ORDOÑEZ PINEDA, JUAN MIGUEL

IES MACIA ABELA

TRABAJO DE FIN DE GRADO 2DAM

VIDEOJUEGO FIGHTBALL

VIDEOJUEGO Y APLICACIONES AUXILIARES

Logotipo

El contenido generado por IA puede ser incorrecto.

Tabla de contenido

[RESUMEN 3](#_Toc199089566)

[ABSTRACT 4](#_Toc199089567)

[INTRODUCCIÓN 5](#_Toc199089568)

[REQUISITOS 6](#_Toc199089569)

[Desarrollo de Interfaces 6](#_Toc199089570)

[Programación Multimedia y Dispositivos Móviles 7](#_Toc199089571)

[Inglés Técnico 7](#_Toc199089572)

[Programación de Servicios y Procesos 7](#_Toc199089573)

[Acceso a Datos 8](#_Toc199089574)

[OBJETIVOS 9](#_Toc199089575)

[OBJETIVOS GENERALES 9](#_Toc199089576)

[OBJETIVOS ESPECIFICOS 9](#_Toc199089577)

[EXPLICACION GENERAL DE LOS NODOS 10](#_Toc199089578)

[Esquema General 10](#_Toc199089579)

[Esquema y explicación acerca del funcionamiento de los chats 11](#_Toc199089580)

[Creación de Usuarios, Gestión de roles y usuarios 12](#_Toc199089581)

[Base de Datos, Nginx y Despliegue Online 13](#_Toc199089582)

[FUNCIONAMIENTO ESPECIFICO Y CAPTURAS 14](#_Toc199089583)

[Servidor Spring: 14](#_Toc199089584)

[Explicación general 14](#_Toc199089585)

[Funcionamiento Interno General 17](#_Toc199089586)

[Paquete Repository 18](#_Toc199089587)

[Paquete Service 19](#_Toc199089588)

[Paquete Controller 20](#_Toc199089589)

[Paquete config 21](#_Toc199089590)

[Dependencias Usadas en El Servidor: 22](#_Toc199089591)

[Aplicación movil: 23](#_Toc199089592)

[Videojuego: 23](#_Toc199089593)

[Aplicación Auxiliar De Escritorio: 23](#_Toc199089594)

[CONCLUSION 24](#_Toc199089595)

[REFERENCIAS BIBLIOGRAFICAS 25](#_Toc199089596)

# RESUMEN

FightBall, es un videojuego de combate, que funciona mediante el motor de videojuegos Godot, teniendo la capacidad de funcionar tanto de manera local, como accediendo a un servidor Spring mediante el cual recibirá peticiones y contestará permitiendo al videojuego interactuar, y acceder a los datos de este, permitiéndole tanto como almacenar datos del servidor y enviarle datos a este.

Para complementar al videojuego, y cumplir los requisitos pedidos, se ha diseñado una aplicación móvil que nos permita consultar los datos del servidor, permitiéndonos acceder a la información del servidor mediante roles de Administrador, Moderador, Jugador.

Finalmente para darle un enfoque desde escritorio, se ha creado una aplicación de escrito auxiliar sencilla, que permita ver información básica, y permita exportarla a archivos como PDF, Word o Txt.

**Palabras clave:**  
Godot, FightBall, Servidor Spring, Aplicación móvil, Aplicación de escritorio, Roles, Multijugador, Exportación de datos, Administrador, Moderador, Jugador.

# ABSTRACT

*FightBall* is a combat video game developed using the Godot engine, designed to operate both locally and in connection with a Spring-based server. This server handles requests and responses, enabling the game to interact with stored data, allowing for both data retrieval and submission. To complement the video game and meet the required specifications, a mobile application has been developed. This application provides users with access to server information based on their assigned roles—Administrator, Moderator, or Player. Additionally, a simple desktop utility has been created to offer a basic overview of server data and support export options in formats such as PDF, Word, or TXT.

**Keywords**  
Godot, FightBall, Spring Server, Mobile Application, Desktop Application, Roles, Multiplayer, Data Export, Administrator, Moderator, Player.

