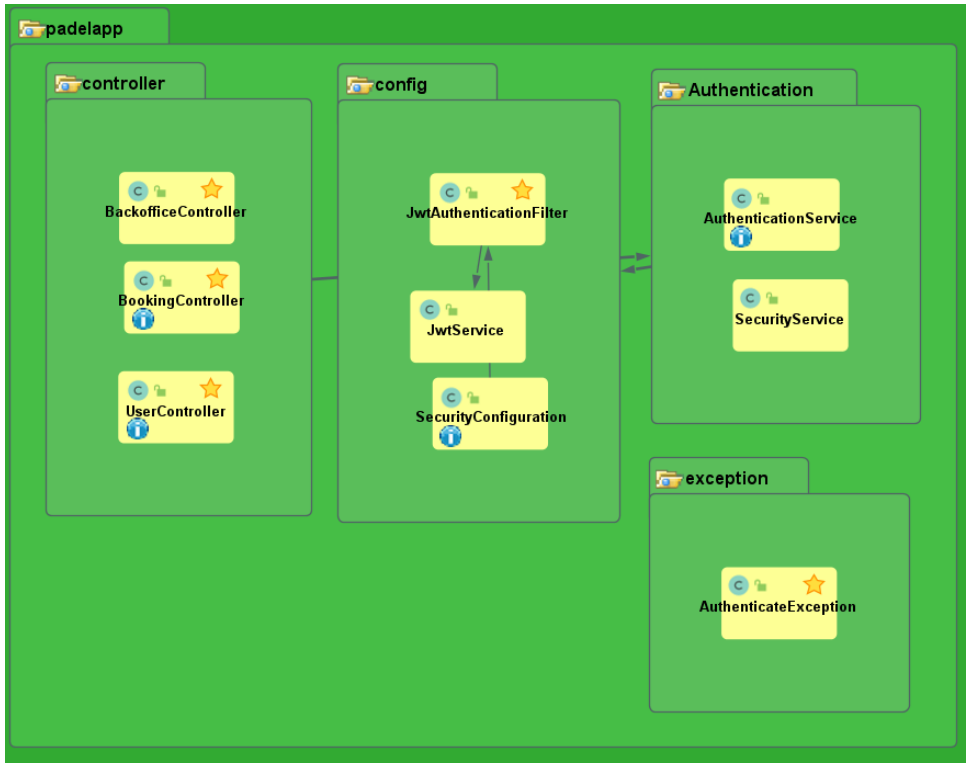

Figura 4.7: Diagrama de estructura de módulos

- **Módulo Boot:** La capa Boot es la capa de configuración de la aplicación y donde se encuentra el bot de Telegram que llama a los servicios del Model. Aquí se establecen las configuraciones específicas del entorno de ejecución, como la conexión a la base de datos, la configuración de loggin, la gestión de dependencias y la configuración del bot de Telegram. Esta capa es esencial para iniciar y configurar la aplicación, asegurando que todos los componentes estén correctamente inicializados y listos para su ejecución.

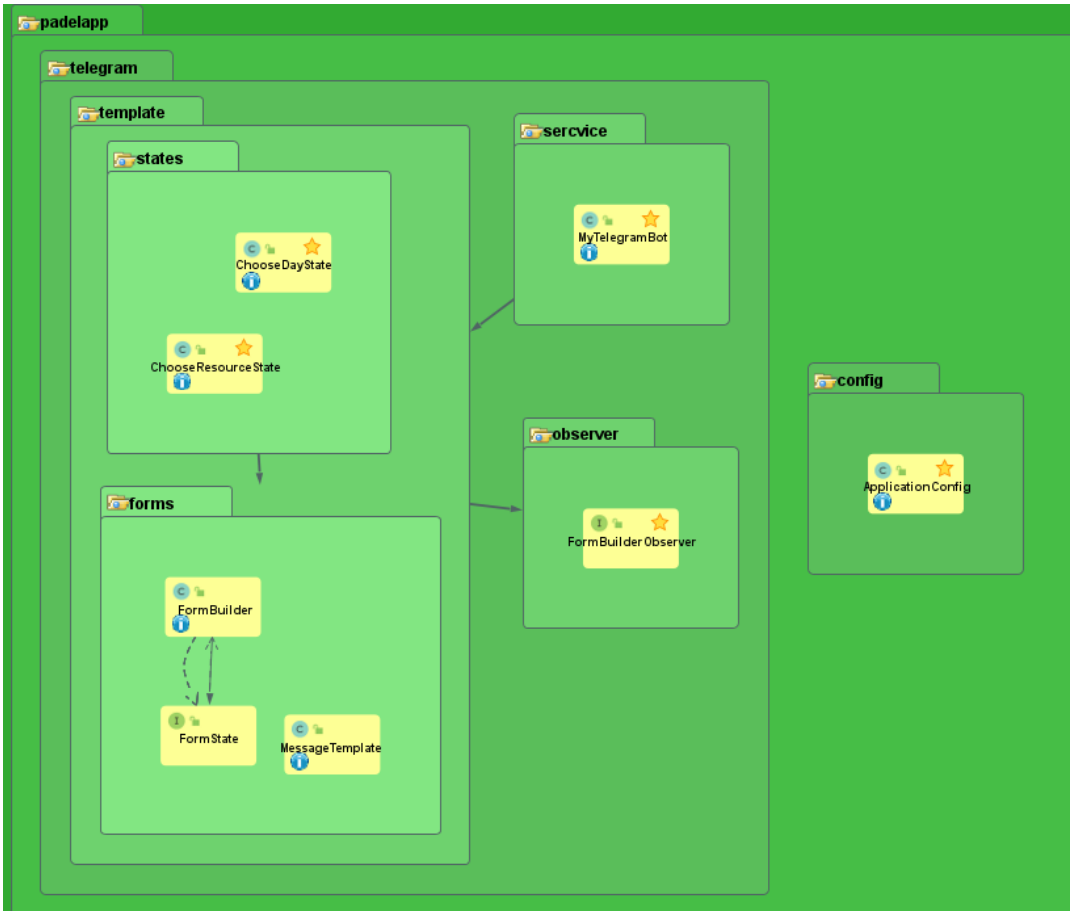

Figura 4.8: Diagrama de estructura de módulos

La combinación de estas capas y su interacción proporciona una estructura modular y cohesiva para tu proyecto. La capa API Model establece los contratos y las interfaces del sistema, la capa Model implementa la lógica de negocio, la capa Web Service maneja las solicitudes de los clientes y la seguridad de acceso, y la capa Boot se encarga de la configuración, la inicialización de la aplicación y el bot de Telegram.

4.4.3. Modelo de domino

A continuación se presentan las entidades mas relevantes que participan en el modelo de dominio y su esquema en la siguiente pagina.

Booking:

- `bookingId` - Tipo de dato: `Long`. Descripción: identificador de la reserva.

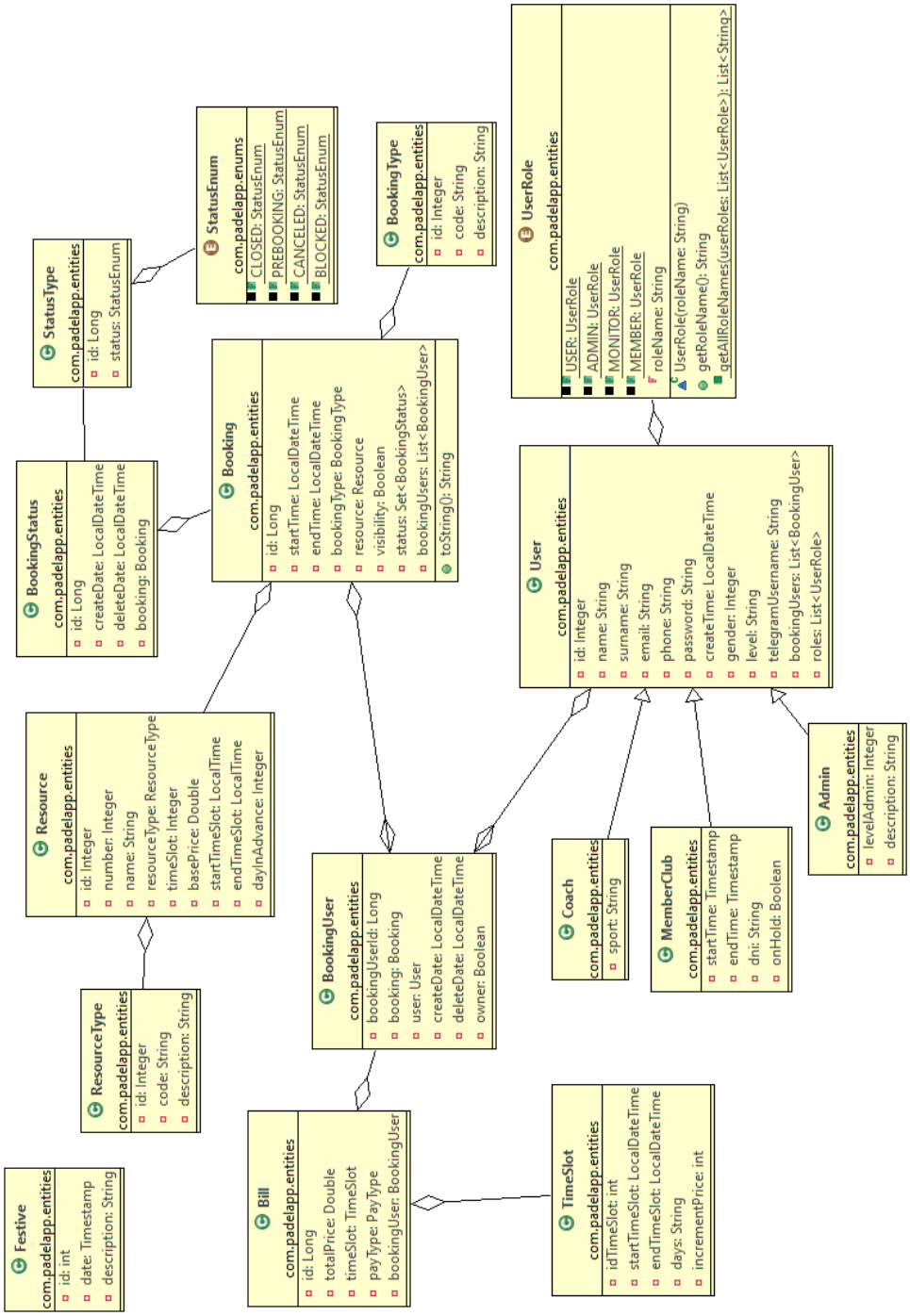

Figura 4.9: Modelo de dominio

- `bookingName` - Tipo de dato: `String`. Descripción: nombre de la reserva.
- `bookingDate` - Tipo de dato: `LocalDate`. Descripción: fecha de la reserva.
- `bookingTime` - Tipo de dato: `LocalTime`. Descripción: hora de la reserva.
- `bookingStatus` - Tipo de dato: `String`. Descripción: estado de la reserva.
- `bookingUserList` - Tipo de dato: `List<BookingUser>`. Descripción: lista de usuarios asociados a la reserva.

