Institut Puig Castellar
Santa Coloma de Gramenet

TiliManager

Projecte de desenvolupament

CFGS Desenvolupament d'Aplicacions Web

Pau Ojeda Gallego

2 DAW B

Resumen del proyecto

TiliManager es una aplicación web y móvil (multiplataforma) de gestión de un equipo de fútbol (estilo *Fantasy*), pero, con la contraparte de que no son jugadores de fútbol reales, sino que estos jugadores están basados en los personajes de Inazuma Eleven. La aplicación trata sobre competir contra otros jugadores en línea, donde los usuarios asumen el rol de entrenadores para gestionar presupuestos, fichajes en tiempo real y tácticas de equipo dentro de diferentes ligas.

El objetivo del proyecto es ofrecer una plataforma divertida y apta para todo usuario. El sistema impone reglas para garantizar el equilibrio entre usuarios, controlando económicamente que los balances no queden en negativo y limitando el tamaño de las plantillas en un rango de entre 5 y 7 jugadores. La metodología se basa en un desarrollo desacoplado. El frontend se ha construido con Angular, Ionic y Tailwind CSS, empleando Signals para una buena reactividad e interfaces de Drag and Drop. El backend se ha desarrollado con Java, Spring Boot y PostgreSQL, aplicando seguridad en el cifrado de contraseñas y validación de datos mediante DTOs.

En conclusión, este proyecto ha permitido llevar a cabo tanto una aplicación multiplataforma (tanto web como aplicación móvil), separando el proyecto en una buena página reactiva y fácil de manejar (Frontend), con una buena lógica, con validación para cada operación en la aplicación (Backend).

Palabras clave

Inazuma Eleven, Juegos Gratuitos Inazuma, Inazuma Eleven Manager, Gestor de ligas Inazuma, Partidos Inazuma, Juego Fantasy Inazuma

Abstract

TiliManager is a web and mobile application (multiplatform) for managing a football team (Fantasy style). However, instead of real-world athletes, the players are based on the iconic characters from Inazuma Eleven. The app is about competing against other players online, where users take on the role of coaches to manage budgets, real-time signings and team tactics within different leagues.

The objective of the project is to offer a fun platform suitable for all users. The system imposes rules to guarantee balance between users, economically controlling that the balances do not become negative and limiting the size of the squads to a range of between 5 and 7 players. The methodology is based on decoupled development. The frontend has been built with Angular, Ionic and Tailwind CSS, using Signals for good reactivity and Drag and Drop interfaces. The backend has been developed with Java, Spring Boot and PostgreSQL, applying security in password encryption and data validation through DTOs.

In conclusion, this project has allowed us to carry out both a multi-platform application (both web and mobile application), separating the project into a good reactive and easy-to-manage page (Frontend), and a good logic, with validation for each operation in the application (Backend).

Keywords

Inazuma Eleven, Free Inazuma Games, Inazuma Eleven Manager, Inazuma League Manager, Inazuma Matches, Inazuma Fantasy Game

Índice

Lista de figuras

(Los dos primeros son los Logos del proyecto, el tercero es las vista de la carta para poner jugadores y sus estadísticas, y los últimos dos de abajo, son svg, creados para cuando algo no se encuentra, y para un botón de ayuda respectivamente)

1 Introducción

Este proyecto consiste en el desarrollo de una aplicación multiplataforma con un Backend donde quedarán todos los datos, y donde estará el propio código del juego, que se integrará con un Frontend, donde todos estos datos serán vistos de una manera muy vistosa, y de una manera reactiva para que los usuarios vean a tiempo real lo que están haciendo. El Backend contactará con una base de datos para gestionar usuarios, puntuaciones, estadísticas, equipos, y las ligas. El objetivo es entregar un producto funcional (aplicación tanto web como móvil jugable), demostrando conocimientos de programación en Java, desarrollo web, diseño de bases de datos y aprender a usar Angular.

La propuesta:

Un Backend REST en Spring Boot para autenticación (Spring Security), y crear todo el código y lógica del proyecto.

Una Base de datos relacional (PostgreSQL) que almacena todos los datos, usando JPA Hibernate y librerías de Spring para que se maneje todo desde el Backend.

Una Frontend con Angular e Ionic que permita usar todas las API que hayan en el Backend.

1.1 Contexto

En el mercado actual existen muchos juegos comerciales de gestión de fútbol, como La Liga *Fantasy* u *Online Soccer Manager*, pero ninguno cuenta con los personajes de la saga Inazuma Eleven. Este proyecto nace para cubrir ese hueco, uniendo las mecánicas competitivas de un juego estilo *Fantasy* con el universo de dicha serie de animación.

Para el desarrollo se han elegido tecnologías maduras en el sector. En el servidor (Backend), Java y Spring Boot permiten crear una API REST sólida y segura para gestionar la lógica. En el cliente (Frontend), Angular e Ionic ofrecen una solución moderna para crear una interfaz reactiva que funcione correctamente tanto en entorno web como en aplicación móvil.

1.2 Justificación

Al hacer un estudio de mercado, se puede apreciar el mundo de los juegos estilo ‘Fútbol Manager’, pero también se observa la falta de un juego ‘Inazuma Eleven Manager’, y es algo que a los fans de la propia saga, les gustaría tener disponible. TiliManager intenta rellenar ese hueco faltante en el mercado.

1.3 Objetivos

1.3.1 Objetivos generales

Desarrollar una aplicación, que sea sencilla de usar, sea funcional tanto como para personas con experiencia previa en este tipo de juegos, como para novatos. Que sea divertida y esté basado en Inazuma Eleven, que faltan juegos sobre este sector.

1.3.2 Objetivos específicos

Diseñar y desarrollar la mecánica básica del juego (mecánicas principales, interfaz, controles).

Implementar la página del juego (fácil de usar).

Diseñar y construir una API REST con Spring Boot que gestione:

- Creación de usuario, registro y autenticación.
- Envío y consulta de puntos.
- Gestión de equipos de cada liga.
- Gestión de ligas.
- Lista de jugadores.
- Código propio del juego

Diseñar la base de datos (modelo entidad-relación) e implementarla en PostgreSQL.

Desarrollar un frontend web que permita:

- Registro/Login.
- Ver a tu equipo y poder vender jugadores.
- Comprar jugadores en el mercado.
- Crear y unirse a ligas.
- Jugar partidos de liga / torneo contra bots
- Poder ver el historial de partidos de todos los usuarios.
- Administrar contenido (modo admin).

Integrar frontend-backend con API para sincronizar datos.

Desplegar la aplicación (backend + DB + frontend) en una página accesible.

Realizar pruebas y mejora de proyecto (pruebas funcionales y de usabilidad).

1.4 Estrategia y planificación del proyecto

Enfoque de desarrollo:

Desde el inicio del proyecto, se ha optado por diseñar la aplicación web y móvil, el mercado de fichajes y toda la lógica del juego completamente desde cero, prescindiendo de plantillas o gestores de contenido externos. Esta decisión permite tener un control absoluto sobre la arquitectura del sistema. De este modo, se garantiza que la comunicación entre la interfaz visual en Angular y el servidor en Spring Boot se adapte exactamente a las necesidades del juego, como el control estricto del presupuesto, el límite de jugadores por plantilla y el registro de operaciones en el panel de administración.

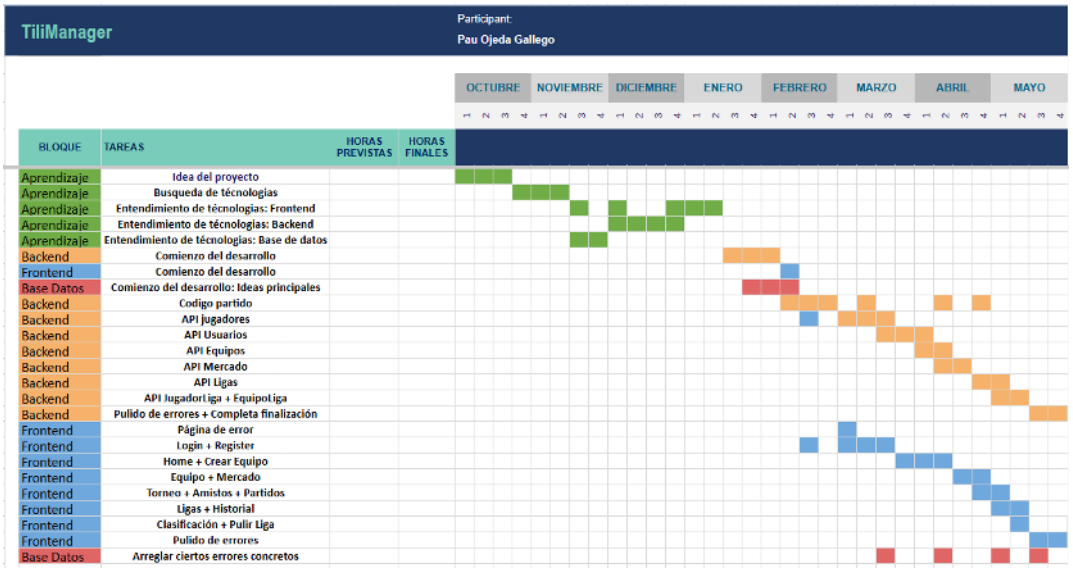
Análisis de viabilidad:

Respecto a la viabilidad temporal, se trata de un proyecto exigente. El desarrollo coordinado de una API REST, una base de datos relacional y un frontend multiplataforma con datos en tiempo real requiere una organización muy clara del calendario para cumplir con los plazos de entrega. A nivel económico, el proyecto no está planteado como un producto comercial para generar ingresos. Sin embargo, su enfoque gratuito y temático lo hace muy viable para llegar a un público amplio. Debido a la popularidad de los juegos estilo Fantasy y el fanatismo por la saga Inazuma Eleven, la aplicación se puede dar a conocer de forma rápida, creando una base de usuarios sólida y dejando el terreno preparado para futuras actualizaciones.