# INTRODUCCIÓN

En el siguiente Documento explicaremos a detalle al usuario, tanto el funcionamiento de toda la suite como los procesos, clases principales, y detalles más específicos de cada una de ella.

Comenzando con un esquema general de toda la suite, como se comunica entre sí, los datos que circulan en ella, las posibilidades que ofrecen cada nodo(aplicación) y la función que cumplen dentro de la suite, aclarando conceptos fundamentales para la función de cada nodo y de este modo que el usuario pueda comprender su uso, e incluso realizar modificaciones dentro del repositorio de GitHub sin ningún tipo de complicación. A continuación, se hablará más específicamente de cada nodo, mostrando un esquema de sus pantallas, su funcionamiento interno, componentes claves para que funcione, comentando la documentación y las partes mas importantes de esta para tener en cuenta y finalizar con un manual para el usuario para el correcto uso del nodo.

Siguiendo con el núcleo de la aplicación que es el servidor, explicando su funcionamiento, que tecnología usa, como funciona y que nos ofrece, gestión de permisos, endpoints, servicios y demás clases que permiten el funcionamiento de este, se comentara acerca de la documentación, haciendo referencia a las clases y métodos más importantes, y partes claves para el funcionamiento del servidor, finalmente como antes mencionado tendremos un manual de usuario para poder acceder correctamente a los endpoints y el uso de estos.  
  
A continuación se hablara acerca del Videojuego ,siendo este un nodo cliente, es un videojuego sencillo diseño que nos permite el acceso a nuestra base de datos y generar datos para ella, de igual manera se explicará el funcionamiento de este, componentes claves y esquema de pantallas que maneja, a parte de una explicación básica del motor de videojuegos Godot, para mejor entendimiento del usuario, siguiendo con su documentación principal , y un manual para el usuario para poder sacar el máximo partido del videojuego.

Seguidamente contaremos con las dos aplicaciones Auxiliares, la aplicación Android y la de escritorio, cumpliendo funciones similares, con la diferencia que la aplicación de escritorio solo permitirá lectura de datos, sin contar el inicio de sesión, permitiendo la aplicación Android generar información que será visible en todos los nodos, se mostrará un esquema del funcionamiento general , documentación y tambien se manual de usuario.

# REQUISITOS

## Desarrollo de Interfaces

**☐** Interfaz gráfica con atributos de usabilidad: eficacia, eficiencia, accesibilidad, ToolTips, accesos por teclado, control de acceso, ayuda/manuales.

**☐** Uso de contenedores y componentes: JFrame, JDialog, JLabel, JTextField, JButton, JRadioButton, JComboBox, JList, JPasswordField.