BookingUser:

- `bookingUserId` - Tipo de dato: `Long`. Descripción: identificador del usuario de reserva.
- `booking` - Tipo de dato: `Booking`. Descripción: reserva asociada al usuario.
- `user` - Tipo de dato: `User`. Descripción: usuario asociado a la reserva.
- `createDate` - Tipo de dato: `LocalDateTime`. Descripción: fecha de creación.
- `deleteDate` - Tipo de dato: `LocalDateTime`. Descripción: fecha de eliminación.
- `owner` - Tipo de dato: `Boolean`. Descripción: indica si el usuario es el propietario.

User:

- `id` - Tipo de dato: `Integer`. Descripción: identificador del usuario.
- `name` - Tipo de dato: `String`. Descripción: nombre del usuario (obligatorio).
- `surname` - Tipo de dato: `String`. Descripción: apellido del usuario.
- `email` - Tipo de dato: `String`. Descripción: correo electrónico del usuario.
- `phone` - Tipo de dato: `String`. Descripción: número de teléfono del usuario.
- `password` - Tipo de dato: `String`. Descripción: contraseña del usuario (obligatorio).
- `createTime` - Tipo de dato: `LocalDateTime`. Descripción: fecha y hora de creación del usuario (obligatorio).
- `gender` - Tipo de dato: `Integer`. Descripción: género del usuario.
- `level` - Tipo de dato: `String`. Descripción: nivel del usuario.
- `telegramUsername` - Tipo de dato: `String`. Descripción: nombre de usuario de Telegram.
- `bookingUsers` - Tipo de dato: `List<BookingUser>`. Descripción: lista de usuarios asociados a reservas.

- `roles` - Tipo de dato: `List<UserRole>`. Descripción: lista de roles del usuario (transitorio).

Resource:

- `id` - Tipo de dato: `Integer`. Descripción: identificador del recurso.
- `number` - Tipo de dato: `Integer`. Descripción: número del recurso.
- `name` - Tipo de dato: `String`. Descripción: nombre del recurso.
- `resourceType` - Tipo de dato: `ResourceType`. Descripción: tipo de recurso asociado.
- `timeSlot` - Tipo de dato: `Integer`. Descripción: intervalo de tiempo del recurso.
- `basePrice` - Tipo de dato: `Double`. Descripción: precio base del recurso.
- `startTimeSlot` - Tipo de dato: `LocalTime`. Descripción: hora de inicio del intervalo de tiempo del recurso.
- `endTimeSlot` - Tipo de dato: `LocalTime`. Descripción: hora de finalización del intervalo de tiempo del recurso.
- `dayInAdvance` - Tipo de dato: `Integer`. Descripción: días de anticipación para reservar el recurso.

TimeSlot:

- `idTimeSlot` - Tipo de dato: `Integer`. Descripción: identificador de la franja horaria.
- `startTimeSlot` - Tipo de dato: `DateTime`. Descripción: hora de inicio de la franja horaria.
- `endTimeSlot` - Tipo de dato: `DateTime`. Descripción: hora de finalización del intervalo de tiempo.
- `days` - Tipo de dato: `String`. Descripción: días en los que está disponible el intervalo de tiempo.
- `incrementPrice` - Tipo de dato: `Integer`. Descripción: incremento de precio para este intervalo de tiempo.

4.4.4. Diseño de la base de datos

En el esquema del modelo relacional se han identificado varias entidades que representan los diferentes conceptos y objetos del sistema (Figura 4.13). A continuación, se detallan las entidades principales junto con una breve descripción de sus atributos:

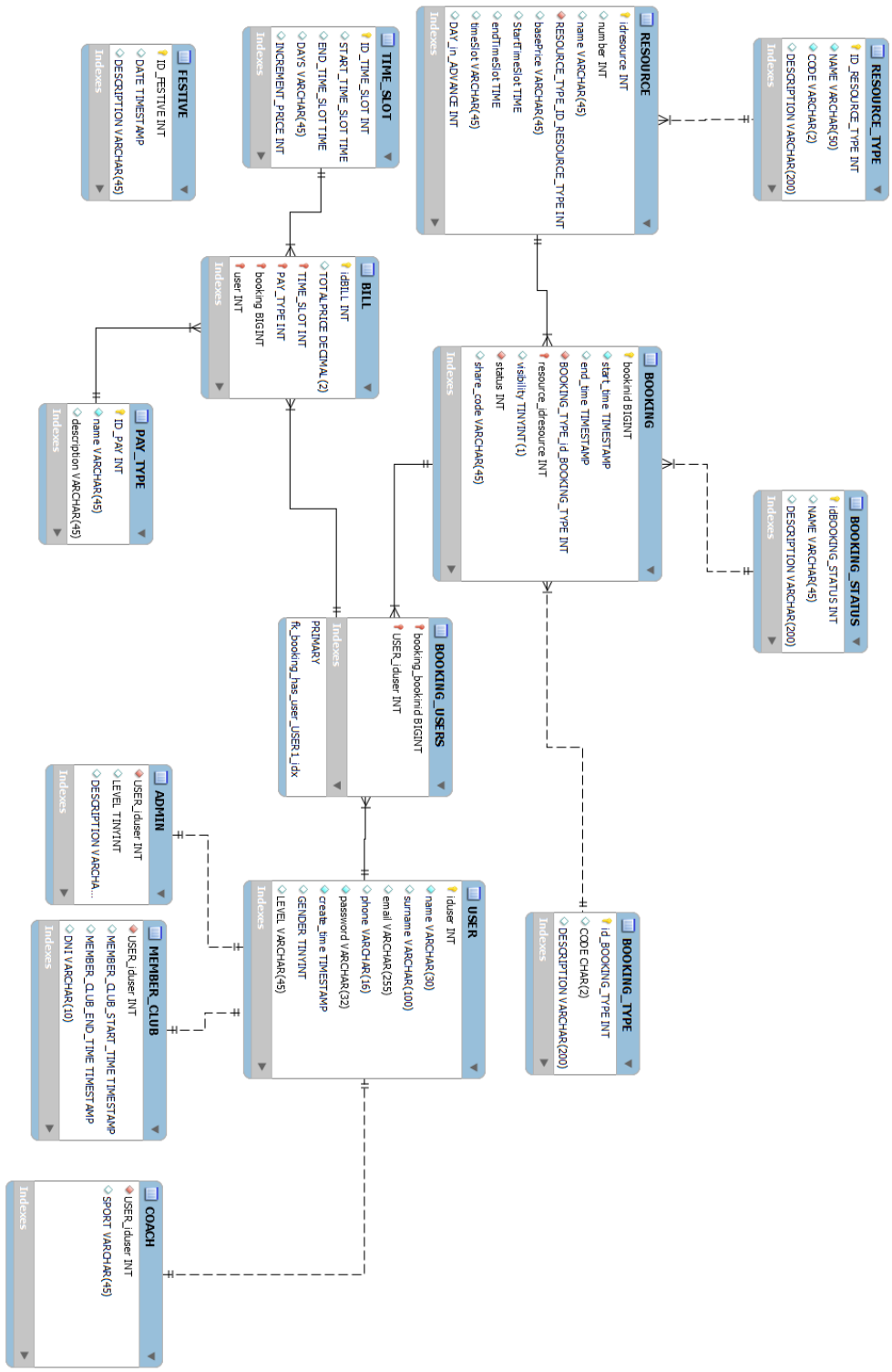

Figura 4.10: Diagrama relacional

Entidad: `USER`

La entidad `USER` representa a los usuarios del sistema. Algunos atributos son:

- `IDUSER`: Identificador del usuario.
- `NAME`: Nombre del usuario.
- `SURNAME`: Apellido del usuario.
- `EMAIL`: Correo electrónico del usuario.
- `PHONE`: Número de teléfono del usuario.
- `PASSWORD`: Contraseña del usuario.
- `CREATE_TIME`: Fecha y hora de creación del usuario.
- `GENDER`: Género del usuario.
- `LEVEL`: Nivel del usuario.
- `USERNAME`: Nombre de usuario del usuario.
- `USER_TYPE`: Tipo de usuario.
- `DESCRIPTION`: Descripción del usuario.
- `SPORT`: Deporte relacionado con el usuario.
- `DNI`: Número de identificación del usuario.
- `ON_HOLD`: Indicador de si el usuario está en espera para ser miembro del club.

Entidad: `BOOKING`

La entidad `BOOKING` representa una reserva en el sistema. Algunos atributos importantes de esta entidad son:

- `BOOKINID`: Identificador de la reserva.
- `START_TIME`: Fecha y hora de inicio de la reserva.
- `END_TIME`: Fecha y hora de finalización de la reserva.
- `BOOKING_TYPE_ID_BOOKING_YPE`: Identificador del tipo de reserva.
- `RESOURCE_IDRESOURCE`: Identificador del recurso reservado.
- `BOOKING_CREATE_DATE`: Fecha y hora de creación de la reserva.
- `BOOKING_DELETE_DATE`: Fecha y hora de eliminación de la reserva.
- `VISIBILITY`: Indicador de visibilidad de la reserva.

Entidad: BOOKING_STATUS

La entidad BOOKING_STATUS representa el estado de una reserva. Algunos atributos relevantes son:

- STATUS_ID: Identificador del estado.
- CREATE_DATE: Fecha y hora de creación del estado.
- DELETE_DATE: Fecha y hora de eliminación del estado.
- BOOKING: Identificador de la reserva asociada al estado.

Entidad: STATUS_TYPE

La entidad STATUS_TYPE representa los diferentes tipos de estado de una reserva. Algunos atributos son:

- STATUS_id: Identificador del estado.
- STATUS: Descripción del estado.