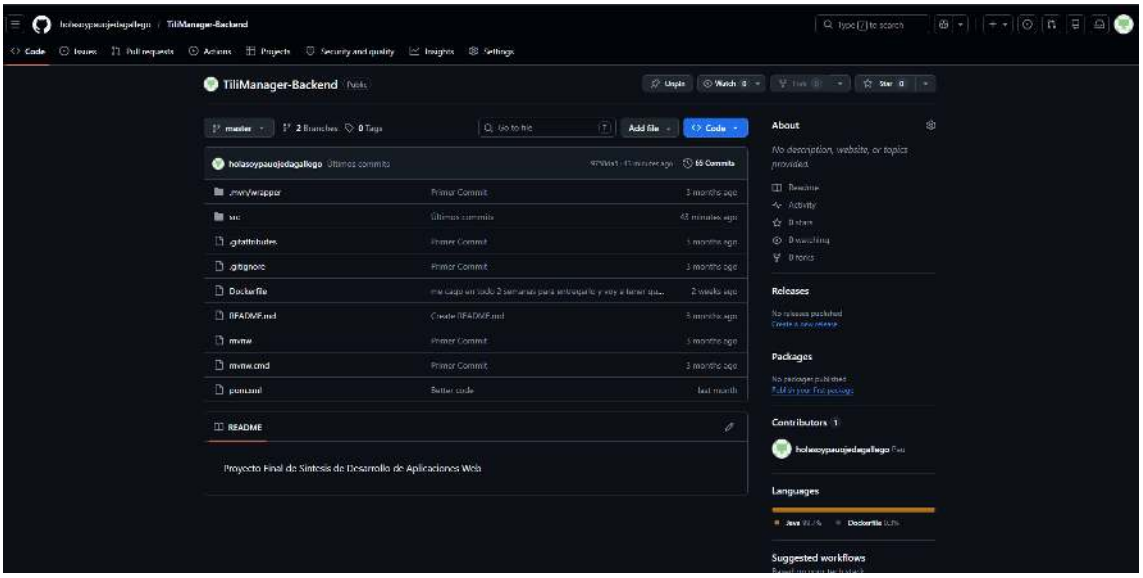
1.5 Metodología de trabajo

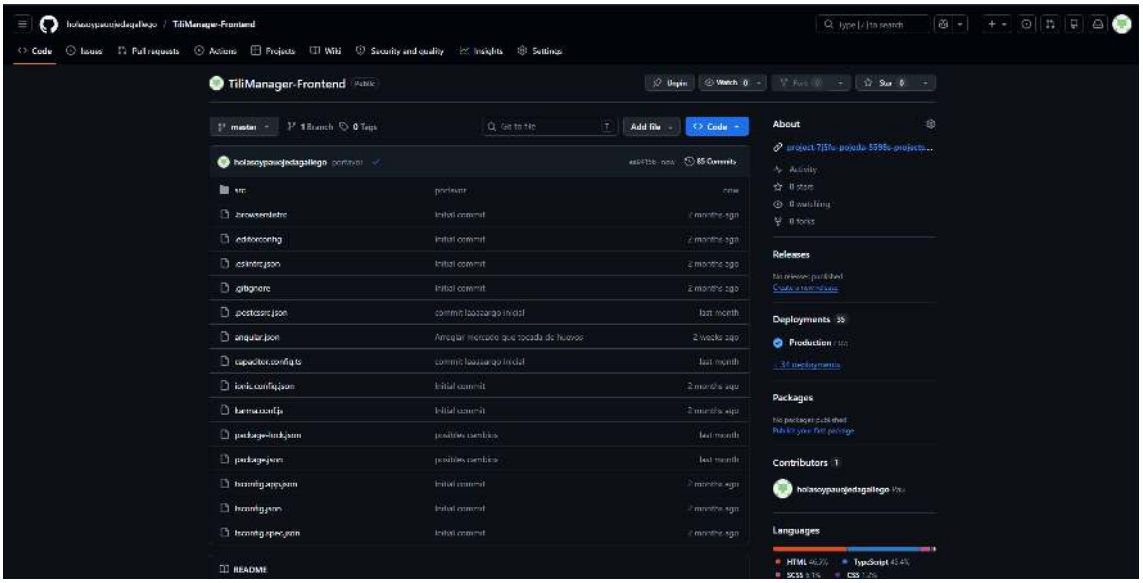
Para la organización de todo el desarrollo del proyecto y no perder tiempo, se ha utilizado dos herramientas clave que ayudan a planificar las tareas y a tener un control de versiones correcto:

Diagrama de Gantt: Para la planificación y el seguimiento de tareas, se ha elaborado un diagrama de Gantt. Esta herramienta permite organizar el proyecto en diferentes fases a lo largo del tiempo, estableciendo las fechas de inicio y fin para cada actividad. De este modo, se obtiene una visión clara del progreso, ayuda a garantizar el cumplimiento de los plazos establecidos.



GitHub: Para el control de versiones del código se ha utilizado GitHub. Esta plataforma ha sido fundamental en el desarrollo, ya que permite tener un repositorio centralizado en la nube con todo el código del Frontend y del Backend. Gracias a esto, se mantiene un historial seguro y detallado de todos los cambios realizados, lo que permite revisar modificaciones o volver a versiones anteriores si ocurre algún error durante la programación sin riesgo de perder el trabajo anteriormente realizado.





1.6 Estudio económico y presupuestario

Inventario de tareas principales (alto nivel)

1. Análisis y diseño (documentación, modelo BD).
2. Desarrollo del cliente (mecánica, UI, assets).
3. Desarrollo backend (API, autenticación, lógica).
4. Desarrollo frontend web.
5. Integración web <-> API.
6. Pruebas (unitarias y funcionales).

7. Despliegue (servidor, configuración).

Componentes y costes orientativos

Software y licencias

IDEs y herramientas: IntelliJ Community, Code OSS — gratuitos.

Frameworks y Librerías: Ionic, Tailwind, Spring Boot, Angular — gratuitas / open-source.

Software: Java, PostgreSQL — gratuitas / open-source.

Recursos gráficos/sonoros: Teniendo en cuenta que se va a usar cosas sobre Inazuma Eleven, se tendría que hablar con Level 5 (Empresa de Inazuma Eleven) para llegar a un acuerdo de uso de su nombre + imágenes, por temas de copyright, con lo cual el proyecto nunca podría ser renumerado por esta razón (A no se que se llegue a acuerdo con Level 5).

Hosting y despliegue

Dominio (opcional): ~10–15 €/año.

Hardware

Ordenador para desarrollo: 300€.

Otros

Dominio y certificado SSL: LetsEncrypt gratuito + arsys de posible dominio pago.

Imprevistos (10–20% del total).

Con este presupuesto, un cliente podría decidir continuar o cancelar. Los costes de mantenimiento anual (servidor + dominio) rondaría 350–550€ /año según elección.

2 Descripción del proyecto

2.1 Anàlisi de requisits

2.1.1 Requisitos funcionales

Gestión de usuarios: El sistema debe permitir el registro de nuevos usuarios, el inicio de sesión mediante credenciales cifradas y la separación de permisos según el rol asignado (USUARIO o ADMIN).

Gestión de ligas: Los usuarios deben tener la posibilidad de crear nuevas ligas competitivas o unirse a ligas ya existentes mediante un identificador, y también ver los puntos (partidos ganados, partidos empatados y partidos perdidos), y que obviamente sean independientes entre ligas.

Gestión de equipos y plantillas: Al entrar en una liga, el usuario debe poder visualizar su plantilla de personajes de Inazuma Eleven. El sistema debe obligar a que las plantillas tengan un tamaño estricto de entre 5 y 7 jugadores para poder competir.

Mercado de fichajes en tiempo real: Debe existir un apartado de mercado donde los usuarios puedan comprar y vender jugadores. El sistema controlará en todo momento que el balance económico del club no quede en números negativos tras una transacción.

Historial de partidos: La aplicación debe registrar un historial con todos los partidos jugados, y que puedan ser rejugados cualquier día.

Jugar encuentros: En la aplicación has de poder jugar partidos, tanto contra un bot como contra jugadores reales.

2.1.2 Requisitos no funcionales

Arquitectura desacoplada: El proyecto debe estar claramente dividido en dos capas independientes: un *Backend* que gestione la lógica y los datos, y un *Frontend* encargado de darle un buen diseño a esa lógica.

Multiplataforma: La interfaz de usuario debe ser híbrida, garantizando un diseño adaptativo que funcione correctamente tanto en navegadores web convencionales como en dispositivos móviles.

Reactividad: La interfaz debe actualizar los datos de manera inmediata ante los cambios del usuario (como la actualización del presupuesto tras una venta) sin necesidad de recargar la página por completo.

Seguridad y persistencia: Las contraseñas de los usuarios deben almacenarse de forma cifrada en la base de datos relacional (PostgreSQL) y cada operación crítica (Vender jugador, comprar jugador) del mercado debe ser validada en el servidor antes de guardarse para evitar fraude.

2.2 Tecnologías

2.2.1 Comparativa de las Tecnologías valoradas

Capa de Interfaz (Backend):

Spring Boot (Java):

- ***Pros***: Es un framework muy maduro, con una gran estabilidad y un excelente rendimiento para gestionar APIs REST y bases de datos relacionales. Cuenta con el respaldo de *Spring Security* para la gestión segura de usuarios.

- ***Contras***: Requiere configurar más estructura de código en comparación con otras alternativas.

Node.js (Express):

- ***Pros**:* Permite programar con JavaScript tanto en el cliente como en el servidor, lo que agiliza el desarrollo y hace que el código sea más ligero.

- ***Contras**:* Para aplicaciones con lógica de negocio compleja o transacciones críticas de mercado, no ofrece el mismo nivel de robustez y tipado estricto que aporta Java.

Capa de Interfaz (Frontend):

Angular + Ionic:

- ***Pros**:* Angular ofrece una estructura muy ordenada mediante componentes y servicios, ideal para proyectos grandes. El nuevo sistema de *Signals* optimiza la reactividad de la interfaz de forma nativa. Por su parte, Ionic permite que el mismo código sirva para web y para aplicación móvil de forma híbrida.

- ***Contras**:* Requiere aprender la sintaxis propia del framework y entender bien la asincronía para evitar problemas de peticiones duplicadas.

React + Ionic:

- ***Pros**:* Es una librería muy popular, flexible y con una comunidad enorme que facilita la creación de interfaces visuales de forma rápida.

- ***Contras**:* Al ser solo una librería visual, obliga a integrar herramientas de terceros para la navegación y la estructura del proyecto, además de requerir más configuraciones añadidas para convertirlo en aplicación móvil.

Capa de Interfaz (Base de Datos):

MySQL:

- ***Pros:** Es una de las bases de datos relacionales más populares del mundo, muy fácil de configurar en entornos locales y con un rendimiento excelente para consultas estándar de lectura y escritura.*

- ***Contras**:* Aunque gestiona muy bien las relaciones simples, puede volverse menos eficiente que otras opciones cuando se realizan

consultas muy complejas o cuando el proyecto escala a niveles corporativos avanzados.

PostgreSQL:

- ***Pros***: Ofrece una robustez superior y funciones avanzadas para la integridad de los datos. Se integra a la perfección con JPA Hibernate en Spring Boot y maneja de forma impecable las relaciones estrictas y complejas entre tablas (como la estructura de usuarios, ligas, equipos y jugadores de este proyecto), además de todo esto, es Código Abierto.

- ***Contras***: Su configuración inicial y la gestión de permisos en el servidor pueden resultar un poco más complejas en comparación con MySQL.

2.2.2 Tecnologías escogidas

Finalmente, tras analizar las diferentes opciones, se han seleccionado las siguientes tecnologías para su estudio en profundidad y su uso en el proyecto:

Java con Spring Boot (Backend): Se ha escogido esta combinación por la robustez y seguridad que ofrece en proyectos de gestión. Al tratarse de un juego que maneja presupuestos, ligas y fichajes, la inyección de dependencias y el tipado estricto de Java garantizan que la lógica del negocio no sufra errores de consistencia. Además, el uso de Spring Security facilita el cifrado correcto de las contraseñas.