**☐** Uso de layout managers: BorderLayout, FlowLayout, GridLayout, CardLayout, etc.

**☐** Gestión de eventos: ActionListener, ChangeListener, etc.

**☐** Uso de cuadros de diálogo: JFileChooser, JOptionPane, etc.

**☐** Componentes personalizados: ListCellRenderers (JList, JComboBox).

**☐** Aceleradores de menú: JMenuBar, JMenu, JMenuItem.

**☐** Pantalla de carga durante operaciones (ej. conexión a BD).

**☐** Soporte para internacionalización (i18n).

**☐** Código documentado con JavaDoc.

**☐** Manual de usuario accesible desde aplicación (web, PDF o embebido).

**☐** Mecanismo de instalación incluido.

## Programación Multimedia y Dispositivos Móviles

**☐** Aplicación móvil con varias pantallas e interfaz con listas, botones, textos, spinners, checkboxes, radiobuttons.

**☐** Notificaciones simples o expandidas, diálogos de confirmación, toasts personalizados.

**☐** Soporte para acciones: abrir navegador, enviar correo, compartir información, llamadas, etc.

**☐** Menús contextuales o globales en la barra superior.

**☐** Pantalla o menú de preferencias usando sharedPreferences.

**☐** Soporte para internacionalización (mínimo 2 idiomas).

**☐** Pantalla 'Acerca de' con autor y contenido multimedia (audio o video).

**☐** Gestión de permisos en tiempo real si es necesario.

**☐** Acceso a base de datos en la nube (AWS) mediante API REST.

**☐** CRUD completo sobre datos de la API.

**☐** Estructura ordenada del código y recursos.

**☐** Manual de uso con capturas y explicación.

## Inglés Técnico

**☐** Redacción de un abstract en inglés (100-200 palabras).

**☐** Incluir contexto, metodología, resultados y conclusiones en el abstract.

**☐** Comparar título con contenido, identificar secciones clave.

**☐** Utilizar vocabulario técnico adecuado y estructura académica clara.

## Programación de Servicios y Procesos

**☐** Aplicación multihilo para mejorar rendimiento.

**☐** Hilos en segundo plano para tareas pesadas.

**☐** Subida de archivo cifrado (simétrico o asimétrico) a servidor FTP.

## Acceso a Datos

☐ Desarrollo de una API REST con Spring Boot.  
☐ Acceso a base de datos (MariaDB, MySQL, PostgreSQL, etc.).  
☐ Estructura por capas para una separación clara de componentes.  
☐ Uso de DTOs y mapeadores para la comunicación cliente-servidor.  
☐ Validación de DTOs y entidades.  
☐ Manejador de excepciones global y excepciones personalizadas.  
☐ Registro de errores y eventos mediante sistema de logs.  
☐ Consultas a la base de datos implementadas mediante JPA.  
☐ Documentación de EndPoints con Swagger.  
☐ Autenticación y autorización de usuarios, con uso de roles.  
☐ Despliegue de la API y la BBDD en AWS.  
☐ Uso de Flyway para la gestión de la base de datos.  
☐ Investigación y desarrollo de pruebas (tests) o WebSockets.  
  ☐ Si se elige **Test**, implementación de pruebas para EndPoints y servicios.  
  ☐ Si se elige **WebSocket**, implementación de al menos 2 EndPoints funcionales.  
☐ Repositorio del proyecto alojado en GitHub.

# OBJETIVOS

### OBJETIVOS GENERALES

1. Diseñar una suite que se pueda conectar entre sí, permitiendo un flujo de datos correcto, seguro y claro, además mediante esta poder cumplir con los requisitos antes especificados.
2. Ampliar el conocimiento en las herramientas que son usadas para el desarrollo del proyecto permitiendo asi ganar mas experiencia y tener mas agilidad en el desarrollo de futuros proyectos, optimizándolos y pudiendo tener un mayor control de las distintas tecnologías.
3. Crear un proyecto algo de diferente a lo normalmente presentado no limitándonos solamente a una aplicación móvil si no, intentando explotar el máximo potencial y buscar usos distintos de nuestro servidor

### OBJETIVOS ESPECIFICOS

1. Desarrollar el servidor evitando al máximo errores indeseados y codigo que no sea claro paro para futuros programadores que quisiesen modificar o implementar sus ideas al proyecto.
2. Llevar a cabo el desarrollo de un videojuego, que funcione de manera optima y tenga un minimo de atractivo estéticamente, el desarrollo se llevara a cabo con el motor de videojuegos Godot, permitiéndonos adaptar a un nuevo entorno.
3. Desarrollar un aplicación móvil, que permita tanto la visualización como edición de datos del servidor, además que sea estáticamente atractiva e intuitiva a la hora de su uso
4. Construir una aplicación de escritorio, cuyo objetivo principal será la exportación de datos del servidor con otros formatos para su posterior uso, formatos como Word o PDF.

# EXPLICACION GENERAL DE LOS NODOS

### Esquema General

Para comenzar la explicación general de la suite usaré el siguiente esquema

Diagrama

El contenido generado por IA puede ser incorrecto.

Como se puede apreciar en el esquema nuestro servidor es el núcleo de toda la suite, de modo que de cierto modo el flujo de datos afecta a toda la suite, por ejemplo:

Empezando por el videojuego un usuario juega una partida, la partida se envía al servidor, este la recibe, seguidamente el mismo usuario va a su dispositivo móvil, y podrá ver su partida, el recuento total de ella, si ha salido victorioso o ha perdido, el mismo usuario tendrá la posibilidad de exportar su partida e imprimirla si asi lo desea, si un administrador decide añadir nuevos enemigos desde la aplicación en modo administrador, este podrá hacerlo , como consecuencia el usuario al jugar el modo oleadas del videojuego vera los enemigos implementados por el administrador, estos mismos enemigos podrán ser exportados desde la aplicación de escritorio, si el usuario quisiera tenerlos en un documento aparte para mejor visualización.