Entidad: BOOKING_USER

La entidad BOOKING_USER representa la relación entre una reserva y un usuario. Algunos atributos relevantes son:

- BOOKING_ID: Identificador de la reserva.
- USER_ID: Identificador del usuario.
- OWNER: Indicador de si el usuario es propietario de la reserva.
- BOOKING_BOOKINID: Identificador de la reserva asociada.
- USER_IDUSER: Identificador del usuario relacionado.

Entidad: BOOKING_TYPE

La entidad BOOKING_TYPE representa los diferentes tipos de reserva disponibles. Algunos atributos de esta entidad son:

- ID_BOOKING_TYPE: Identificador del tipo de reserva.
- CODE: Código del tipo de reserva.
- DESCRIPTION: Descripción del tipo de reserva.

Entidad: `RESOURCE`

La entidad `RESOURCE` representa los recursos disponibles para las reservas. Algunos atributos relevantes son:

- `IDRESOURCE`: Identificador del recurso.
- `NUMBER`: Número del recurso.
- `NAME`: Nombre del recurso.
- `RESOURCE_TYPE_ID_RESOURCE_TYPE`: Identificador del tipo de recurso.
- `BASE_PRICE`: Precio base del recurso.
- `START_TIME_SLOT`: Hora de inicio del intervalo de tiempo del recurso.
- `END_TIME_SLOT`: Hora de finalización del intervalo de tiempo del recurso.
- `TIME_SLOT`: Duración del intervalo de tiempo del recurso.
- `DAY_IN_ADVANCE`: Días de antelación requeridos para la reserva del recurso.

Entidad: `RESOURCE_TYPE`

La entidad `RESOURCE_TYPE` representa los diferentes tipos de recursos disponibles. Algunos atributos son:

- `ID_RESOURCE_TYPE`: Identificador del tipo de recurso.
- `NAME`: Nombre del tipo de recurso.
- `CODE`: Código del tipo de recurso.
- `DESCRIPTION`: Descripción del tipo de recurso.

Entidad: `admin`

Esta entidad representa a los administradores del sistema. Algunos de los atributos de esta entidad son:

- `USER_iduser`: Identificador del usuario administrador.
- `LEVEL_ADMIN`: Nivel de privilegios del administrador.
- `DESCRIPTION`: Descripción del administrador.

Entidad: `COACH`

La entidad `COACH` representa a los entrenadores del sistema.

- `USER_IDUSER`: Identificador del usuario entrenador.
- `SPORT`: Deporte al que está asociado el entrenador.

Entidad: `FESTIVE`

La entidad `FESTIVE` representa los días festivos en el sistema.

- `ID_FESTIVE`: Identificador del día festivo.
- `DATE`: Fecha del día festivo.
- `DESCRIPTION`: Descripción del día festivo.

Entidad: `MEMBER_club`

La entidad `MEMBER_club` representa los miembros de un club.

- `USER_IDUSER`: Identificador del usuario miembro del club.
- `MEMBER_CLUB_START_TIME`: Fecha y hora de inicio de la membresía.
- `MEMBER_CLUB_END_TIME`: Fecha y hora de finalización de la membresía.
- `DNI`: Número de identificación del miembro del club.

Entidad: `TIME_SLOT`

La entidad `TIME_SLOT` representa los diferentes intervalos de tiempo disponibles para las reservas. Algunos atributos son:

- `ID_TIME_SLOT`: Identificador de la franja horaria.
- `START_TIME_SLOT`: Hora de inicio de la franja horaria.
- `END_TIME_SLOT`: Hora de finalización de la franja horaria.
- `DAYS`: Días en los que está disponible el intervalo de tiempo.
- `INCREMENT_PRICE`: Precio adicional por utilizar este intervalo de tiempo.

Entidad: `BILL`

La entidad `BILL` representa las facturas generadas por las reservas. Algunos atributos relevantes de esta entidad son:

- `IDBILL`: Identificador de la factura.
- `TOTAL_PRICE`: Precio total de la factura.
- `TIME_SLOT`: Intervalo de tiempo de la reserva.
- `PAY_TYPE`: Tipo de pago utilizado.
- `BOOKING`: Identificador de la reserva asociada a la factura.
- `USER`: Identificador del usuario relacionado con la factura.

Entidad: `PAY_TYPE`

La entidad `PAY_TYPE` representa los diferentes tipos de pago disponibles. Para cuando se implemente la funcionalidad de pago.

- `ID_PAY`: Identificador del tipo de pago.
- `NAME`: Nombre del tipo de pago.
- `DESCRIPTION`: Descripción del tipo de pago.

Estas son algunas de las entidades presentes en el modelo relacional de la base de datos. Cada entidad tiene sus atributos específicos que permiten almacenar y relacionar la información de manera organizada y coherente.

4.4.5. Diagrama de secuencia

En esta sección, se presentan los diagramas de secuencia más relevantes de la API, con el propósito de brindar una visión clara y detallada de las interacciones y flujos de datos que ocurren durante su funcionamiento. Estos diagramas permiten visualizar de manera gráfica la secuencia de eventos y las interacciones entre los diferentes componentes de la API, lo que facilita la comprensión de cómo se llevan a cabo las operaciones y se procesan los datos en el sistema. En lugar de detallar exhaustivamente cada diagrama de secuencia para cada interacción individual, se mencionaran unos pocos ya que siguen el mismo patrón.

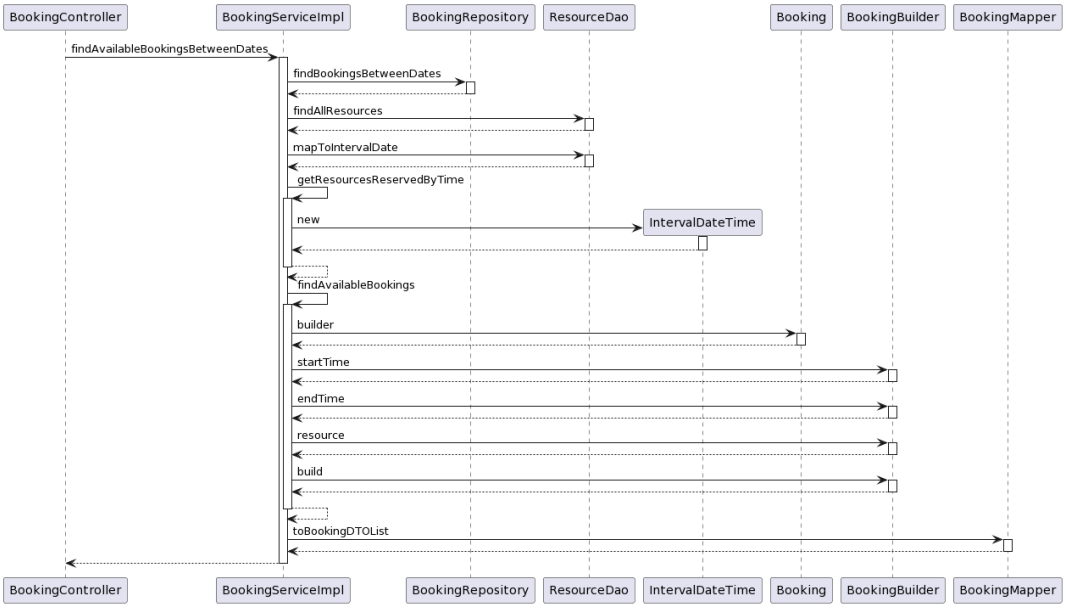

Figura 4.11: Diagrama de secuencia: búsqueda de recurso disponible entre dos fechas

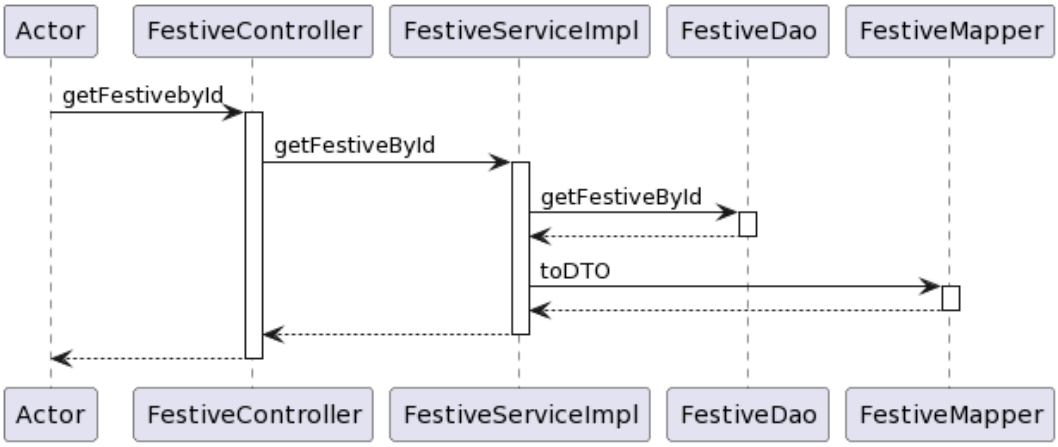

Figura 4.12: Diagrama de secuencia: obtener un día festivo mediante id.