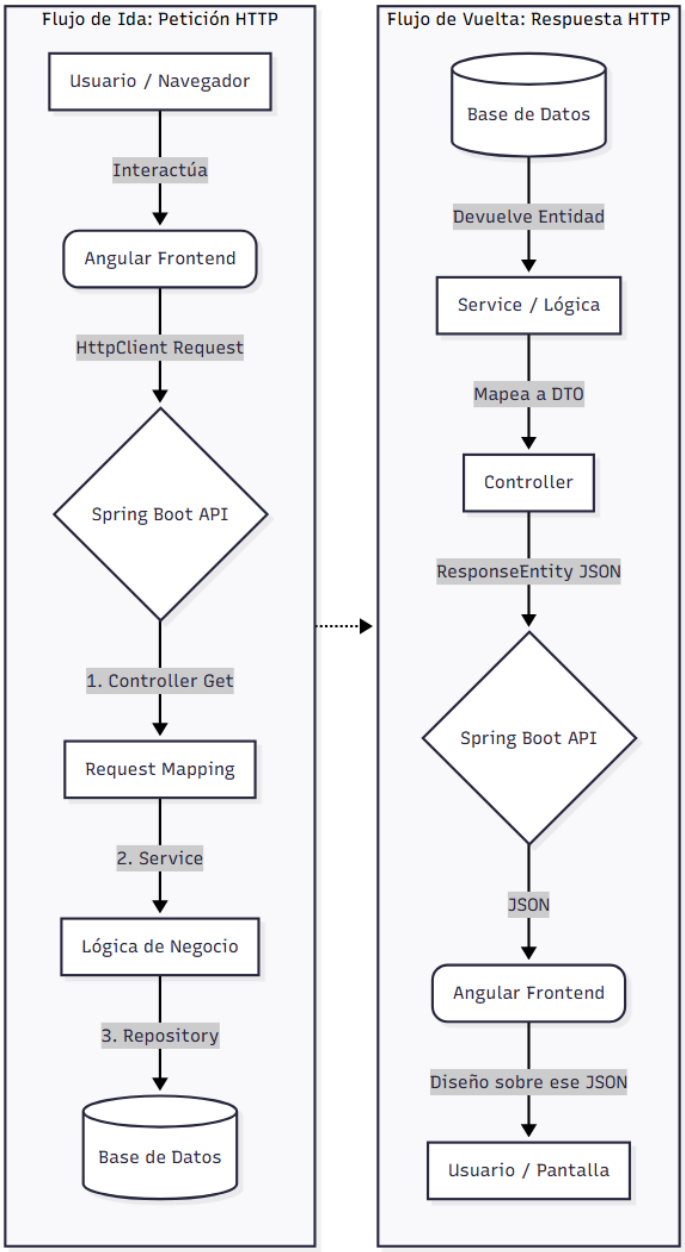
PostgreSQL (Base de datos): Al ser una base de datos relacional, se adapta perfectamente al modelo de datos del proyecto, donde los usuarios, las ligas, los equipos y los jugadores guardan relaciones directas y estrictas entre sí. Su integración con JPA Hibernate desde Spring Boot permite gestionar todas las consultas de forma eficiente.

Angular con Ionic (Frontend): La elección se justifica principalmente por la necesidad de que el proyecto sea multiplataforma. Ionic permite compilar la aplicación tanto para navegadores web como para dispositivos móviles sin tener que programar el código dos veces. Angular se ha seleccionado por su modularidad y por el uso de Signals, lo que ha permitido que la interfaz responda en tiempo real a las acciones del usuario (como el cambio de saldo tras una venta) de manera fluida.

Extra - Tailwind CSS (Estilos): Se ha elegido utilizar Tailwind en vez de estilos normales por el simple hecho de que no cambia nada, solo que las clases con estilos ya están creadas, por ende se agiliza el tiempo al no tener que crear clases con estilos. Y no veo necesario comparar, porque creo que no es una elección, es un añadido al proyecto y pueden convivir juntos (estilos normales y Tailwind).

2.3 Estructura del proyecto

El desarrollo de TiliManager se ha organizado dividiendo el proyecto en dos carpetas o directorios principales independientes: el servidor (**Backend**) con Spring y el cliente (**Frontend**) con Angular. Esta separación facilita el mantenimiento del código y asegura que cada capa cumpla con su función específica.



Aquí se puede apreciar el flujo del proyecto, este empieza con el usuario, por ejemplo interactuando con su propio equipo, le llega la petición al Backend por el Controller, le pasa al Service, que es donde la lógica se haya, busca en la base de datos ese equipo, lo envuelve en una DTO, y lo pasa como JSON al frontend, y ya el frontend es el encargado de con esos datos, darles un diseño entendible, bonito y sencillo para que el usuario normal pueda entenderlo y disfrutar de la aplicación.

2.3.1 Estructura del Backend (Spring Boot)

En la capa del servidor se ha seguido el patrón de diseño por capas estándar de Spring Boot. El código se organiza dentro del paquete principal mediante la siguiente estructura de carpetas:

config/ (Configuración): Contiene las clases encargadas de la seguridad del sistema (Spring Security), los filtros de autenticación y las reglas para

el cifrado de las contraseñas de los usuarios, y también dispone de datos para iniciar correctamente el servidor.

controller/ (Controladores): Es la capa de entrada de la aplicación. Aquí se definen los endpoints REST (las URLs) que reciben las peticiones del Frontend, como por ejemplo las rutas para registrar un usuario, listar el mercado de fichajes o realizar una venta.

dto/ (Objetos de Transferencia de Datos): Carpetas donde se guardan las clases que sirven para moldear y transportar los datos estrictamente necesarios entre el cliente y el servidor, evitando problemas de rendimiento o referencias circulares en los archivos JSON.

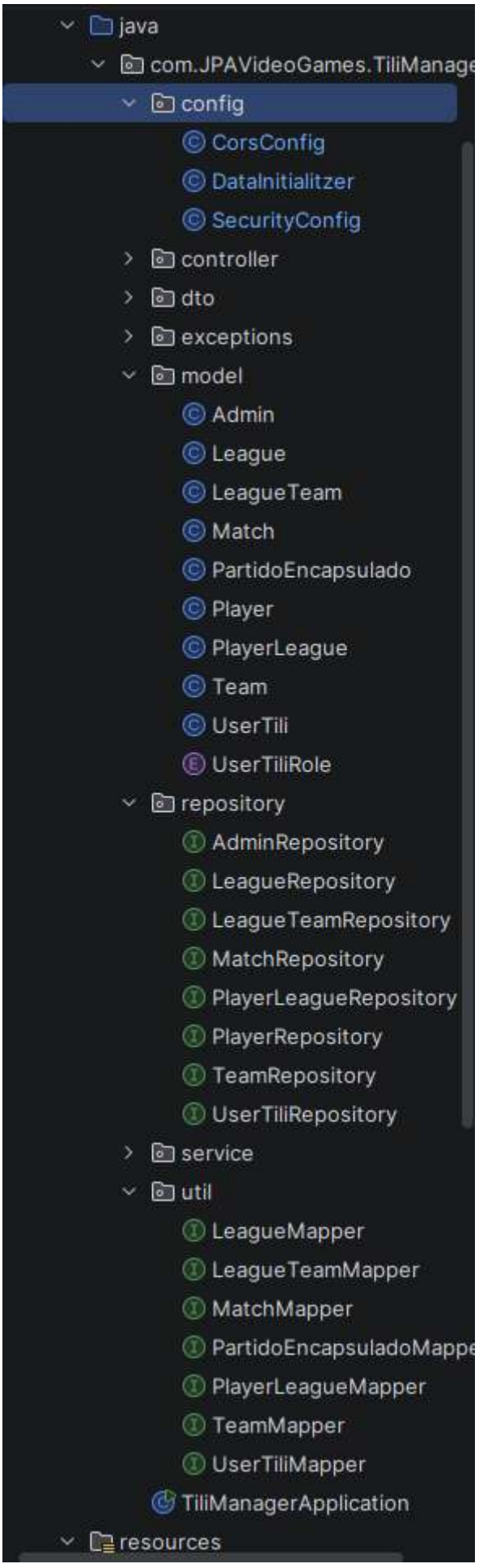
exceptions/ (Excepciones): Esto es para el control de excepciones. Si al crear una liga fallara algo, se podría crear el 'LeagueException' para controlar errores.

model/ (Entidades): Contiene las clases de Java que representan las tablas de la base de datos, como UserTili, Team, League.

repository/ (Repositorios): Son las interfaces que se comunican directamente con la base de datos PostgreSQL utilizando Spring Data JPA. Permiten realizar las operaciones de lectura, escritura y actualización de datos de forma limpia.

service/ (Servicios): Contiene la lógica de la aplicación. En esta carpeta se encuentran las reglas estrictas del sistema, como la comprobación de que un equipo tenga entre 5 y 7 jugadores o la validación del saldo económico antes de confirmar una transacción.

util/ (Utilidades): Contiene todas las herramientas extra para el proyecto, el uso se le ha dado, ha sido para todos los mappers de cada clase, básicamente, para pasar de *entidad* a *dto*, o al revés, con MapStruct, ahorrando así tiempo .


java
com.JPAVideoGames.TiliManage
config
CorsConfig
DataInitializer
SecurityConfig
controller
dto
exceptions
model
Admin
League
LeagueTeam
Match
PartidoEncapsulado
Player
PlayerLeague
Team
UserTili
UserTiliRole
repository
AdminRepository
LeagueRepository
LeagueTeamRepository
MatchRepository
PlayerLeagueRepository
PlayerRepository
TeamRepository
UserTiliRepository
service
util
LeagueMapper
LeagueTeamMapper
MatchMapper
PartidoEncapsuladoMappe
PlayerLeagueMapper
TeamMapper
UserTiliMapper
TiliManagerApplication
resources

2.3.2 Estructura del Frontend (Angular + Ionic)

En la capa de la interfaz de usuario, el proyecto se organiza aprovechando la estructura modular y los nuevos bloques de control que ofrece Angular, distribuyendo los elementos de la siguiente manera dentro de la carpeta raíz src/:

app/components/ (Componentes): Contiene las piezas visuales reutilizables de la aplicación, como la cabecera global (header) o las tarjetas de los personajes (app-jugador-card).

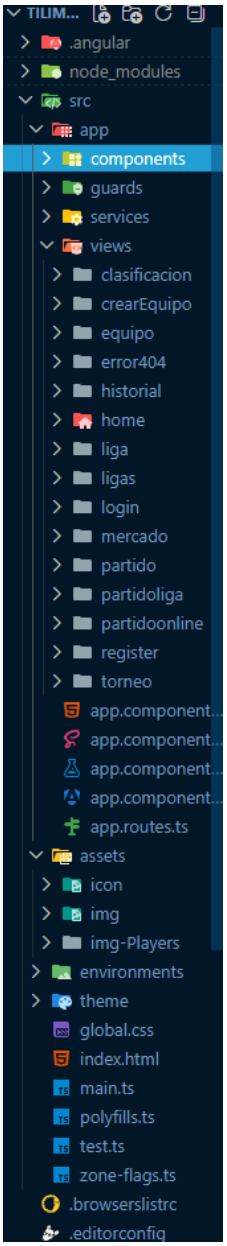
app/views/ (Páginas): Son las pantallas completas por las que el usuario navega, organizadas por módulos (por ejemplo: la pantalla de Login, el panel de gestión del Equipo, la vista del Mercado de fichajes o el Historial).

app/services/ (Servicios del Cliente): Clases encargadas de realizar las peticiones HTTP hacia el Backend y de gestionar el estado global de la aplicación. Aquí se definen las Signals de Angular (como this.auth.team) para que la interfaz reaccione inmediatamente cuando cambian los datos.

app/guards/ (Controladores de acceso): Archivos que controlan el acceso a las diferentes páginas, impidiendo la entrada que un usuario no autenticado pueda entrar a las páginas de liga, o partido o que un usuario común acceda al panel de administración.

assets/ (Imagenes): Aquí se guardan todas las imágenes de cualquier tipo (svg, webp, png, jpg...) en reiteradas carpetas, para poder diferenciarlas fácilmente.

enviroments/ (Desarrollo): Aquí se guarda los entornos de desarrollo, cuando se está en desarrollo, va mejor tener subido en local, y tener un *prod*, qué es para producción, ahí si tenerlo subido en web.