Como podemos ver el funcionamiento de la suite consistirá constantemente de un intercambio de información basado en, realizar petición y luego recibir una respuesta de esta por parte del servidor, salvo una excepción.

### Esquema y explicación acerca del funcionamiento de los chats

Si prestamos atención al esquema anteriormente mostrado podemos ver cómo tanto el videojuego y la aplicación envían chats, con chats nos referimos a mensajes en tiempos real, que a diferencia del demás flujo de información este realiza una conexión en tiempo real por medio de WebSocket, del cual hablaremos mas a profundidad mas adelante, de este modo permitiendo a los demás clientes de la sesión ver los mensajes de los otros clientes en tiempo real, como se puede ver en el siguiente esquema.

Diagrama

El contenido generado por IA puede ser incorrecto.

Como podemos ver es el servidor quien permite la creación de una conexión a una sesión, para que asi los usuarios puedan recibir los demás mensajes que sean enviados por los demás usuarios sin necesidad de estar constantemente realizando peticiones para ver si ha habido mensajes nuevos, en cambio pudiendo recibirlos en tiempo real sin mayor complicación.

### Creación de Usuarios, Gestión de roles y usuarios

Como podemos ver continuamente podemos realizar peticiones y manipular datos, realizar cambios y crear datos hacia nuestro servidor desde los distintos nodos, pero para esto a aparte de una configuración lógica pertinente para que esto funcione correctamente tambien necesitaremos configurar permisos y roles, estos están configurados dentro del servidor Spring, de modo que según con que usuario hayamos entrado y que permisos tenga este nos permitirá realizar distintas acciones, acceder a un tipo determinado de información.

En esta aplicación poseemos tres tipos de Roles que los usuarios pueden tener, el uso, aplicación y gestión de estos roles será principalmente notorio en la aplicación son los siguientes:

* **Jugador**: Es el rol más básico, permite al usuario generar partidas por medio de jugarlas en el videojuego, habiendo iniciado sesión, permite solamente la lectura de todo tipo de entidades de la base de datos, salvo ver los demás usuarios y partidas de los otros.
* **Moderador**: rol un poco más avanzado, bastante parecido al jugador, con la diferencia de que este permite ver las partidas de todos los jugadores y ver todos los usuarios registrados, mas tampoco puede crear entidades ni modificarlas, salvo partidas generadas si tiene rol de jugador.
* **Administrador**: el Rol con la mayor cantidad de permisos, puede ver cualquier entidad de la base de datos, y tiene permisos para poder modificar estas y eliminarlas.

### Base de Datos, Nginx y Despliegue Online

Finalmente, y antes de pasar a explicar cada nodo en detalle, parece importante hacerse otra pregunta, constantemente se ha hecho mención de que el servidor funciona como núcleo principal de la suite, y que dicho servidor usa una base de datos, pero ¿dónde se encuentra este o cómo funciona?

En el desarrollo normalmente el servidor se despliega en local, permitiéndonos ir haciendo pruebas en nuestro propio equipo, en nuestro caso ejecutando el programa con nuestro **IDE** (entorno de desarrollo), y un servidor de maría DB, simulando la arquitectura cliente-servidor en nuestro propio equipo.

Sin embargo, para el despliegue online esto no sirve, de este modo pues hemos accedido a diversas tecnologías que nos permitirán el despliegue y funcionamiento en línea de nuestro servidor:  
**Amazon Web Service** Y **Nginx**.

**AWS**, que significa **Amazon Web Services**, es una plataforma de servicios en la nube ofrecida por Amazon. Permite a las empresas y desarrolladores crear y ejecutar aplicaciones y servicios en la nube de manera escalable, segura y confiable. AWS ofrece una amplia gama de servicios, desde computación e almacenamiento hasta bases de datos, análisis y más.