4.5. Diseño del bot de Telegram

En esta sección, se detalla el diseño del bot de Telegram como parte integral de la aplicación desarrollada. El bot de Telegram se ha implementado con el propósito de permitir a

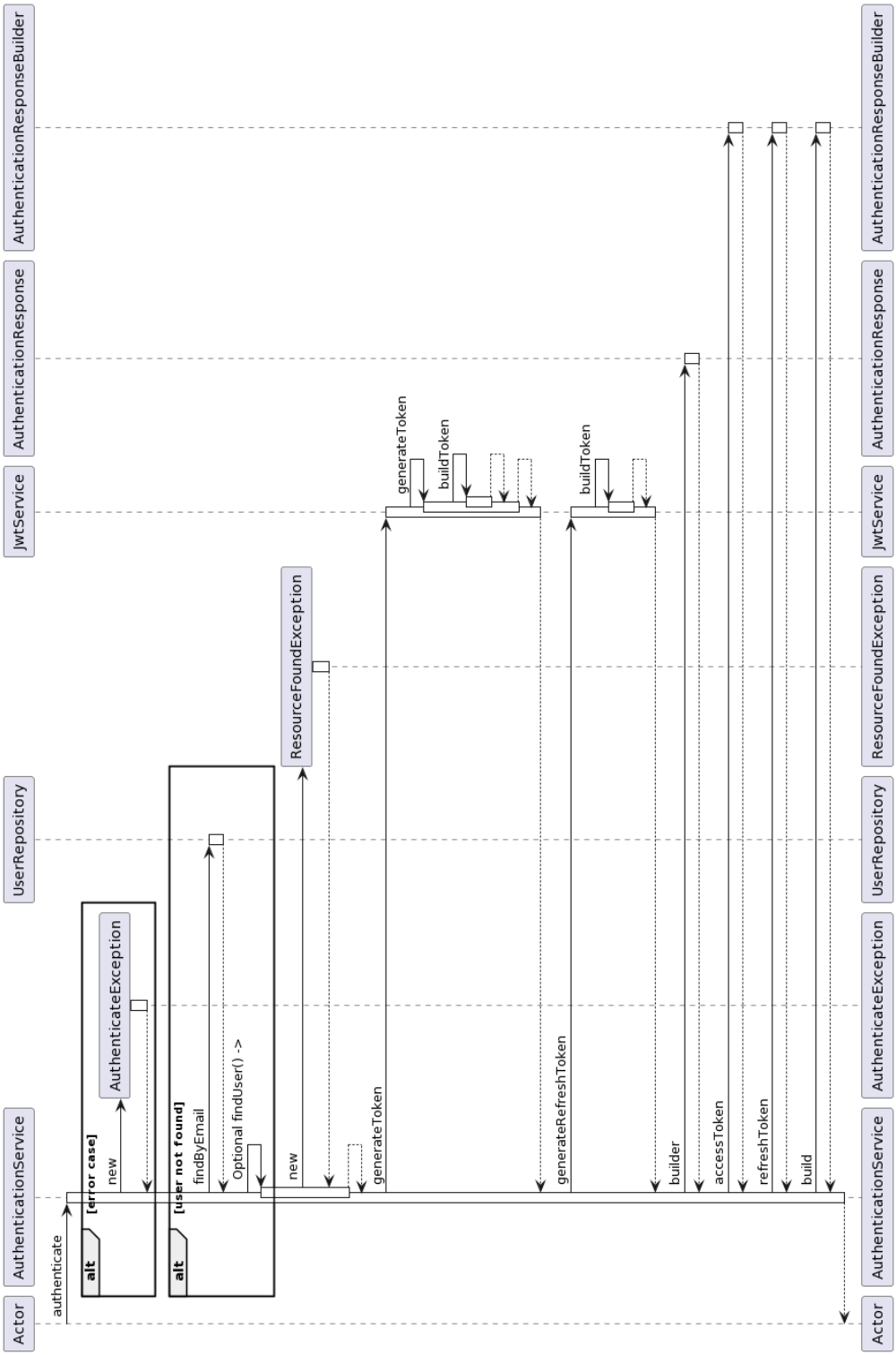

Figura 4.13: Diagrama de secuencia: autenticación de usuarios.

los usuarios interactuar con el sistema de gestión de reservas de pádel de manera conveniente y accesible. A continuación, se explica el flujo de interacción del bot con los usuarios. Se detallan las funcionalidades principales que el bot ofrece, como la visualización de horarios disponibles, la realización de reservas, la consulta de reservas existentes y la gestión de notificaciones. Se describen los comandos y/o botones que los usuarios pueden utilizar para acceder a estas funcionalidades, y se proporciona una explicación clara de cómo se procesan y responden las solicitudes del usuario.

En términos de diseño de la conversación, se considera la implementación de un modelo de diálogo estructurado que guíe a los usuarios a través de las diferentes opciones y acciones disponibles. Se define la lógica del bot para interpretar los mensajes de los usuarios, validar la información ingresada y proporcionar respuestas claras y concisas.

Se han utilizado varios patrones de diseño con el objetivo de mejorar la modularidad, la flexibilidad y la mantenibilidad del software implementado, así como para ofrecer una comunicación guiada al usuario del bot de Telegram. A continuación, se describen los patrones de diseño utilizados:

4.5.1. Patrón Builder

Este patrón se ha utilizado para la construcción de objetos complejos paso a paso. En el contexto del bot de Telegram, se ha empleado el patrón Builder en la implementación de la clase `FormBuilder`, que se encarga de construir formularios interactivos para guiar la interacción con el usuario. El Builder permite crear instancias de `FormBuilder` de manera flexible y escalable, configurando los pasos y opciones del formulario de manera modular.[17]

4.5.2. Patrón Estado

Se ha utilizado para modelar las diferentes etapas o estados por las que puede pasar el formulario interactivo del bot de Telegram. Cada estado representa una situación específica en el flujo de interacción con el usuario.Un ejemplo es la la clase `ChooseResourceState` que permite seleccionar un recurso implementa la interfaz `FormState`, que define los métodos comunes para gestionar los diferentes estados del formulario. El patrón Estado facilita la gestión y transición entre los distintos estados del formulario. [18]

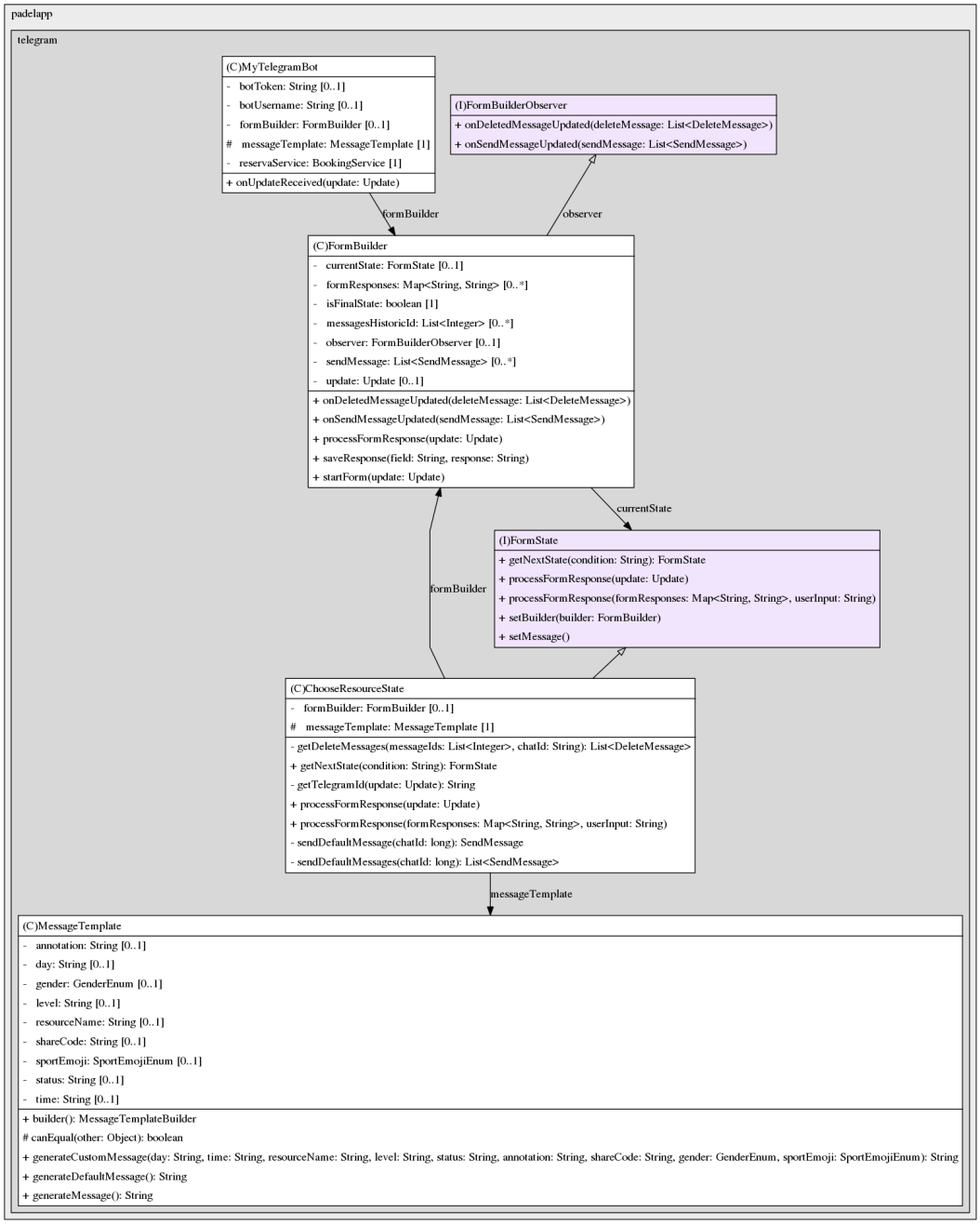

Figura 4.14: Modelo de dominio: Patrones aplicados en el diseño

4.5.3. Patrón Plantilla

El patrón Plantilla se ha aplicado en el diseño de la clase `MessageTemplate`. Esta clase define una estructura básica para la generación de mensajes personalizados que se enviarán al usuario. Mediante el uso del patrón Plantilla, se pueden generar mensajes con contenido dinámico basado en variables y condiciones específicas, lo que permite una comunicación guiada y adaptada a las necesidades del usuario. [19]

4.5.4. Patrón Observador

El patrón Observador se utiliza para establecer una relación de dependencia uno a muchos entre objetos, de modo que cuando un objeto cambia de estado, todos los objetos dependientes son notificados y actualizados automáticamente. En este proyecto, se ha utilizado el patrón Observador en la interfaz `FormBuilderObserver`. Esta interfaz define métodos como `onDeletedMessageUpdated` y `onSendMessageUpdated`. La clase `FormBuilder` tiene una asociación con `FormBuilderObserver` a través del atributo `observer`. El patrón Observador ha permitido la comunicación efectiva entre objetos y ha asegurado la consistencia cuando ocurren cambios en el estado de los formularios. [20]

4.5.5. Diagramas de actividad

En esta sección, se detallan los diagramas de actividad más relevantes del bot de Telegram y la comunicación con la API a la hora de reservar un recurso.