2.4 Descripción de los componentes

TiliManager funciona mediante la interacción de varios componentes bien definidos. A nivel arquitectónico, se utiliza un modelo cliente-servidor donde las responsabilidades están totalmente separadas. A continuación, se describen los componentes principales del sistema y su funcionamiento interno:

2.4.1 Componentes del Servidor (Backend)

- **Módulo de Seguridad y Autenticación (Spring Security):** Su cometido es proteger la API de accesos no autorizados. Cuando un usuario inicia sesión, esta librería intercepta la petición, verifica las credenciales en la base de datos, aplicando un cifrado hashing, y valida el rol del usuario (UserTiliRole). Actúa como una barrera de seguridad.

- **Controladores REST (Capa de Entrada y Salida):** Son los encargados de exponer las URLs de la aplicación. Su funcionamiento interno consiste en recibir los objetos JSON enviados por el Frontend, mapearlos mediante los DTOs, activar el servicio correspondiente y devolver una respuesta envuelta en otra DTO.
- **Lógica (Servicios):** Es el componente central del código del juego. Aquí se ejecutan las reglas y restricciones del sistema de forma estricta a través de métodos Java. Por ejemplo, al invocar la acción de venta, el servicio comprueba el tamaño de la plantilla (rango de 5 a 7 jugadores), calcula el nuevo saldo del equipo después de la venta y, si todo es correcto, autoriza la operación.
- **Capa de Persistencia (Repositories / JPA Hibernate):** Este componente traduce las instrucciones del lenguaje Java a sentencias SQL de forma automática. Permite que el servidor pueda guardar, actualizar o consultar datos en la base de datos sin necesidad de escribir código SQL manual de forma repetitiva.

2.4.2 Componentes de la Interfaz (Frontend)

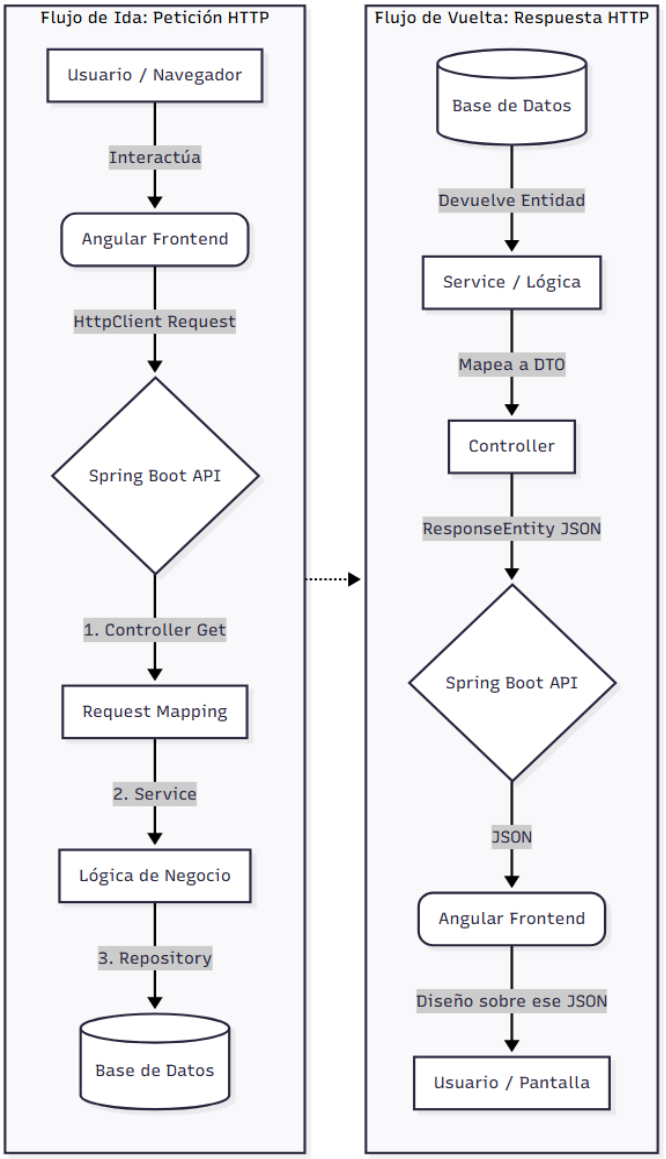
- **Servicios de Comunicación (HTTP Client):** Son los componentes encargados de conectar el Frontend con el Backend. Envían las peticiones web de forma asíncrona (async) hacia el servidor y procesan las respuestas mediante promesas síncronas (await), evitando duplicidades de clics o peticiones fantasma.
- **Gestor de Estado Reactivo (Angular Signals):** Su función principal es controlar el estado de la aplicación en tiempo real (por ejemplo, el presupuesto actual del equipo o la lista de jugadores seleccionados). Cuando un dato cambia en un Signal, este componente notifica inmediatamente a las pantallas para que el usuario vea el cambio reflejado al instante sin recargar la página.
- **Componentes Visuales:** Piezas de interfaz reutilizables (desarrolladas con Ionic y Tailwind CSS) que se encargan de pintar la información en pantalla. Se utiliza también la librería (Angular drag-drop) para arrastrar a los personajes en el equipo, y las tarjetas de los jugadores.

2.4.3 Componente de Almacenamiento (Base de Datos)

- **Base de Datos Relacional (PostgreSQL):** Aquí es donde se guarda toda la información. Estructura los datos en tablas vinculadas entre sí mediante claves primarias y foráneas (*1 Jugador de Liga <-> 1 Equipo*).

2.4.4 Esquema de Funcionamiento Interno (Extendido)

Para entender el flujo conceptual y cómo cooperan todos los componentes ante una acción del usuario (por ejemplo, la venta de un jugador), se sigue el siguiente ciclo de comunicación:



1. El usuario realiza la acción visual de vender un jugador en Angular (Frontend).

2. El Servicio de Angular bloquea el botón para evitar dobles clics, y si todo es correcto, envía una petición al Backend.

3. El Controlador de Spring Boot recibe la petición y el Servicio válida que la plantilla no se quede con menos de 5 jugadores y que la transacción sea lícita.

4. Si la validación es correcta, el Repositorio actualiza el saldo y da de baja al jugador en PostgreSQL (lo cual permite a ese jugador salir en los próximos mercados).

5. El Backend responde con éxito, el Frontend actualiza el Capacitor Storage, y conjuntamente el Signal de presupuesto y la interfaz se redibuja en tiempo real ante el usuario.

2.5 Definición de las funcionalidades

2.5.1 Gestión de Acceso y Perfiles (Registro y Login)

Descripción: Permite a los nuevos usuarios crear una cuenta en la plataforma, y a los ya registrados acceder a la aplicación de una forma segura.

Proceso: El usuario completa un formulario con sus datos de acceso. En el registro, el sistema valida que el nombre de usuario no exista y cifra la contraseña antes de guardarla. En el inicio de sesión, el backend

comprueba las credenciales, y autoriza la entrada cargando el perfil en el frontend, guardando la ID en formato UUID, única para cada usuario, y secreta, siendo la única forma de tenerla haciendo login con correo y contraseña correctos. Ese ID será más tarde utilizado para validar todas las gestiones de cada usuario

Estado: Completamente implementada.

2.5.2 Creación e Incorporación a Ligas

Descripción: Funcionalidad para que los usuarios puedan crear sus propias ligas donde competir entre sí de forma independiente.

Proceso: El usuario puede elegir entre dos opciones: crear una liga nueva (introduciendo un nombre, donde en el backend ya se encargará de crearla al completo a partir de ese nombre) o unirse a una existente. Al hacerlo, el backend genera automáticamente un equipo vacío asociado a esa liga y a ese usuario en particular.

Estado: Completamente implementada.

2.5.3 Gestión de la Plantilla y Alineación

Descripción: Permite al usuario visualizar los personajes de Inazuma Eleven que tiene en propiedad y organizar su disposición sobre el terreno de juego.

Proceso: Se muestra la lista de jugadores disponibles en el equipo. En caso de no tener jugadores (Te acabas de unir a una liga) se te redireccionará en el Frontend a una página para crear el equipo, y el nombre de este, donde el backend validará esos datos, y el saldo final del equipo. Una vez con equipo, utilizando la funcionalidad táctil o de ratón Drag and Drop, el usuario arrastra las tarjetas de los jugadores para colocarlos en sus respectivas posiciones. El sistema comprueba constantemente que el tamaño total del equipo se mantenga en el rango obligatorio de entre 5 y 7 jugadores (En caso de 5, no podrás vender más, en caso de 7, no podrás comprar más).

Estado: Completamente implementada.

2.5.4 Mercado de Fichajes y Control Económico (Compra/Venta)

Descripción: Aquí se permitirá al usuario realizar las transacciones económicas de los jugadores disponibles en la liga.

Proceso: Cada liga tendrá su propio mercado, cada uno independiente entre ellos, solo dependiente de la liga, y se actualizará a tiempo real. (Cada día a las 24 de la noche se actualizará el mercado, y no se podrá comprar hasta el próximo día a las 16, dando tiempo así a visualizar los jugadores, sin que te los quiten). En susodicha sección, se listan los jugadores transferibles con sus precios en formato reducido (ej. 3M). Si el

usuario decide comprar, el backend descuenta el dinero y valida que el saldo no sea negativo. Si decide vender, se procesa la transacción mediante una promesa síncrona para evitar dobles clics accidentales, se añade el dinero al presupuesto y el frontend cambia el estado de su Signal para actualizar la pantalla al instante. El sistema bloquea la venta si la plantilla va a quedar con menos de 5 jugadores, lo mismo si compra y ya tiene 7 jugadores.

Estado: Completamente implementada.

2.5.5 Historial de Partidos

Descripción: Sistema de guardado de todos los partidos hasta la fecha

Proceso: El usuario ve toda una lista de partidos, con la fecha en la que se jugaron, que dos equipos jugaron entre sí, en caso de que sea un partido de liga, salga en qué liga han jugado y cual es el resultado final. Teniendo una opción para revivir los encuentros.

Estado: Completamente implementada.

Aquí tienes la reestructuración completa del apartado 2.5.5 Sistema de Partidos, desglosado detalladamente en cuatro subclases o modalidades de juego diferentes, manteniendo el nivel de escritura directo, claro y en formato impersonal:

2.5.5 Sistema de Partidos y Modos de Juego

Descripción: Mecánica competitiva central encargada de evaluar la fuerza y la disposición táctica de las plantillas creadas por los usuarios. El sistema cruza las estadísticas ofensivas y defensivas de la alineación para resolver los encuentros y simular los resultados. Esta funcionalidad se divide en cuatro modalidades bien definidas:

2.5.5.1 Partido Torneo (Eliminatoria contra Bot):

Proceso: El usuario inscribe a su equipo en una competición de eliminación directa contra un bot controlado por la aplicación. Esta modalidad cuenta con un factor de riesgo: si el equipo del usuario se alza con la victoria, el sistema le obsequia una recompensa en metálico para aumentar su saldo, por el contrario, si el equipo pierde el encuentro, se le resta dinero de su saldo.

Estado: Completamente implementada.

2.5.5.2 Partido de Liga:

Proceso: Consiste en el enfrentamiento directo contra otros equipos que pertenecen a la misma liga del usuario. Una victoria otorga una recompensa económica elevada y suma 3 puntos en la tabla de la liga, mientras que un empate suma 1 punto. A diferencia del modo torneo, sufrir una derrota no resta en tu saldo, simplemente se te obsequian 0 puntos añadidos en la clasificación.

Estado: Completamente implementada.