 Esta nos permitirá crear un Relational Database Service (**RDS**), para tener nuestra base de datos en la nube y con **EC2**, que nos permite desplegar un equipo virtual **Ubuntu** en remoto y este será el encargado de alojar nuestro servidor, el cual será manejado mediante **Nginx**.

**Nginx**, pronunciado en inglés como “engine-ex”, es un famoso software de servidor web de código abierto. En su versión inicial, funcionaba en servidores web HTTP. Sin embargo, hoy en día también sirve como proxy inverso, balanceador de carga HTTP y proxy de **correo** electrónico para IMAP, POP3 y SMTP.

# FUNCIONAMIENTO ESPECIFICO Y CAPTURAS

## Servidor Spring:

### Explicación general

Para comenzar con la explicación acerca de nuestro servidor, primeramente, hablaremos de los siguientes conceptos claves, seguidos de una breve explicación para asi poder comprender más a fondo el funcionamiento del servidor :

* **API-REST:** Una API de REST es una interfaz de programación de aplicaciones (API) que sigue los principios de diseño del estilo de la arquitectura REST. REST significa transferencia de estado representacional y consiste en un conjunto de reglas y recomendaciones para diseñar una API web.

REST no es un protocolo ni un estándar, sino más bien un conjunto de límites relacionados con la arquitectura. Los desarrolladores de las API pueden implementarlo de distintas maneras.

Cuando el cliente envía una solicitud a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información se entrega por medio de peticiones HTTP en un formato determinado, **JSON** (JavaScript Object Notation),es el formato de archivo más conocido, ya que tanto las máquinas como las personas pueden comprenderlo, y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.



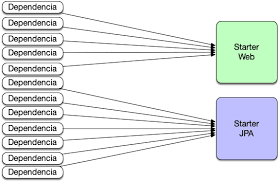
* **Petición HTTP**: Las solicitudes HTTP son mensajes enviados por el cliente al servidor para iniciar una acción específica. Tienen una estructura específica y se caracterizan por un método (o verbo) que permite distinguir el tipo de acción requerida.

Los métodos de solicitud HTTP identifican la acción que se desea realizar en el recurso. Aunque puedan ser nombres, normalmente suelen denominarse *verbos HTTP*, precisamente porque representan la acción a realizar. Los métodos principales son los siguientes:

* + **Get**: Solicita el recurso indicado, que el servidor proporciona como body de la respuesta.
  + **Post**: Envía datos al servidor usando el recurso adecuado.
  + **Put**: Crea un nuevo recurso o reemplaza el recurso indicado con el body de la solicitud.
  + **Delete**: Elimina el recurso indicado.

Dichas Peticiones contienen **cabeceras(Headers),**que permiten tanto al cliente como al servidor enviar información adicional a la petición.

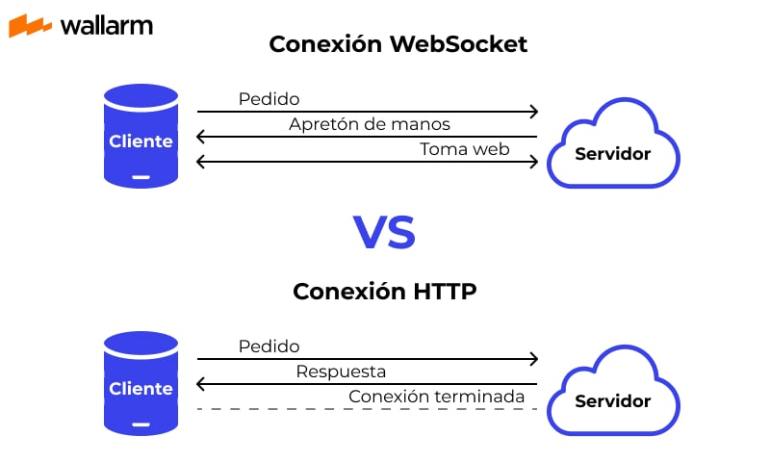
* **Java Spring Boot:** Java Spring Boot (Spring Boot) es una herramienta que acelera y simplifica el desarrollo de microservicios y aplicaciones web con Spring Framework, debido a que tradicionalmente para configurar manualmente todo el servidor incluyendo todas las dependencias necesarias y ahorrando gran parte del trabajo necesario para desplegar nuestro servidor manualmente.