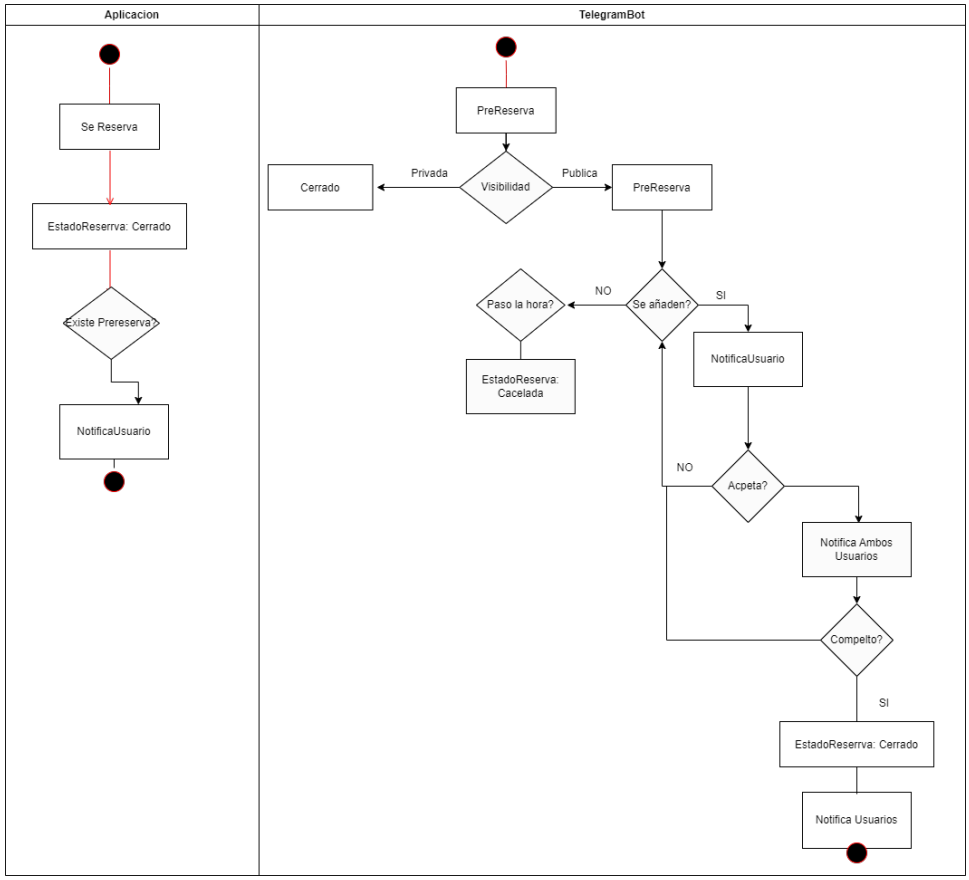

Figura 4.15: Diagrama de actividad reserva de recurso

Capítulo 5

Implementación

En este capítulo, se presenta la estructura final del código tanto del bot como de la API desarrollados en el contexto de este proyecto. Además, se detalla las dependencias utilizadas asi como las medidas tomadas para adapterse al diseño.

5.1. API openapi-generator

Para agilizar y simplificar el proceso de implementación se ha optado por el uso del plugin "openapi-generator-maven-plugin"[21] ya que es de gran utilidad en el desarrollo de un API para utomatizar la generacion de codigo a partir de la especificacion de OpenAPI.

El plugin openapi-generator-maven-plugin es una herramienta que se integra con Maven, una conocida herramienta de construcción y gestión de proyectos en Java. Este plugin permite generar automáticamente el código fuente de la API, así como otros artefactos relacionados, a partir de la especificación OpenAPI.

La ventaja clave de este plugin es que elimina gran parte del trabajo manual y propenso a errores que implica escribir el código de la API desde cero. En su lugar, simplemente proporcionamos la especificación OpenAPI, que describe la estructura de la API, los endpoints, los modelos de datos y otros detalles importantes.

Al configurar el plugin en el archivo pom.xml del proyecto Maven, podemos especificar las opciones de generación, como el lenguaje de programación de destino, el directorio de salida y las bibliotecas adicionales que se deben incluir en el código generado.

```xml
<plugin>
    <groupId>org.openapitools</groupId>
    <artifactId>openapi-generator-maven-plugin</artifactId>
    <version>5.3.0</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <inputSpec>${project.basedir}/src/main/resources/swagger/apiPAdel.yaml</inputSpec>
                <generatorName>spring</generatorName>
                <apiPackage>com.api.apimodel</apiPackage>
                <modelPackage>com.api.model</modelPackage>
                <configOptions>
                    <delegatePattern>true</delegatePattern>
                    <serializableModel>true</serializableModel>
                    <!-- javax.* to jakarta.* -->
                    <useSpringBoot3>true</useSpringBoot3>
                    <documentationProvider>springdoc</documentationProvider>
                    <interfaceOnly>true</interfaceOnly>
                    <additionalModelTypeAnnotations>@lombok.Builder @lombok.NoArgsConstructor
                        @lombok.AllArgsConstructor
                    </additionalModelTypeAnnotations>
                </configOptions>
            </configuration>
```

Figura 5.1: Configuración plugin openapi-generator

Cuando ejecutamos el comando de construcción de Maven, el plugin openapi-generator-maven-plugin procesa la especificación OpenAPI y genera automáticamente el código fuente de la API, que puede ser fácilmente integrado en nuestro proyecto.

Esta generación automática de código ahorra tiempo y reduce la posibilidad de errores, ya que el código se basa directamente en la especificación OpenAPI, que es la fuente de verdad de la API. Además, el plugin puede ser configurado para adaptarse a las necesidades específicas del proyecto, permitiendo personalizar el código generado según las convenciones y estándares establecidos.

Otra ventaja del uso de este plugin es que, al generar el código a partir de la especificación OpenAPI, se mantiene una fuerte coherencia entre la documentación de la API y su implementación. Esto facilita la comprensión y el mantenimiento de la API a lo largo del tiempo, ya que cualquier cambio en la especificación OpenAPI puede ser fácilmente reflejado en el código generado.

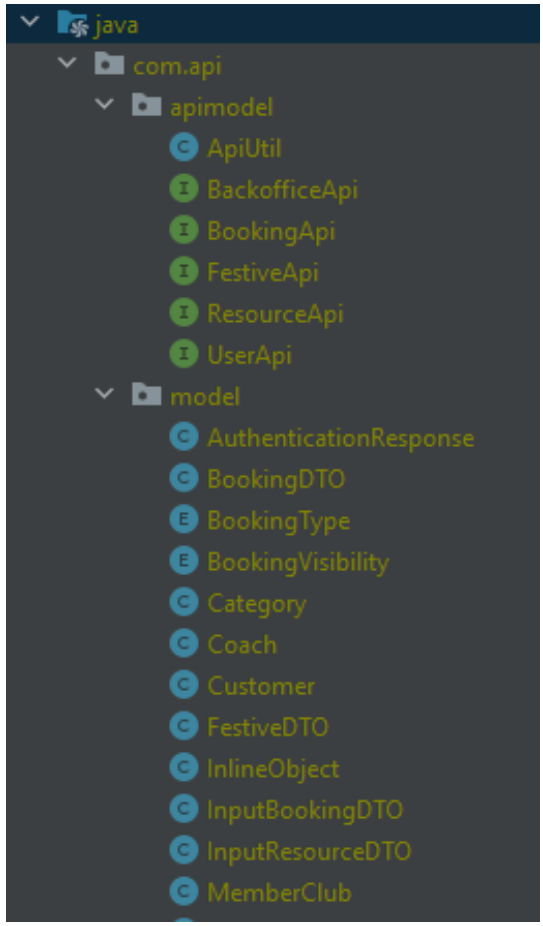

Figura 5.2: Interfaces de los controladores y estructuras de datos autogeneradas

5.1.1. Dependecias

En este apartado se detallan las dependencias utilizadas en el proyecto para facilitar la programación y aumentar la reutilización del código. Cada dependencia ofrece beneficios específicos que contribuyen al desarrollo eficiente y a la calidad del software.

- **JPA (Java Persistence API)**: JPA es una especificación estándar de Java que proporciona una interfaz de programación para administrar la persistencia de los datos en aplicaciones Java. Se utiliza para mapear objetos Java a tablas en una base de datos relacional y realizar operaciones de consulta y manipulación de datos de manera sencilla y eficiente. La interfaz JpaRepository de Spring Data JPA define métodos predefinidos para realizar operaciones comunes en una entidad, como guardar, eliminar y buscar registros. Sin embargo, en ocasiones es necesario realizar consultas personalizadas que no se ajustan a las operaciones predefinidas.

Para ello, se utiliza la anotación @Query junto con una consulta en lenguaje específico, como JPQL (Java Persistence Query Language) o SQL, para definir una consulta personalizada. Esta anotación se coloca encima de un método en el repositorio de la entidad correspondiente.

```java
@Repository
@Transactional
public interface BookingRepository extends JpaRepository<Booking, Long> {

    1 usage  new *
    @Query("SELECT b FROM Booking b WHERE b.startTime >= :startDate AND b.endTime <= :endDate")
    List<Booking> findBookingsBetweenDates(@Param("startDate") LocalDateTime startDate, @Param("endDate") LocalDateTime endDate);
```

Figura 5.3: Uso de la dependencia JPA.

- **Lombok**: Lombok es una biblioteca de Java que ayuda a reducir la cantidad de código repetitivo y tedioso al generar automáticamente métodos getter, setter, constructores, toString y otros métodos comunes en tiempo de compilación. Esto mejora la legibilidad del código y acelera el desarrollo al evitar la escritura manual de código redundante.

- **MapStruct**: MapStruct es una biblioteca de generación de código que simplifica el mapeo de objetos Java entre diferentes estructuras de datos. Permite generar automáticamente código de mapeo entre DTOs (Data Transfer Objects), entidades JPA y otros objetos relacionados, evitando así la necesidad de escribir código de mapeo manualmente y reduciendo el riesgo de errores.

```java
@Component
@Mapper(componentModel = "spring", unmappedTargetPolicy = ReportingPolicy.IGNORE,
        uses = {OffsetDateTimeMapper.class})
public interface FestiveMapper {
        1 usage  1 implementation  Manuel comesaña
        @Named("FestiveDTO")
        @Mapping(source = "id", target = "festiveId")
        @Mapping(source = "date", target = "date", qualifiedByName="OffsetDateTime")
        FestiveDTO toDTO(Festive festive);
}
```

Figura 5.4: Ejemplo de mapeo usando MapStruct

- **Hibernate**: Hibernate es un framework de mapeo objeto-relacional (ORM) que simplifica la interacción entre una aplicación Java y una base de datos relacional. Proporciona una capa de abstracción sobre el almacenamiento de datos y ofrece una forma sencilla y declarativa de realizar operaciones de persistencia y consulta.