2.5.5.3 Partido Amistoso:

Proceso: Permite disputar un partido amistoso contra cualquier equipo registrado en la base de datos de la aplicación, independientemente de si pertenece o no a la misma liga que el usuario. Es un modo diseñado para testear alineaciones y probar el rendimiento de los personajes sin que el resultado cambie los puntos de la clasificación ni afecte al presupuesto.

Estado: Completamente implementada.

2.5.5.4 Revivir Partido:

Proceso: Al acceder al apartado del historial de partidos, el sistema permite seleccionar cualquier enfrentamiento pasado. Al pulsar sobre él, la aplicación recupera de la base de datos el registro exacto de las alineaciones, el marcador y los goles y situaciones que ocurrieron en su momento, permitiendo al usuario volver a revivir de forma cómo se desarrolló susodicho encuentro.

Estado: Completamente implementada.

3 Diseño e Implementación

En este capítulo, se explicará más a detalle, cómo funciona cada apartado de la aplicación, en un lenguaje más técnico:

3.1 Implementación del Backend

Comenzando por el backend, que es donde la lógica de toda la aplicación se haya:

3.1.1 Lógica Partido

La lógica encargada de resolver los encuentros se ejecuta íntegramente en la capa del Backend dentro de los servicios de Spring Boot. Esta decisión garantiza que los usuarios no puedan cambiar los resultados modificando el código desde el navegador.

3.1.1.1 Algoritmo de simulación:

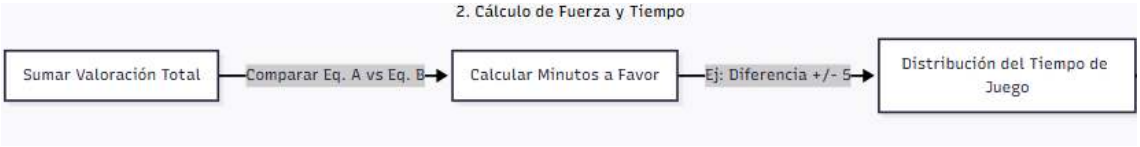
Cuando se solicita comenzar un encuentro, el servidor no calcula el resultado de forma aleatoria, sino que coge los datos de los dos equipo y da el resultado basado en las estadísticas reales de las alineaciones guardadas en PostgreSQL:

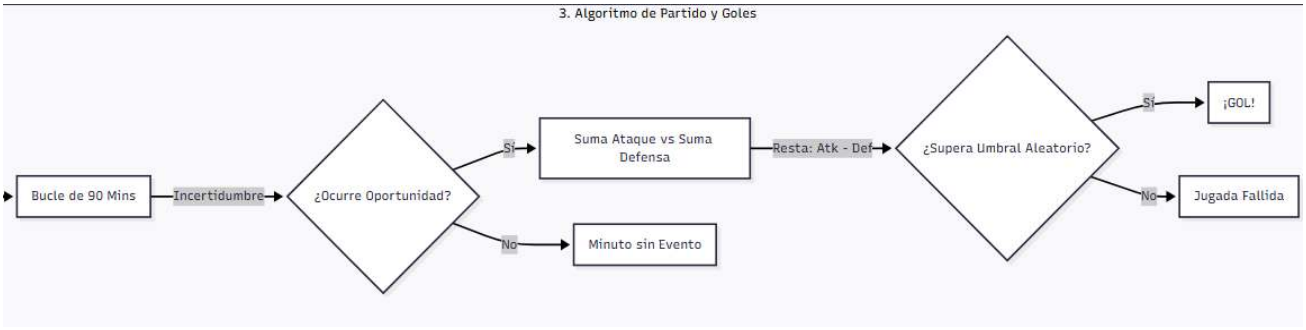
Carga de coeficientes: El sistema recupera los atributos de ataque, defensa y valoración de los personajes de Inazuma Eleven que el usuario tiene en su equipo.

Cálculo de valores: Se calcula el valor total de la fuerza del equipo sumando los puntos de la valoración de cada personaje y se compara con los valores del rival (ya sea un bot o el equipo de otro usuario).

El equipo que tenga una mejor valoración, tendrá más minutos a su favor de los 90 que hay en cada partido. (Si por ejemplo, el valor de equipo A es de 194 y el valor de equipo B es de 189, equipo A tendrá 50 minutos a su favor y equipo B 40 minutos [equipo A = 45+5, equipo B = 45-5])

Resolución: Para simular la incertidumbre de los partidos de Inazuma Eleven, y también hacer que ningún partido sea igual, lo que se hace es añadir un algoritmo que determina si puede llegar a pasar algo, o no, si llega a suceder algo, puede suceder una oportunidad de gol. Para la oportunidad de gol, primero, se suma la defensa de todo el equipo que está siendo chutado, y todo el ataque del equipo que está chutando. Después se resta el Ataque del equipo atacante con la defensa del otro equipo, y si supera el valor generado por un algoritmo, acabará esa jugada en gol, por el contrario, no acabará en gol





3.1.1.2 Gestión de las modalidades de juego:

Dependiendo del modo de partido seleccionado por el cliente, el servicio del Backend activa una lógica de negocio específica para actualizar el estado de la base de datos:

Control del modo Liga: Si el partido corresponde a una liga, el sistema registra el resultado en la tabla de clasificaciones. En caso de victoria o empate, invoca los métodos correspondientes para sumar los puntos (3 o 1 respectivamente) y realiza un ingreso en el saldo del club. Si el equipo pierde, se guardan 0 puntos en la jornada, pero no se le resta nada en el saldo.

Control del modo Torneo: Al tratarse de una eliminatoria directa contra un bot, el servicio aplica una lógica de riesgo financiero. Si el algoritmo determina que el usuario gana, se calcula una recompensa económica y se ingresa en el saldo del equipo. En caso de derrota, el servicio ejecuta una resta del saldo del equipo.

Control del modo Amistoso: El endpoint específico para los amistosos procesa el partido emulando las estadísticas de cualquier club de la base de datos. No hay ni recompensa ni penalización económica.

Persistencia para Revivir Encuentros: Para hacer viable la funcionalidad de consultar el historial, cada partido finalizado genera un registro en la base de datos. No solo se almacena el marcador final, sino que se guarda la estructura de las alineaciones en el momento exacto del partido, y todas las ocasiones de gol. Esto permite que, en futuras actualizaciones, el servidor pueda servir estos datos estructurados para recrear visualmente el encuentro tal y como sucedió.

3.1.1.3 Seguridad:

Cuando se envía la solicitud de jugar un partido, se valida la ID de ese usuario, que es completamente secreta, y la única manera de conseguir el ID de un usuario es iniciando sesión con la contraseña correcta.

Antes de enviar cada encuentro, toda la información de cada encuentro se encapsula en una DTO donde no haya información sensible (es decir, que la información del usuario de la otra persona, como la contraseña, el ID del otro usuario, no salga del backend por seguridad).

Aquí tienes el subapartado 3.1.2 Lógica de Usuarios enfocado en el Backend, manteniendo de forma estricta el nivel de escritura directo, técnico, claro e impersonal utilizado en los bloques anteriores.

3.1.2 Lógica Usuarios

La gestión de usuarios y la seguridad de la plataforma se centralizan en el Backend mediante la integración de Spring Security. Este componente es el encargado de cifrar las contraseñas de todos los usuarios y asegurar que cada petición web esté vinculada a un usuario legítimo con los permisos adecuados.

3.1.2.1 Flujo de Registro y Cifrado de Datos:

Cuando se recibe una solicitud de creación de cuenta a través del endpoint de registro, el servicio ejecuta una serie de validaciones lógicas antes de interactuar con la base de datos:

Verificación de duplicados: Se realiza una consulta previa en el repositorio (UserTiliRepository) para comprobar que el nombre de usuario introducido no esté registrado en el sistema. Si el usuario ya existe, el servicio interrumpe el proceso y devuelve una excepción controlada hacia el Frontend.

Cifrado de la contraseña: Por criterios estrictos de seguridad, las contraseñas nunca se almacenan en texto plano. El servicio aplica un algoritmo de hashing seguro para encriptar la clave antes de realizar la persistencia en PostgreSQL.

Asignación de Rol: Al guardar el nuevo perfil, el sistema le asigna por defecto el rol de usuario común. Su susodicho rol le permite jugar al juego como un usuario común.

ID Secreto: El ID, ha de ser secreto, es un UUID generado al crear el usuario en base de datos, y se le facilita al usuario al iniciar sesión con las credenciales correctas, el ID ha de guardarse en el Frontend, y sirve para cualquier cambio que se desee hacer, por ejemplo, al vender un jugador, la forma de validar correctamente que el usuario es el real, es por la ID secreta.

3.1.2.1 Seguridad:

Ante cualquier petición con datos del usuario (que abarca casi todas las peticiones que se hacen en el backend), se encapsula con mucho cuidado, de la siguiente manera:

Validación de datos:

Es la capa con las restricciones más estrictas, ya que se encarga de auditar los datos introducidos por un usuario anónimo en el formulario de registro e inicio de sesión:

En los procesos donde el usuario envía información de Registro o Inicio de sesión, estos DTOs obligan a que los datos cumplan con reglas estrictas de formato. Se controlan los campos vacíos, se validan las sintaxis de los correos y se limitan los tamaños del texto 3-16 carácteres para el nombre de usuario. Además, se aplican expresiones regulares para restringir caracteres especiales, lo que previene de forma drástica ataques comunes de inyección de código o la inserción de datos masivos que puedan saturar el sistema.

Ofuscación de Información:

En los procesos donde el servidor responde al cliente, estos DTOs actúan como un escudo de privacidad. Al transformar las entidades internas en objetos simplificados, se garantiza que la aplicación solo exponga datos públicos (como el nombre o el correo). Los datos críticos y altamente sensibles, como los ID secretos de la base de datos o los hashes de las contraseñas quedan completamente fuera, eliminando cualquier riesgo de fuga de información a través de la red.

3.1.3 Lógica Ligas

La gestión de las ligas y las agrupaciones de equipos constituye el núcleo organizativo del sistema. A nivel de negocio, se han implementado reglas e instrucciones que garantizan la integridad de la competición:

3.1.3.1 Creación y Control de Límites:

Al procesar la creación de una nueva competición mediante una DTO de la liga, el servidor realiza un control de saturación consultando el repositorio. Se restringe de forma estricta que un mismo usuario pueda ser propietario de más de 3 ligas de forma simultánea. Tras superar la validación, el sistema continua para generar todos los jugadores para esa liga y automatiza el alta del mercado de fichajes inicial.

En una liga, también podrás eliminar equipos de usuarios si eres el dueño de esta, también podrás borrar la liga en sí, borrando todos los jugadores relacionados a tu liga, y todos los equipos creados para jugar en la liga.

3.1.3.2 Integridad en el Borrado Cascada:

Para evitar registros huérfanos en PostgreSQL, el borrado de una liga exige una validación de seguridad de triple factor, cruzando el ID secreto, el nombre y el correo electrónico del propietario que solicita la baja. Una vez confirmada la autoría, el sistema vacía secuencialmente las listas de equipos desvinculando a los jugadores de sus respectivos clubes y borrando la relación entre esos jugadores y la liga.

3.1.3.3 Inscripción y Restricciones de Acceso:

El método encargado de añadir equipos a una liga pública valida que el identificador de la liga exista, que el usuario sea correcto y que la competición no se encuentre en estado privado. Para mantener la estabilidad del motor de juego y evitar problemas de rendimiento en el almacenamiento, la entidad League autogestiona el tamaño de la comunidad, denegando de forma automática cualquier intento de inscripción si la liga alcanza el límite máximo de 20 equipos permitidos.