Como podemos ver en la imagen, Srpring boot, nos incluye todas las dependencias en un starter package, que se adapte a nuestras necesidades.

* **Websocket:** es un protocolo de red basado en TCP que establece cómo deben intercambiarse datos entre redes. Puesto que es un protocolo fiable y eficiente, es utilizado por prácticamente todos los clientes. El protocolo TCP establece conexiones entre dos **puntos finales de comunicación**, llamados ***sockets***. De esta manera, el intercambio de datos puede producirse en las dos direcciones en tiempo real.

A Diferencia de HTTP, este protocolo nos permite tener comunicación en tiempo real, basta con que el cliente establezca una conexión con el servidor, que se confirma mediante el llamado *apretón de manos* o *WebSocket Protocol Handshake*. Con él, el cliente envía al servidor todos los datos de identificación necesarios para el intercambio de información.



Teniendo claro los conceptos anteriores, que son claves para entender el funcionamiento de nuestro servidor, debido a que nuestro servidor funciona mediante **Spring Boot,** permitiéndonos crear asi nuestra **API REST** a la cual accederán los clientes, ya sea tanto mediante **Peticiones HTTP**  o bien accediendo mediante **WebSocket**.

Con eso claro ya podremos seguir con una explicación mas a detalle del funcionamiento interno de nuestro servidor.

### Funcionamiento Interno General

Comenzaremos viendo la estructura de las clases que estamos usando dentro del proyecto, este sigue el modelo, **MVC** (Modelo-Vista-Controlador) es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización. De modo que sea mas sencillo el desarrollo:

Texto

El contenido generado por IA puede ser incorrecto.

Viendo las estructuras vemos que se ha dividido en distintos paquetes, que resumiremos de la siguiente manera:

1. **config**: Contiene clases de configuración de Spring (seguridad, beans, etc.)
2. **controller**: Las clases Controlador que manejan las peticiones HTTP (REST endpoints)
3. **dto**: Objetos de Transferencia de Datos (Data Transfer Objects) para la comunicación cliente-servidor
4. **Entity**: Las entidades del modelo de datos (mapeo a tablas de base de datos)
5. **Enums**: Tipos enumerados usados en la aplicación
6. **exceptions**: Manejo personalizado de excepciones
7. **mapper**: Clases para conversión entre Entidades y DTOs
8. **Message**: Manejo de mensajes/i18n para el websocket
9. **repository**: Interfaces de Spring Data JPA para acceso a datos
10. **service**: Lógica de negocio (interfaces e implementaciones)

Entre todos estos paquetes los que miraremos más a fondo serán los siguientes:

* Repository
* Services
* Controller
* Config

Miraremos a profundo estos tomando con ejemplo la entidad Match, estos paquetes sirven principalmente como auxiliares, por ejemplo, manejando las excepciones que pueda producir nuestro servidor, mapeando las entidades según nuestra necesidad, o demás funciones, vamos a mirar a profundidad los paquetes que nos permiten manejar los datos en el servidor y dar acceso a los clientes:

### Paquete Repository

Pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Dentro de este tenemos los distintos interfaces, que extienden de JpaRepository, permitiéndonos acceder a los distintos métodos CRUD acrónimo de Create (Crear), Read (Leer), Update (Actualizar) y Delete (Borrar), además permitiéndonos la creación de Querys Personalizadas según nuestra necesidad.

### Paquete Service

Patrón de fondo

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Este paquete contendrá dos componentes, una clase y un interfaz, los cuales nos permitirán acceder a los métodos del repositorio previamente creado, pudiendo acceder así a la información de nuestra base de datos, aquí implementamos toda la lógica necesaria que vayamos a usar respecto al manejo de datos.

### Paquete Controller

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Finalmente, dentro del paquete controlador tenemos las clases controladores, cuya función es crear el endpoint a donde el cliente accederá, por ejemplo, en este caso si el cliente quisiera ver todas las peticiones tendrá que realizar la siguiente petición con el método **GET**, que es el tipo indicado con la anotación **@GetMapping**

54.164.115.171/api/matches

Recibiendo asi una lista de partidas DTO, que son las partidas que llama el método al servicio y una vez obtenidas las transforma, dando al usuario solamente la información que necesitemos, en el caso de que algo saliera mal devolverá un error.