- **SLF4J (Simple Logging Facade for Java)**:Es una herramienta que proporciona una abstracción para el registro de eventos en aplicaciones JavaL.Simplifica el proceso de registro de eventos al proporcionar una fachada común para diferentes sistemas de registro. Permite escribir código de registro independiente del sistema de registro

subyacente y brinda la flexibilidad de cambiar de sistema de registro sin modificar el código existente. Facilita la detección de problemas y errores en el código, contribuye a mejorar la capacidad de depuración y diagnóstico en el desarrollo de software.[22]

Estas dependencias se han seleccionado cuidadosamente para mejorar la productividad, la calidad del código y la eficiencia del desarrollo en el proyecto. Cada una de ellas cumple un papel fundamental en diferentes aspectos de la programación, como la persistencia de datos, la generación de código y el registro de eventos. Al aprovechar estas dependencias, se logra una mayor modularidad, una mayor reutilización de código y un desarrollo más ágil y eficiente.

5.2. Seguridad

5.2.1. Autentificacion

JwtService: Esta clase se encarga de la generación y validación de tokens JWT. Se utiliza para generar tokens de acceso y tokens de actualización en base a la información del usuario. También proporciona métodos para extraer información del token, como el nombre de usuario. En la implementación del API, se configuran valores como la clave secreta y las expiraciones de los tokens a través de anotaciones de valor (`@Value`) estos se encontraran el el archivo (`aplication.properties`).

JwtAuthenticationFilter: Esta clase es un filtro de autenticación que se ejecuta en cada solicitud HTTP entrante. Su función es extraer el token JWT del encabezado de autorización y autenticar al usuario correspondiente. Utiliza el servicio JwtService para validar el token y obtener la información del usuario. Si la autenticación es exitosa, se establece el contexto de autenticación de Spring Security. Esta clase se registra como un componente en la configuración de Spring para que se aplique a todas las solicitudes.

AuthenticationService: Esta clase es un componente de servicio que contiene la lógica para registrar y autenticar usuarios. En el caso del registro, crea un nuevo objeto User a partir de los datos proporcionados, cifra la contraseña y lo guarda en el repositorio. En la autenticación, utiliza el administrador de autenticación (`AuthenticationManager`) para verificar las credenciales del usuario. Además, utiliza el servicio JwtService para generar tokens JWT y realizar validaciones. La clase también incluye un registrador de eventos (`Logger`) para registrar mensajes relacionados con la autenticación.

5.3. Implementación del bot de Telegram

En la aplicación de gestión de reservas de pádel, se han definido varios puntos de entrada que permiten a los usuarios acceder a diferentes funcionalidades y obtener la información necesaria. A continuación, se explica cada uno de estos puntos de entrada:

1. **/partidas**: Permite a los usuarios obtener información sobre las pistas de pádel disponibles para realizar reservas. Al acceder a este punto, se muestra una lista de las pistas disponibles, seleccionando uno de los próximos cinco días, el horario y la disponibilidad en tiempo real. Los usuarios pueden consultar esta información para seleccionar la pista deseada y proceder con la reserva. Seleccionando en el recurso que deseen podran unirse a una partida o reservarlo de forma privada o pública.

2. **/misreservas**: Este punto de entrada está destinado a proporcionar a los usuarios acceso a sus reservas existentes. Al acceder a este punto, los usuarios pueden ver una lista de las reservas que han realizado, incluyendo detalles como la fecha, la hora y la pista reservada. Además, se puede mostrar información adicional, como el estado de la reserva (confirmada, pendiente, cancelada, etc.) y la posibilidad de realizar modificaciones o cancelaciones.

3. **/info**: El punto de entrada /info ofrece a los usuarios acceso a información general sobre el sistema de gestión de reservas de pádel. Aquí se pueden proporcionar detalles sobre el funcionamiento del sistema, las políticas de reservas, las tarifas, las reglas de uso de las instalaciones, entre otros. Es una sección informativa que brinda a los usuarios una visión completa de cómo utilizar el sistema y qué esperar al realizar una reserva.

4. **/entrenamientos**: Este punto de entrada está dedicado a los entrenamientos de pádel ofrecidos en el centro. Los usuarios pueden acceder a esta sección para obtener información sobre los diferentes tipos de entrenamientos disponibles, los horarios, los niveles de habilidad requeridos y los instructores asignados. También se pueden proporcionar detalles sobre el proceso de inscripción y los beneficios de participar en los entrenamientos.

Capítulo 6

Pruebas

Las pruebas de la API son fundamentales para garantizar el correcto funcionamiento de una aplicación o servicio web. Postman es una herramienta popular que permite realizar pruebas exhaustivas y automatizadas a través de solicitudes HTTP. En este caso, utilizaremos Postman para probar los endpoints de la API específica.

6.1. Objetivos

El objetivo de las pruebas es validar que los endpoints de la API se comporten de acuerdo con las especificaciones y requisitos establecidos. A través de las pruebas, podemos verificar la funcionalidad, el rendimiento, la seguridad y la integridad de la API. También podemos detectar y corregir posibles errores o problemas antes de que la API se implemente en un entorno de producción.

En primer lugar, se realizarán pruebas de caja negra, las cuales permitirán evaluar el comportamiento general de la API sin necesidad de conocer los detalles internos de su implementación. Se probarán los diferentes endpoints de la API, verificando las respuestas esperadas y asegurándose de que los usuarios identificados tengan acceso adecuado según sus roles. Esta etapa de pruebas permitirá validar el flujo general de la aplicación y su capacidad para manejar diferentes situaciones de uso.

En segundo lugar, se llevarán a cabo pruebas de error y excepciones para evaluar cómo la API maneja situaciones específicas de error. Se probarán tanto respuestas correctas como respuestas de error, como por ejemplo, los códigos de error 404 (Recurso no encontrado) y 403 (Acceso no autorizado). Estas pruebas se enfocarán en verificar que la API devuelva códigos de error apropiados, mensajes claros y mantenga la integridad de los datos en casos de excepciones y errores.

6.2. Pruebas de los end-points

Cuadro 6.1: Casos de Prueba para el Endpoint POST /booking

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Datos de reserva válidos	Código de respuesta 200 (Éxito)	Código de respuesta 200
2	Usuario no autorizado	Código de respuesta 403 (No autorizado)	Código de respuesta 403
3	Excepción de validación	Código de respuesta 405 (Excepción de validación)	Código de respuesta 405

Cuadro 6.2: Casos de Prueba para el Endpoint GET /booking{id}

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Datos de reserva válidos	Código de respuesta 200 (Éxito)	Código de respuesta 200
2	Usuario no autorizado	Código de respuesta 403 (No autorizado)	Código de respuesta 403
3	ID de reserva no válido	Código de respuesta 404 (Recurso no encontrado)	Código de respuesta 404

Cuadro 6.3: Casos de Prueba para el Endpoint GET /booking/findByDate

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Fechas válidas	Código de respuesta 200 (Operación exitosa)	Código de respuesta 200
2	Valor de fecha inválido	Código de respuesta 400 (Valor de fecha inválido)	Código de respuesta 400
3	Usuario no autorizado	Código de respuesta 403 (Error de autenticación)	Código de respuesta 403
4	Excepción de validación	Código de respuesta 405 (Excepción de validación)	Código de respuesta 405

Cuadro 6.4: Casos de Prueba para el Endpoint GET /booking/findByType

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Tipo de reserva válido	Código de respuesta 200 (Operación exitosa)	Código de respuesta 200
2	Tipo de reserva inválido	Código de respuesta 400 (Tipo de reserva inválido)	Código de respuesta 400
3	Usuario no autorizado	Código de respuesta 403 (Error de autenticación)	Código de respuesta 403
4	Excepción de validación	Código de respuesta 405 (Excepción de validación)	Código de respuesta 405

Cuadro 6.5: Casos de Prueba para el Endpoint DELETE /booking/bookingId

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	ID de reserva válido	Código de respuesta 204 (Reserva eliminada con éxito)	Código de respuesta 204
2	ID no válido	Código de respuesta 400 (ID de reserva no válido)	Código de respuesta 400
3	Usuario no autorizado	Código de respuesta 403 (No autorizado)	Código de respuesta 403
4	Reserva no encontrada	Código de respuesta 404 (Reserva no encontrada)	Código de respuesta 404

Cuadro 6.6: Casos de Prueba para el Endpoint GET /resource/id

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	ID de recurso válido	Código de respuesta 200 (Operación exitosa) y recurso solicitado	Código de respuesta 200 y recurso solicitado
2	ID de recurso inválido	Código de respuesta 400 (ID de recurso inválido)	Código de respuesta 400
3	Operación no autorizada	Código de respuesta 401 (Operación no autorizada)	Código de respuesta 401
4	Recurso no encontrado	Código de respuesta 404 (Recurso no encontrado)	Código de respuesta 404

Cuadro 6.7: Casos de Prueba para el Endpoint PUT /backoffice/resource/id

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	ID de recurso válido y datos actualizados	Código de respuesta 200 (Operación exitosa) y recurso actualizado	Código de respuesta 200 y recurso actualizado
2	ID de recurso inválido	Código de respuesta 400 (ID de recurso inválido)	Código de respuesta 400
3	Operación no autorizada	Código de respuesta 401 (Operación no autorizada)	Código de respuesta 401
4	Recurso no encontrado	Código de respuesta 404 (Recurso no encontrado)	Código de respuesta 404
5	Excepción de validación	Código de respuesta 405 (Excepción de validación)	Código de respuesta 405

Cuadro 6.8: Casos de Prueba para el Endpoint DELETE /backoffice/resource/id

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	ID de recurso válido	Código de respuesta 204 (Recurso eliminado exitosamente)	Código de respuesta 204
2	ID de recurso inválido	Código de respuesta 400 (ID de recurso inválido)	Código de respuesta 400
3	Operación no autorizada	Código de respuesta 401 (Operación no autorizada)	Código de respuesta 401
4	Recurso no encontrado	Código de respuesta 404 (Recurso no encontrado)	Código de respuesta 404

Cuadro 6.9: Casos de Prueba para el Endpoint POST /user

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Datos del usuario válido	Código de respuesta 200 (Operación exitosa) y usuario creado	Código de respuesta 200 y usuario creado

Cuadro 6.10: Casos de Prueba para el Endpoint GET /user/login

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Nombre de usuario y contraseña válidos	Código de respuesta 200 (Operación exitosa) y token de usuario	Código de respuesta 200 y token de usuario
2	Nombre de usuario y contraseña inválidos	Código de respuesta 405 (Nombre de usuario o contraseña inválidos)	Código de respuesta 405