3.1.4 Lógica Equipos

La gestión de las plantillas e integrantes constituye el núcleo dinámico del simulador. El backend procesa todas las transacciones financieras y estructurales de los clubes, asegurando que no se rompa nada al alterar un equipo.

3.1.4.1 Estructura y Reglas Básicas del Equipo:

El ciclo de vida de un equipo se rige por restricciones algorítmicas severas encargadas de validar su composición en el almacenamiento persistente:

Límite Estricto de Plantilla: Un equipo está diseñado para albergar un máximo de 7 jugadores. El sistema intercepta las solicitudes de incorporación y bloquea de manera automática si el club intenta exceder este límite. En el sentido inverso, para garantizar la integridad del equipo, las operaciones de venta exigen que la plantilla no se reduzca por debajo del mínimo de 5 jugadores.

Control Presupuestario: El balance financiero se evalúa de forma transaccional. Durante un proceso de adquisición, el servidor calcula el precio del jugador frente al saldo actual del club, si la operación resulta en un saldo negativo, la base de datos revierte el proceso instantáneamente para evitar fraudes económicos.

3.1.4.2 Modelado y Arquitectura de Relaciones (Equipo, Liga):

Para hacer viable que la plataforma aloje múltiples ligas independientes simultáneamente, se ha diseñado una estructura de relaciones entrelazadas en la base de datos:

La Entidad del Equipo (Identidad Global):

Representa la propiedad base del usuario (su nombre único, su propietario y su balance económico global). Conceptualmente, un usuario posee un equipo base. Sin embargo, este equipo no puede competir ni albergar estadísticas deportivas de forma directa si no está vinculado a un entorno competitivo cerrado.

La Entidad de Relación de Competición (Liga y Equipo):

Para conectar ambas realidades de forma óptima, se ha diseñado una tabla intermedia de emparejamiento. Esta entidad funciona como una extensión deportiva del equipo dentro de una liga específica. Su propósito fundamental se divide en dos vertientes técnicas:

Separación de Estadísticas: Permite que el equipo guarde su registro de partidos ganados, empatados, perdidos, así como el cómputo de goles a favor y en contra de manera exclusiva para esa competición, manteniendo intacta su identidad global.

Restricción de Unicidad: El sistema aplica una regla de validación indexada que prohíbe terminantemente que un mismo equipo físico pueda inscribirse dos veces en la misma liga, previniendo duplicidades en los emparejamientos y garantizando una clasificación limpia.

3.1.4.3 Flujo Transaccional de Auditoría (Compra y Venta):

Cada vez que un equipo interactúa con el mercado, la lógica de negocio procesa el intercambio modificando el saldo y la disponibilidad del jugador dentro de la liga. Una vez confirmada la validez de la operación, el sistema de almacenamiento guarda el nuevo saldo del equipo, y añade al jugador a el mismo. También al hacer eso, se genera un registro histórico inmutable con las identidades del equipo y del jugador, permitiendo una supervisión completa de la actividad desde las herramientas de administración.

3.1.5 Lógica Jugadores

El diseño de los jugadores de la aplicación requiere una arquitectura capaz de resolver un desafío técnico: permitir que múltiples ligas interactúen con el mismo catálogo de jugadores simultáneamente, sin que las compras, ventas o estadísticas de una liga afecten a las demás.

3.1.5.1. Definición del Catálogo Maestro de Personajes:

El sistema parte de una base de datos global e inmutable que actúa como diccionario o catálogo maestro donde todos los jugadores estarán guardados con todas las estadísticas. En este nivel, cada jugador se define exclusivamente por sus rasgos de identidad y sus valores técnicos:

Métricas de Rendimiento: Atributos numéricos de valoración general, nivel de ataque y nivel de defensa que servirán como valores para la lógica de partidos.

Valor Económico: Un precio base de mercado según su nivel de rendimiento.

Personajes Especiales: En el catálogo donde se hallan todos los jugadores, también hay ciertos jugadores que son especiales, es decir, no se pueden fichar por usuarios normales, son específicamente para los bots y están separados del resto.

3.1.5.2. Arquitectura - Jugador y Liga:

Para viabilizar la coexistencia de ligas independientes, la aplicación prohíbe que un equipo se vincule directamente al *catálogo maestro* de jugadores. En su lugar, se ha implementado una clonación a través de una entidad intermedia que vincula los jugadores a una liga.

- Cuando una nueva liga se da de alta en el sistema, la lógica de negocio activa un proceso automatizado:
- Extrae el listado completo de jugadores desde el catálogo *maestro*.
- Genera una copia única de cada personaje para esa liga en particular.
- Inicializa las variables dinámicas a nivel de entorno (goles anotados en esa competición e identificador del equipo propietario).
- Gracias a esta abstracción, cada liga gestiona su propia burbuja de datos. Un jugador puede ser propiedad de un usuario en una liga determinada y, simultáneamente, estar libre en el mercado de fichajes de otra liga distinta, teniendo además un historial de goles completamente independiente en cada una de ellas.

3.1.5.3. Lógica de Consulta y Estado del Mercado:

Los servicios de extracción de datos del servidor actúan sobre esta capa intermedia para alimentar la interfaz del usuario, segmentando las consultas según el contexto de la petición:

Filtro de Agrupación: Permite recuperar de forma aislada los personajes asignados a un club concreto dentro de una liga para renderizar la alineación táctica del usuario.

Gestión de Disponibilidad: Almacena y expone el listado de jugadores cuyo propietario se encuentre vacío o nulo. Este subconjunto de datos define dinámicamente la cartelera del mercado de fichajes de la liga, asegurando que solo los futbolistas sin club asignado sean elegibles para operaciones de compra.

3.1.5 Lógica Mercado

El mercado de fichajes opera como un entorno financiero dinámico encargado de regular la compra y venta de los personajes dentro de cada una de las ligas activas. Su arquitectura está diseñada para gestionar la disponibilidad de los activos sin sobrecargar el almacenamiento permanente, aplicando mecánicas de volatilidad temporal para incentivar la concurrencia de los usuarios.

3.1.5.1. Aislamiento y Estructura de la Cartelera de Fichajes:

Para evitar colisiones entre ligas, la capa de servicios inicializa un mercado independiente para cada liga. Cada uno de estos mercados consta de dos elementos clave:

Muestra Aleatoria Restringida: El sistema realiza una consulta selectiva para extraer exclusivamente los personajes clonados de esa liga sin equipo (agentes libres). Si la lista es muy extensa, el servidor aplica un algoritmo de barajado aleatorio y extrae un conjunto limitado a un máximo de 20

personajes simultáneos. De este modo, se garantiza un mercado variado y único en cada liga.

Control de Operabilidad: Cada mercado gestiona una lógica que determina si las compras están permitidas. Esta propiedad bloquea o habilita la capacidad de comprar jugadores..

3.1.5.2. Automatización de Ciclos Temporales (Tareas Programadas del Servidor):

La volatilidad del mercado se automatiza mediante la configuración de un planificador de tareas en el servidor que actúa en base a horarios preestablecidos:

Ciclo de Rotación y Reinicio (23:59): Al finalizar la jornada, el planificador ejecuta una actualización global. Este proceso elimina la cartelera de jugadores del día anterior, consulta de nuevo la base de datos de agentes libres de cada liga, selecciona un nuevo grupo de hasta 20 jugadores de forma aleatoria y congela temporalmente la posibilidad de fichar. Esto fuerza a que los mercados se renueven cíclicamente de forma automática.

Ciclo de Apertura Financiera (16:00): A las 16:00, una segunda tarea automatizada se ejecuta en el mercado y cambia a estado activo la posibilidad de fichar. A partir de ese instante, el servidor vuelve a validar y procesar las solicitudes de compra recibidas desde las interfaces de los clientes.

3.1.5.3. Sincronización Transaccional Inmediata:

Para mantener la consistencia de los datos y evitar problemas de concurrencia (como que dos usuarios intenten adquirir al mismo personaje simultáneamente en la misma liga), el mercado implementa actualización en tiempo real. En el momento exacto en que un club compra con éxito la compra de un futbolista superando el control presupuestario, el servidor intercepta el mercado correspondiente, localiza al jugador y actualiza su estado vinculándolo al identificador del nuevo equipo propietario. Esta acción retira inmediatamente al jugador del flujo de agentes libres visibles, garantizando una simulación limpia y transparente.

3.1.6 Lógica Historial

La persistencia del historial competitivo cumple una doble función dentro de la arquitectura del sistema: actúa como un registro de auditoría inmutable para verificar la integridad de la simulación y proporciona los datos detallados necesarios para reconstruir la narrativa de los encuentros ante los usuarios.

3.1.6.1. Estructura del Registro del Encuentro:

Cada vez que la lógica de simulación resuelve un partido, los datos resultantes se guardan en una entidad de registro histórico. Esta estructura almacena de forma permanente los siguientes componentes:

Metadatos e Identidades: Almacena de forma explícita las referencias del entorno (la liga donde ocurrió el suceso, en caso de que fuera un partido de

liga), los dos clubes implicados (local y visitante) y una marca de tiempo generada automáticamente en el momento exacto de la persistencia para datar el evento.

Marcador Final: Registra los atributos numéricos correspondientes al cómputo total de anotaciones de ambos bandos, sirviendo como la fuente para los cálculos de la tabla clasificatoria.

3.1.6.2. Patrón de Encapsulamiento de Sucesos en Tiempo Real (Cronología del Partido):

Para evitar almacenar únicamente un resultado frío y permitir que la interfaz del usuario recree el desarrollo del partido minuto a minuto, la lógica de negocio implementa una técnica de encapsulamiento secuencial.

Durante la simulación de los 90 minutos de juego, el backend genera una lista dinámica de micro-eventos cronológicos. Cada uno de estos sucesos encapsula una porción aislada de información con una estructura estricta:

Indicador Temporal: El minuto exacto en el que ocurre la acción.

Jugador y Club Asociado: La identidad del jugador que realiza la acción y el club al que defiende en ese instante.

Naturaleza del Suceso: Guarda que sucede en ese instante, si hubo gol, o si la jugada acabó en nada.

Estado del Marcador Parcial: El resultado instantáneo de goles locales y visitantes justo tras la conclusión de dicho suceso.

Esta colección de micro-eventos se acopla de manera compacta dentro del registro principal del partido. Al persistir de forma integrada, se garantiza que toda la película del encuentro quede ligada indisolublemente al identificador del partido.