### Paquete config

El paquete config contiene distintas clases que permiten configurar distintas utilidades, pero vamos a centrarnos en Security Config,

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Esta clase es la encargada de controlar que usuarios tienen permisos y pueden acceder a los endpoints que especifiquemos:

Texto

El contenido generado por IA puede ser incorrecto.

Como podemos observar mediante requestMatchers, podemos indicar los permisos requeridos para cada endpoint, indicándole mediante roles, permitiendo a los usuarios acceso o negárselo según la necesitemos.

### Dependencias Usadas en El Servidor:

Para el buen funcionamiento del servidor, no es suficiente solamente con Java, tenemos que añadir distintas dependencias que nos permitan llevar a cabo las modificaciones, por lo tanto en la siguiente lista se describirá el funcionamiento de cada una de ellas:

1. **spring-boot-starter-data-jpa** - Integración con JPA para acceso a bases de datos relacionales
2. **spring-boot-starter-data-rest** - Exponer repositorios JPA como servicios REST automáticamente
3. **spring-boot-starter-web** - Soporte para desarrollo de aplicaciones web MVC
4. **spring-boot-devtools** - Herramientas de desarrollo para reinicio automático
5. **mariadb-java-client** - Controlador JDBC para MariaDB
6. **spring-boot-starter-test** - Utilidades para testing (JUnit, Mockito, etc.)
7. **lombok** - Generación de código boilerplate (getters, setters, constructores)
8. **flyway-core/flyway-mysql** - Herramienta de migraciones de base de datos
9. **modelmapper** - Mapeo automático entre objetos DTO y Entidades
10. **spring-boot-starter-validation** - Validación de datos en endpoints
11. **springdoc-openapi** - Documentación automática de API (Swagger/OpenAPI)
12. **spring-boot-starter-security** - Seguridad y autenticación
13. **jjwt (Java JWT)** - Creación y validación de tokens JWT
14. **spring-boot-starter-websocket** - Soporte para comunicación WebSocket

## Aplicación movil:

### Explicación General

Primeramente, para entender el funcionamiento de nuestra aplicación móvil, mostrare un pequeño esquema que permita entender la el flujo de esta para luego explicar un par de conceptos claves para entender el funcionamiento y herramientas que usa:

### Videojuego:

Adasdadda

### Aplicación Auxiliar De Escritorio:

Adasdadda

## CONCLUSION

Blalbalblabllablba

# REFERENCIAS BIBLIOGRAFICAS

B, G., & B, G. (2025, 13 marzo). *¿Qué es NGINX y cómo funciona?* ES Tutoriales. <https://www.hostinger.com/es/tutoriales/que-es-nginx>

*what-is-aws*. (s. f.). [Vídeo]. Amazon Web Services, Inc. <https://aws.amazon.com/es/what-is-aws/>

*¿Qué es una API de REST?* (s. f.). <https://www.redhat.com/es/topics/api/what-is-a-rest-api>

*¿Qué son las solicitudes HTTP?* (s. f.). aulab.es. <https://aulab.es/articulos-guias-avanzadas/108/que-son-las-solicitudes-http>

*HTTP headers - HTTP | MDN*. (2025, 26 marzo). MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Headers>

Ibm. (2023, 17 julio). *¿Qué es Java Spring Boot? | IBM*. <https://www.ibm.com/es-es/topics/java-spring-boot>

Caules, C. Á. (2025, 3 marzo). *Spring Boot ¿Qué es y cómo funciona?* Arquitectura Java. <https://www.arquitecturajava.com/spring-boot-que-es/>

Equipo editorial de IONOS. (2020, 7 agosto). *¿Qué es WebSocket?* IONOS Digital Guide. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-websocket/>

Swagger-Api. (s. f.). Swagger 2.X annotations. GitHub. <https://github.com/swagger-api/swagger-core/wiki/Swagger-2.X---Annotations>