Cuadro 6.11: Casos de Prueba para el Endpoint POST /backoffice/resource

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Datos del recurso válido	Código de respuesta 200 (Operación exitosa) y recurso creado	Código de respuesta 200 y recurso creado
2	Datos del recurso inválido	Código de respuesta 405 (Entrada inválida)	Código de respuesta 405

Cuadro 6.12: Casos de Prueba para el Endpoint GET /festive/idfestive

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	ID de festivo válido	Código de respuesta 200 (Operación exitosa) y festivo solicitada	Código de respuesta 200 y festivo solicitada
2	ID de festivo inválido	Código de respuesta 400 (ID de festivo inválido)	Código de respuesta 400
3	Festivo no encontrada	Código de respuesta 404 (festivo no encontrada)	Código de respuesta 404

Cuadro 6.13: Casos de Prueba para el Endpoint GET /festive/date

Prueba	Entrada	Resultado Esperado	Resultado Obtenido
1	Fecha de festivo válida	Código de respuesta 200 (Operación exitosa) y festivo solicitada	Código de respuesta 200 y festivo solicitada
2	Fecha de festivo inválida	Código de respuesta 400 (Fecha de festivo inválida)	Código de respuesta 400
3	Festivo no encontrada	Código de respuesta 404 (festivo no encontrada)	Código de respuesta 404

Cuadro 6.14: Casos de Prueba para el Endpoint PUT /backoffice/festive/idfestive

Prueba	**Entrada**	**Resultado Esperado**	**Resultado Obtenido**
1	ID de festivo válido y datos actualizados	Código de respuesta 200 (Operación exitosa) y festivo actualizada	Código de respuesta 200 y festivo actualizada
2	ID de festivo inválido	Código de respuesta 400 (ID de festivo inválido)	Código de respuesta 400

Cuadro 6.15: Casos de Prueba para el Endpoint DELETE /backoffice/festive/idfestive

Prueba	**Entrada**	**Resultado Esperado**	**Resultado Obtenido**
1	ID de festivo válido	Código de respuesta 204 (festivo eliminada exitosamente)	Código de respuesta 204
2	ID de festivo inválido	Código de respuesta 400 (ID de festivo inválido)	Código de respuesta 400
3	Festivo no encontrada	Código de respuesta 404 (festivo no encontrada)	Código de respuesta 404

Capítulo 7

Conclusiones y lineas futuras

7.1. Conclusiones

La implementación de la API REST y el bot de Telegram para reservas ha sido exitosa, proporcionando funcionalidades clave para la gestión de reservas de manera eficiente y conveniente. La API REST ha demostrado ser una herramienta poderosa para la comunicación y manipulación de datos, permitiendo a los usuarios realizar diversas operaciones relacionadas con las reservas de forma segura y confiable. El bot de Telegram ha brindado una interfaz amigable y accesible para que los usuarios realicen reservas y consultas, mejorando la experiencia del usuario y facilitando la interacción con el sistema. Ambas soluciones han contribuido a optimizar el proceso de reserva, agilizando la comunicación y mejorando la eficiencia en la gestión de reservas.

Aunque no se han alcanzado todos los objetivos establecidos inicialmente, el desarrollo del API REST y el bot de Telegram sobre reservas ha sido un paso importante en la dirección correcta. Las funcionalidades implementadas y los conocimientos adquiridos sientan las bases para futuras mejoras y expansiones, permitiendo ofrecer una solución más completa y adaptada a las necesidades de los usuarios.

7.2. Lineas Futuras

A continuación se presentan algunas líneas futuras que podrían explorarse en el desarrollo del API REST y el bot de Telegram sobre reservas:

1. Implementación de Pasarela de Pagos: Se podría considerar la integración de una pasarela de pagos para permitir a los usuarios realizar pagos de manera segura y conveniente al realizar una reserva. Esto mejoraría la experiencia del usuario y brindaría una forma más completa de gestionar todo el proceso de reserva, desde la selección hasta el pago.

2. Interfaz de Usuario con React Native: Para ampliar la accesibilidad y ofrecer una experiencia fluida en dispositivos móviles, se podría considerar la implementación de una interfaz de usuario utilizando React Native. Esto permitiría que la aplicación esté disponible en plataformas móviles, como iOS y Android, y brindaría a los usuarios la capacidad de realizar reservas y acceder a la funcionalidad de manera nativa desde sus dispositivos móviles.

3. Integración con Java Mail Sender: Para mejorar la comunicación con los usuarios, se podría implementar la funcionalidad de envío de correos electrónicos utilizando Java Mail Sender. Esto permitiría enviar notificaciones, confirmaciones de reserva y recordatorios por correo electrónico, manteniendo a los usuarios informados de manera eficaz y automatizada.

4. Inicio de Sesión con Redes Sociales: Para simplificar el proceso de inicio de sesión y aumentar la adopción de la plataforma, se podría agregar la opción de inicio de sesión utilizando cuentas de redes sociales populares, como Facebook, Google o Twitter. Esto permitiría a los usuarios registrarse e ingresar al sistema de manera más rápida y conveniente, utilizando sus credenciales de redes sociales existentes.

5. Implementación de Funcionalidades Avanzadas: Como líneas futuras, se podría considerar la incorporación de funcionalidades avanzadas, como la gestión de disponibilidad en tiempo real, la programación de reservas recurrentes, la generación de informes y estadísticas, entre otras. Estas mejoras proporcionarían una mayor flexibilidad y personalización en la gestión de reservas y permitirían adaptarse a diferentes necesidades y escenarios.

6. Despliegue del bot en otro servidor y creación de una autorización especial para ese bot: Se podría explorar la opción de desplegar el bot de Telegram en un servidor distinto al de la API y crear una autorización especial para ese bot. Esto permitiría una mayor escalabilidad y separación de las funcionalidades, así como la implementación de medidas de seguridad específicas para el bot.

Apéndice A

Manual de uso

Este manual se divide en dos secciones, la primera dedicada al uso de la API y la segunda, al uso del bot de Telegram.

A.1. Manual de uso para front-end

El "Manual de uso para front-end" es una guía completa que proporciona a los usuarios del front-end de la aplicación toda la información necesaria para comunicarse correctamente con el backend. Este manual tiene como objetivo garantizar que el front-end tenga un conocimiento completo de las estructuras de entrada y salida, así como de todas las posibles respuestas del backend.

A continuación, se describen los pasos clave para utilizar el manual de uso:

1. Importar el archivo YAML: El primer paso consiste en importar el archivo YAML que describe la especificación de la API en Swagger Editor desde la URL: `https://editor-next.swagger.io/` (editor de Swagger en línea). Esto permite cargar la estructura de la API y sus end-points en la interfaz del editor. Pulsando en File y despues Import File seleccionamos en archivo que esta en la carpeta del proyecto apiPAdel.Yaml

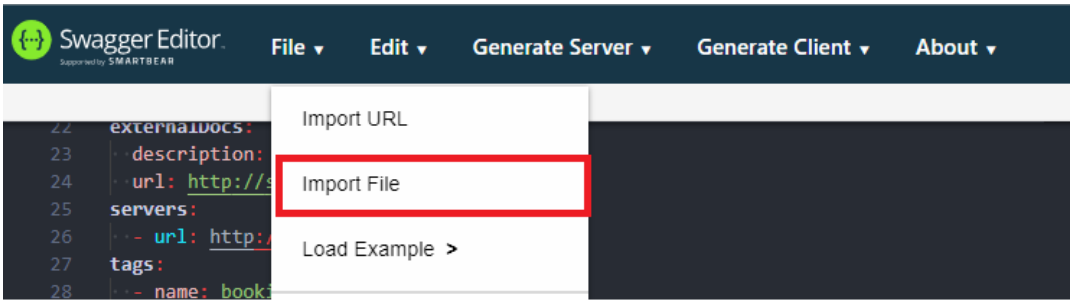

Figura A.1: Como importar el archivo YAML

2. Explorar la estructura de la API: Una vez que el archivo YAML se ha importado correctamente, los usuarios pueden explorar la estructura de la API. Esto implica navegar por los diferentes endpoints, comprender los parámetros de solicitud requeridos y opcionales, y familiarizarse con las posibles respuestas.

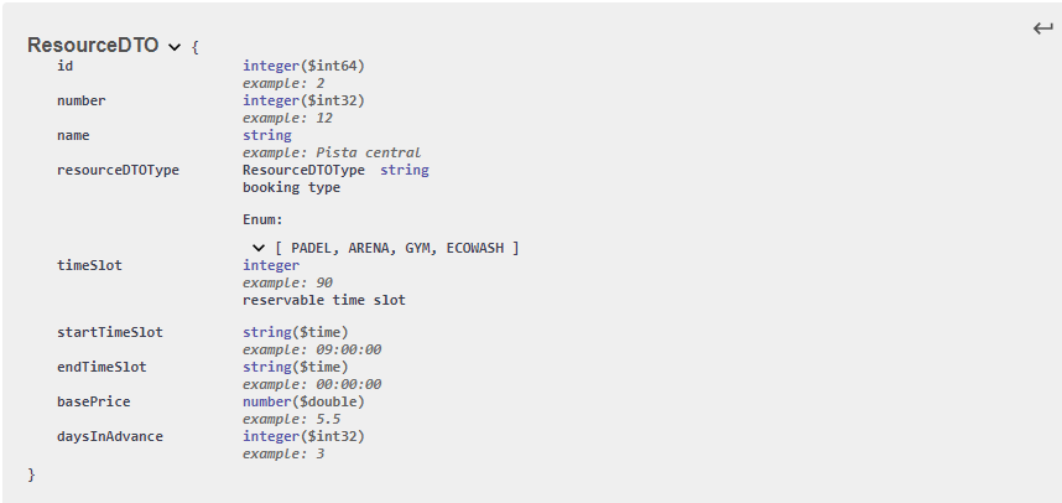

Figura A.2: Ejemplo de la estructura de datos: ResourceDTO

3. Ver ejemplos de solicitudes y respuestas: Swagger Editor proporciona ejemplos de solicitudes y respuestas para cada endpoint de la API. Los usuarios pueden utilizar estos ejemplos como referencia para construir sus propias solicitudes y comprender cómo interpretar las respuestas devueltas por la API.

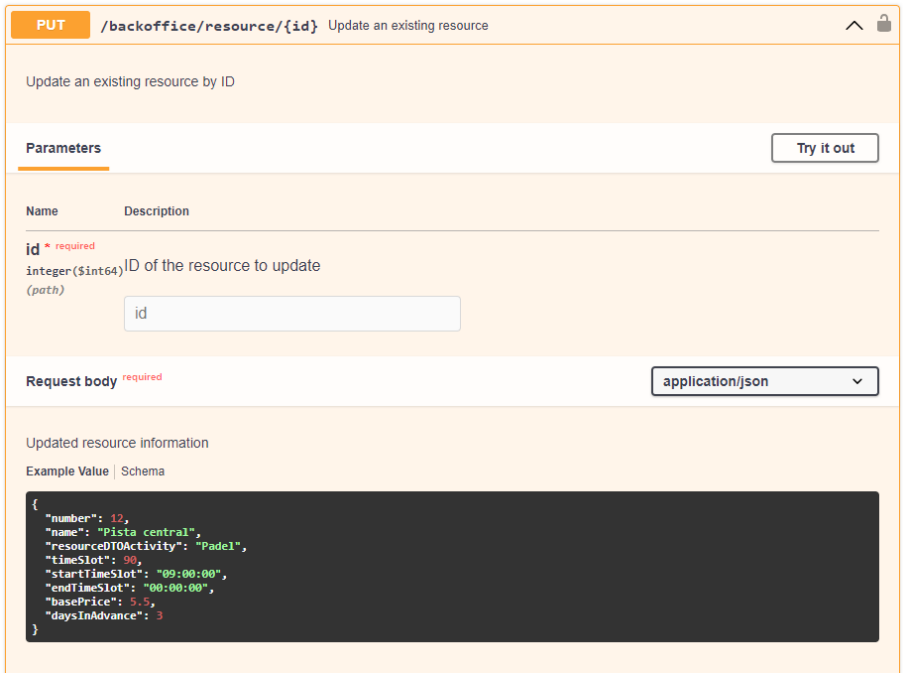

Figura A.3: Ejemplo de entrada para la modificación de un recurso

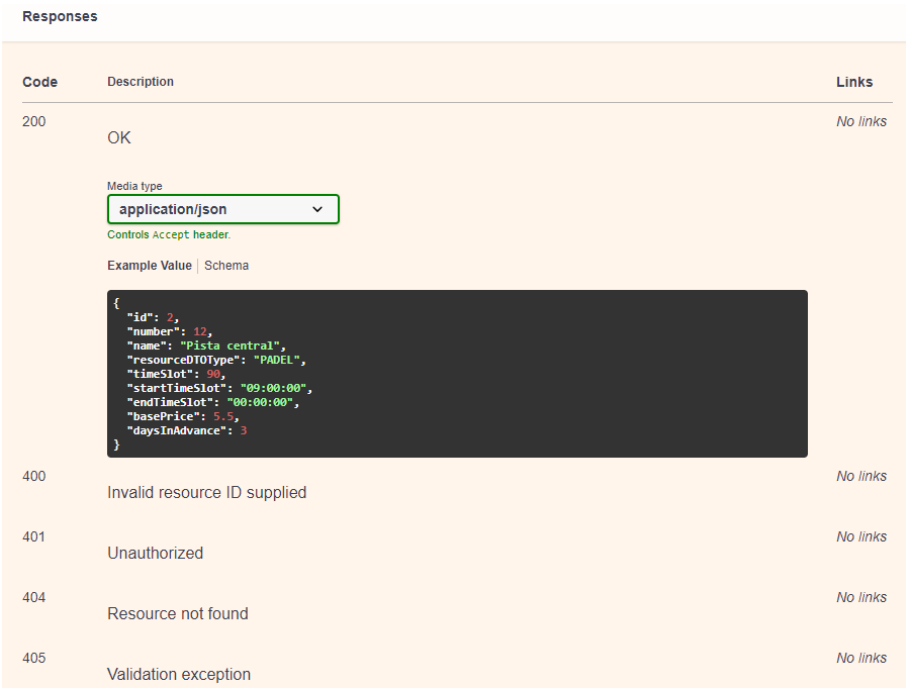

Figura A.4: Ejemplo de respuesta para la modificación de un recurso

4. Realizar llamadas a los endpoints: A través de la interfaz del Swagger Editor, los usuarios pueden realizar llamadas a los endpoints de la API y ver las respuestas correspondientes. Esto les permite probar la funcionalidad de la API y verificar que está respondiendo correctamente.

   En el contexto de un sistema de autenticación, el uso de un token de acceso es fundamental para garantizar la seguridad y autorización adecuada de las solicitudes al backend. Este token de acceso se obtiene mediante el proceso de inicio de sesión o creación de un usuario en el sistema.

   Con el token de acceso en mano, el cliente (frontend) puede incluirlo en el encabezado de autorización de cada solicitud subsiguiente como un token Bearer, haciendo click en el candado de la petición. Esto permite al servidor verificar la identidad del usuario y autorizar las solicitudes en función de los permisos asignados al usuario.

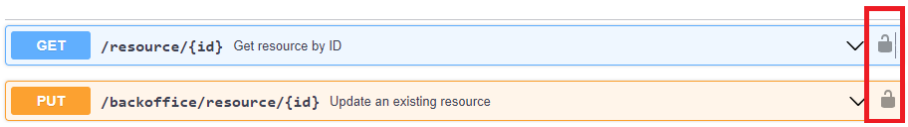

Figura A.5: End-points que necesitan autorización

Para acceder a los endpoints de administración (backoffice), es necesario contar con un

rol de administrador. Esto implica que el usuario autenticado debe tener los privilegios correspondientes para acceder y realizar operaciones en esas partes del sistema.

A.2. Uso del boot

El uso del bot de Telegram es muy sencillo y sigue la lógica de negocio que utilizan los usuarios en el grupo de Telegram. Para acceder al bot, simplemente se busca su nombre de usuario, que es @PadelMachBot. El bot de Telegram ofrece un menú con varias opciones para que el usuario pueda interactuar con él de manera conveniente.

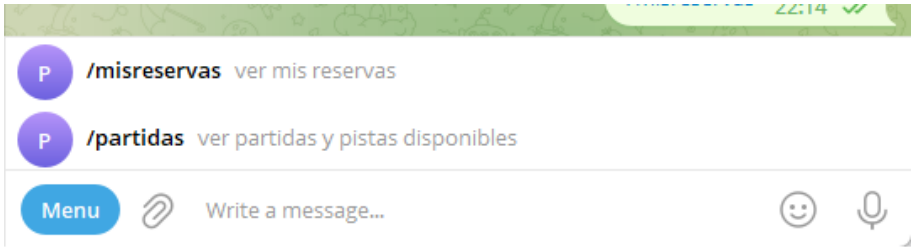

Figura A.6: Despliegue del menú del bot

La interfaz del bot está diseñada de manera intuitiva, guiando al usuario a través de las opciones disponibles. El bot presenta al usuario diferentes opciones para elegir, y el usuario puede seleccionar la opción deseada simplemente con una pulsación.

Una característica destacada del bot es que no permite retroceder en las elecciones realizadas. Esto significa que una vez que el usuario ha seleccionado una opción, no puede cambiar de opinión o retroceder en el flujo de interacción. Esto garantiza que el usuario siempre esté guiado y las respuestas mostradas por el bot correspondan a las elecciones realizadas por el usuario.

Figura A.7: Captura empleando el bot de Telegram

Apéndice B

Contenidos del CD-ROM

Los contenidos que acompañan a la memoria entregada son los siguientes:

- `MemoriaTFGManuelComesanaCouto.pdf`: copia de la memoria en formato PDF.
- `padelapp`: código fuente de la API.
- `Images`: imágenes utilizadas en la memoria del proyecto.
- `apipadel.yaml`: definición de la API.

Bibliografía

[1] cmdsport. «Padel se erige como el deporte de mayor crecimiento a nivel mundial.» (2020), dirección: https://www.cmdsport.com/esencial/cmd-raqueta/padel-se-erige-deporte-mayor-crecimiento-nivel-mundial/.

[2] Playtomic. «Condiciones generales - Playtomic Blog.» (2022), dirección: https://blog.playtomic.io/condiciones-generales/.

[3] «Spring Framework.» (2022), dirección: https://es.wikipedia.org/wiki/Spring_Framework.

[4] «OpenAPI Specification 3.0.3.» (2022), dirección: https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.3.md.

[5] Trello. «Trello - utilidades.» (2020), dirección: https://trello.com/tour.

[6] MySQL. «MySQL Documentation.» (2022), dirección: https://dev.mysql.com/doc/.

[7] MySQL. «MySQL Workbench Documentation.» (2022), dirección: https://dev.mysql.com/doc/workbench/.

[8] Telegram. «Telegram.» (2022), dirección: https://telegram.org/.

[9] A. Friends. «Características de XAMPP.» (2022), dirección: https://www.apachefriends.org/es/features.html.

[10] A. IT. «¿Qué es Postman?» (Fecha de publicación), dirección: https://www.arquitectoit.com/postman/que-es-postman/.

[11] JetBrains. «IntelliJ IDEA.» (Fecha de publicación), dirección: https://www.jetbrains.com/idea/.

[12] «Scrum - Atlassian.» (), dirección: https://www.atlassian.com/es/agile/scrum.

[13] «¿Qué es Scrum? - Scrum.org.» (), dirección: https://www.scrum.org/resources/blog/que-es-scrum.

[14] Tecnova. «DDD (Domain-Driven Design).» (2021), dirección: https://www.tecnova.cl/2021/06/23/ddd-domain-driven-design/.

[15] Swagger. «Adopting an API-First Approach.» (Fecha de publicación), dirección: https://swagger.io/resources/articles/adopting-an-api-first-approach/.

[16] talent. «Salario de ingeniero de software.» (2020), dirección: https://es.talent.com/salary?job=ingeniero+de+software.

[17] R. Guru. «Builder.» (Fecha de publicación), dirección: https://refactoring.guru/es/design-patterns/builder.

[18] «State.» (), dirección: https://refactoring.guru/es/design-patterns/state.

[19] «Template Method.» (), dirección: https://refactoring.guru/es/design-patterns/template-method/java/example.

[20] R. Guru. «Observer.» (Fecha de publicación), dirección: https://refactoring.guru/es/design-patterns/observer.

[21] «OpenAPI Generator - GitHub.» (), dirección: https://github.com/OpenAPITools/openapi-generator/tree/master/modules/openapi-generator-maven-plugin.

[22] «SLF4J - Simple Logging Facade for Java.» (2015), dirección: https://www.adictosaltrabajo.com/2013/09/10/slf-4j/.