3.1.6.3. Estrategia de Consulta y Consumo de Datos:

La capa de extracción de información expone estos registros aplicando filtros basados en el contexto de la navegación del cliente:

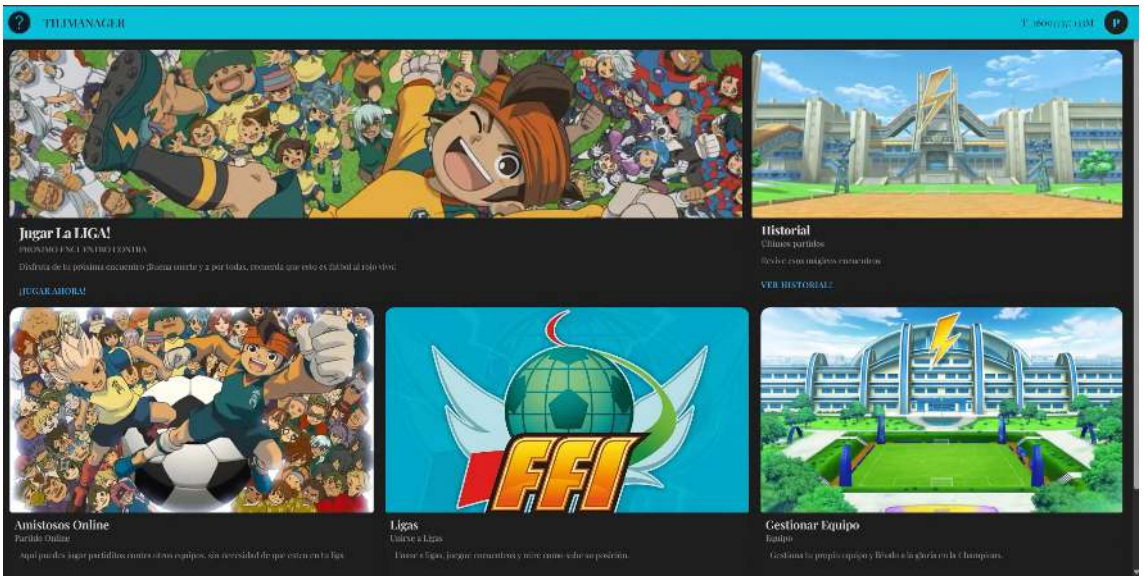
Segmentación por Entorno: El sistema indexa las consultas en el almacenamiento para recuperar cronológicamente todos los encuentros disputados dentro de una liga específica, facilitando la visualización del calendario general.

Reconstrucción Narrativa: Al solicitar un encuentro concreto mediante su identificador único, el servidor transfiere tanto los datos globales como la lista encapsulada de sucesos. Esto permite que el front-end decodifique los códigos numéricos y reproduzca de forma interactiva la simulación de los eventos ante el usuario, garantizando un consumo de red eficiente al empaquetar todo el historial en una única transacción de datos.

3.2 Implementación del FrontEnd

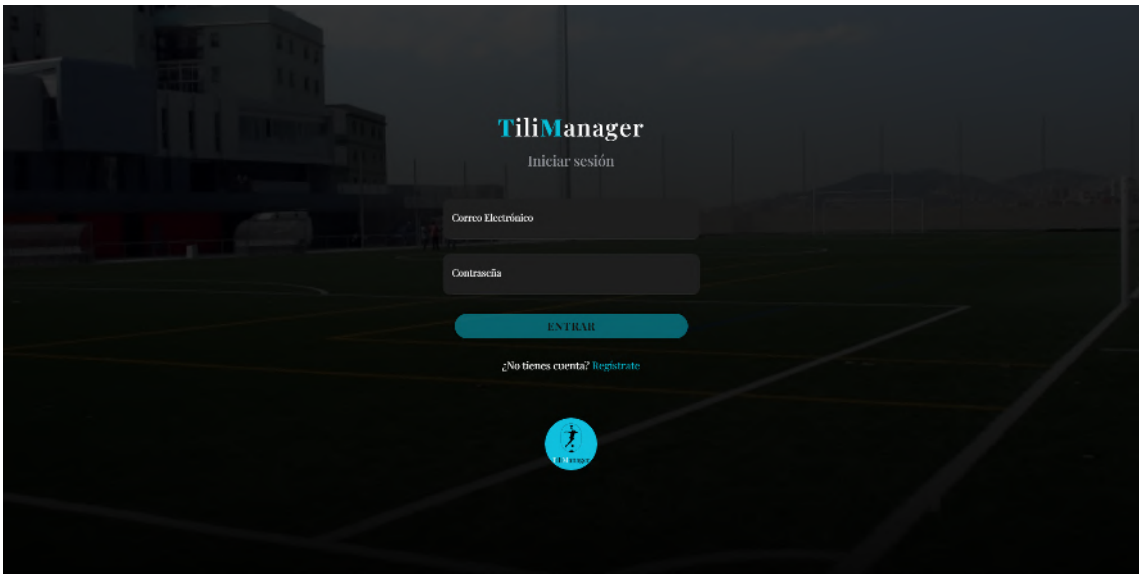
Esta sección detalla la arquitectura de la interfaz de usuario, describiendo la disposición visual, el comportamiento dinámico de los componentes y las reglas de para la correcta navegación, para que así los usuarios tengan una correcta experiencia a lo largo de la página.

3.2.1 Home:



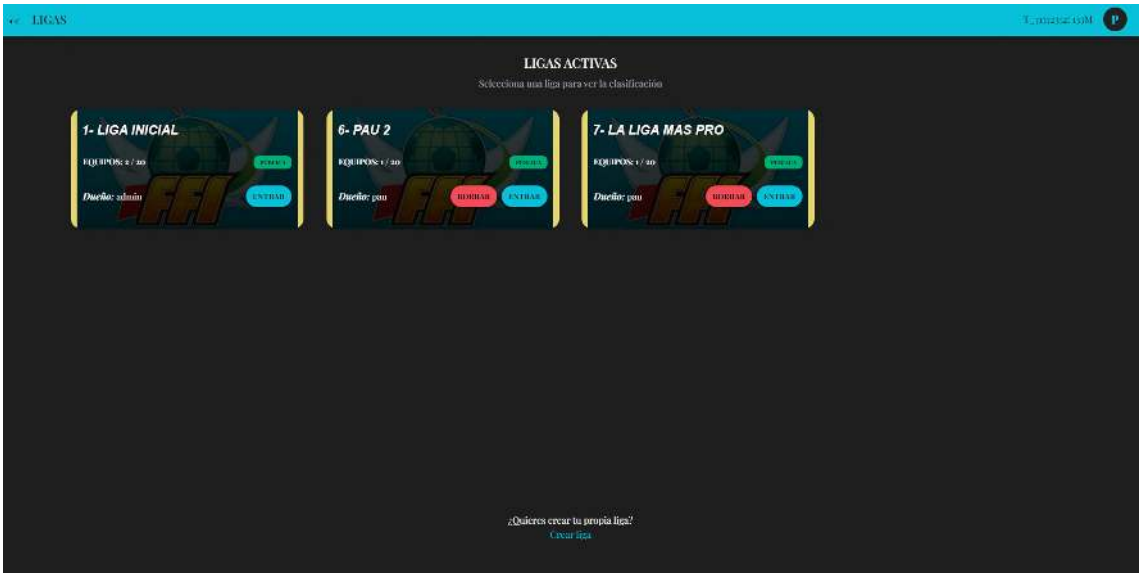
La importancia del diseño es elevada para una correcta página web, así que lo que se intenta aquí, es que sea muy sencilla al usuario de entender, con bloques muy grandes para cada opción, y con un título concreto, para que el usuario nada más leerlo, pueda de una manera sencilla, ejecutar su próxima opción. También dispone de un botón de ayuda, para quien se sienta perdido, muy importante para hacer una web más accesible y sencilla de usar.

3.2.1 Registro e Inicio de sesión:

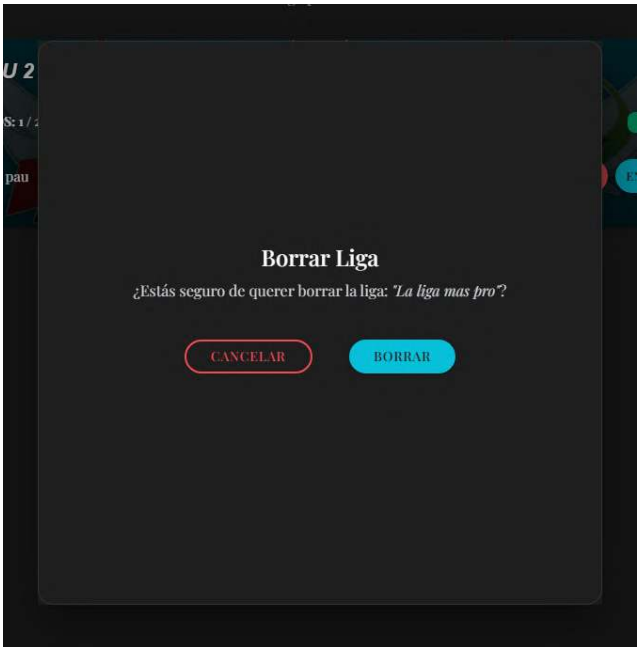


En esta parte, lo más importante e imprescindible es que el usuario sepa porque falla, entonces hay que cuidar muy bien porque el usuario puede fallar (que el nombre sea muy largo que el correo no se validó, que la contraseña sea muy corta) y estar preparado de antemano, para poner un error y que el usuario sepa de una manera muy sencilla que está fallando, para que así susodicho no se frustre y deje de usar la aplicación.

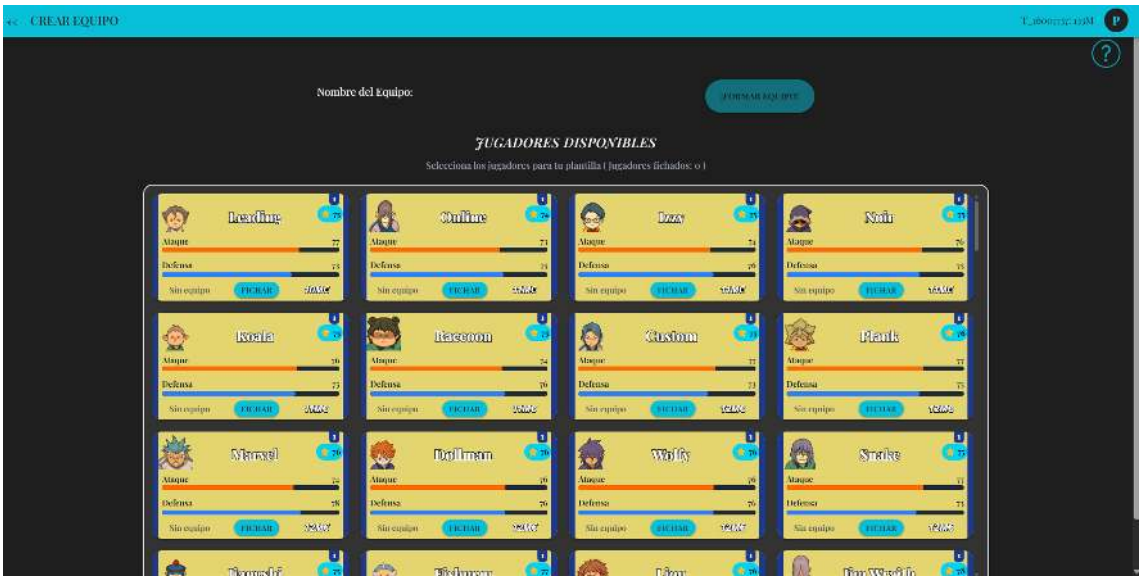
3.2.2 Ligas:



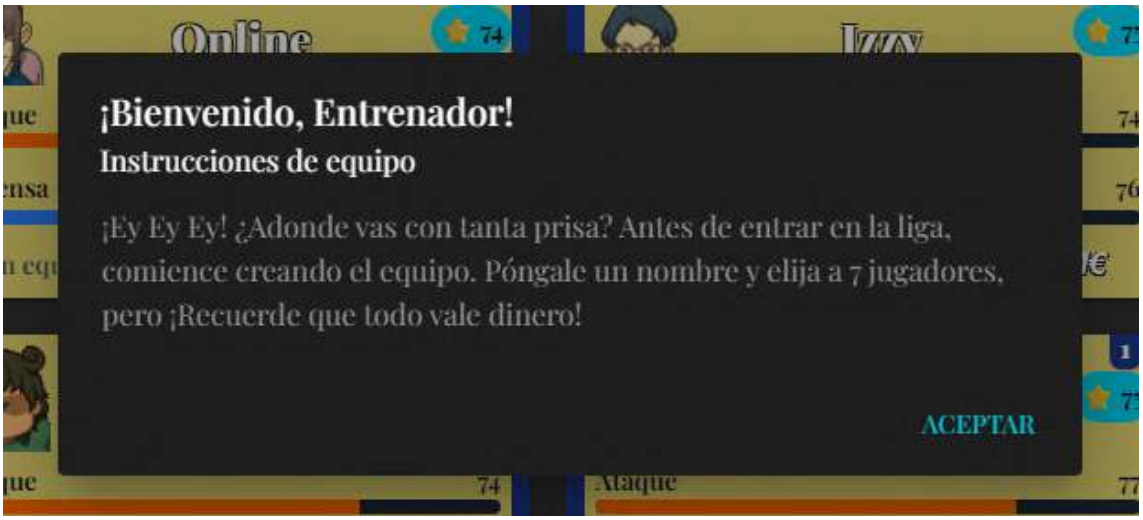
En ligas se tiene que poder manejar la creación de ligas, el poder unirte a ellas y poder borrarlas, hay que tener en cuenta que pueden haber muchas ligas, pero sólo puedes llegar a crear 3 por usuario. En caso de que sea la liga tuya, tiene que salir la opción de borrar tu liga, con una confirmación, en caso de error



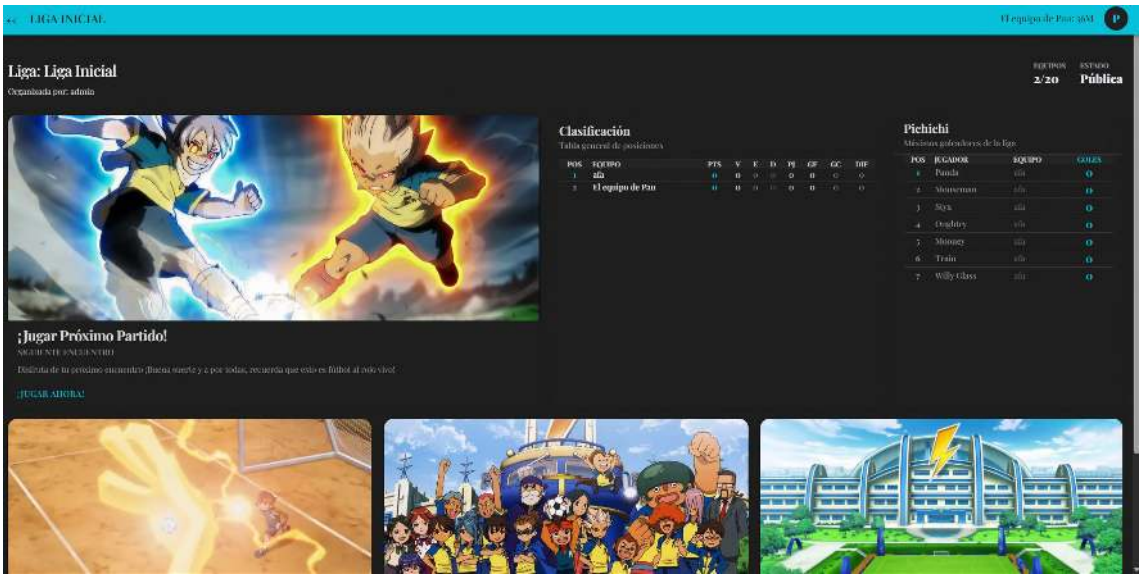
3.2.3 Crear Equipo:



Una vez te unas a una liga, te ha de salir la pantalla de crear equipo, en la cual has de poder ver toda la lista de jugadores disponibles, e ir navegando entre cual va mejor para tu equipo, también has de poder poner el nombre de tu equipo, con también comprobaciones de errores de escritura, como en el registro e inicio de sesión, y con un botón de ayuda arriba a la derecha, por si no entiendes alguna cosa.



3.2.4 La liga:



Aquí estará disponible la pantalla de la liga, se puede ir cambiando desde el apartado de ligas en caso de que estés en más de una liga. Aquí se ha de poder ver la clasificación de todos los equipos de la liga, con el sistema de puntos, y las siguientes opciones:

Jugar liga: Esto llevará a un partido contra otro equipo de la liga, en caso de que haya más de dos equipos en una liga, si en la liga hay un solo equipo, no se podrá jugar.

Clasificación: Aquí se ha de poder apreciar la propia clasificación, cuantos puntos tiene cada equipo, cual es la diferencia de goles, y al lado se ha de poder apreciar el pichichi de la liga, los jugadores con más goles en la susodicha liga.

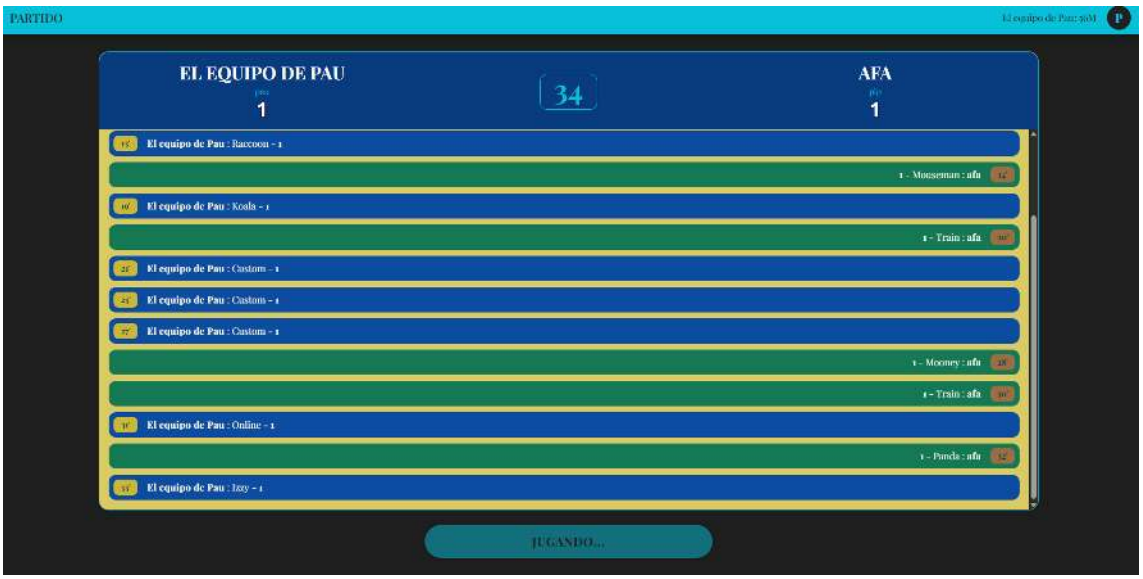
Torneo: Aquí te enviará a un partido contra un bot concreto, en el cual te has de batir a duelo equipo contra equipo.

Mercado de liga: Esto te llevará al mercado de la susodicha liga en cuestión, con 20 jugadores que ya habrán podido ser fichados, o no.

Gestionar Equipo: Aquí se puede ver el plantel del equipo, y puedes cambiar a tus jugadores de alineación, o venderlos si ya nos los quieres.

3.2.5 Partidos:

En la aplicación se pueden ver varios tipo de partidos (Torneos, Partidos de Liga, Amistosos...) todos esos partidos utilizan la misma pantalla, en la cual se puede apreciar el botón para empezar el partido, y asimismo verlo en tiempo real ocurrir



Durante el partido ha de tener:

Equipo Local: El equipo local ha de tener su nombre, el nombre de usuario propietario del equipo, y la cantidad de goles que lleva el equipo en ese minuto.

Minuto: en el medio debe estar el minuto en el que estamos ahora mismo.

Equipo visitante: Exactamente lo mismo que el equipo local, pero para el visitante, nombre del equipo, nombre del propietario y goles en ese momento del equipo.

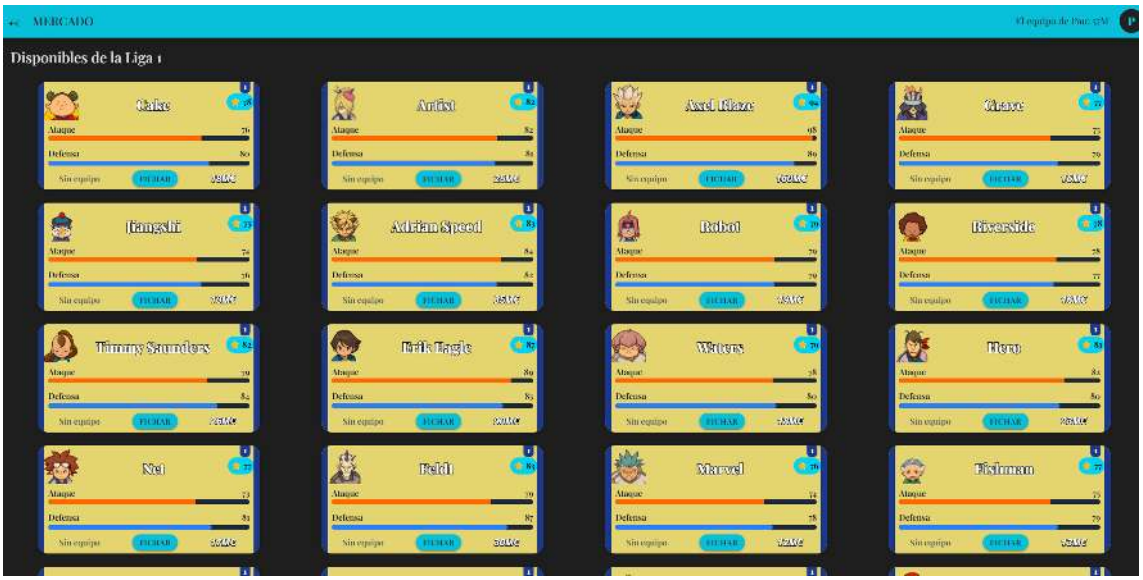
El Partido: Abajo, ha de estar lo que ocurre en cada momento con, el minuto del partido en el que está sucediendo, el nombre del equipo en el cual sucede algo, el nombre del jugador el cual esta haciendo la jugada, y si acaba en gol o no.

3.2.6 Equipo:

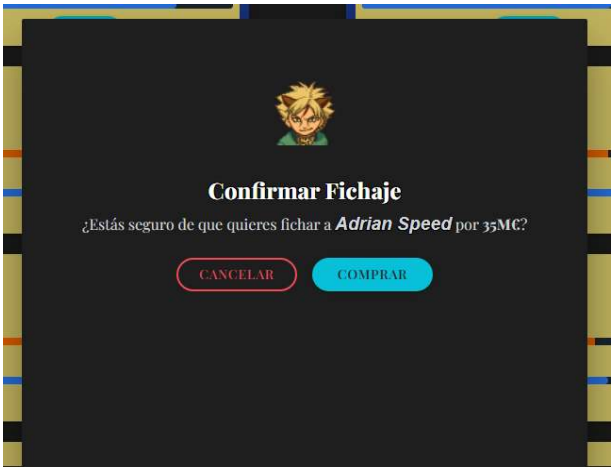
Para la pantalla del equipo, con Angular se usa Drag and Drop propio, el cual permite, con unos estilos concretos, cambiar a los jugadores de posición como si fuera un partido. pudiendo cambiar a los defensores de lado, o colocar al delantero estrella de portero.

También se ha de poder, tocando a un jugador, venderlo, con una pestaña para confirmar la venta.

3.2.7 Mercado:



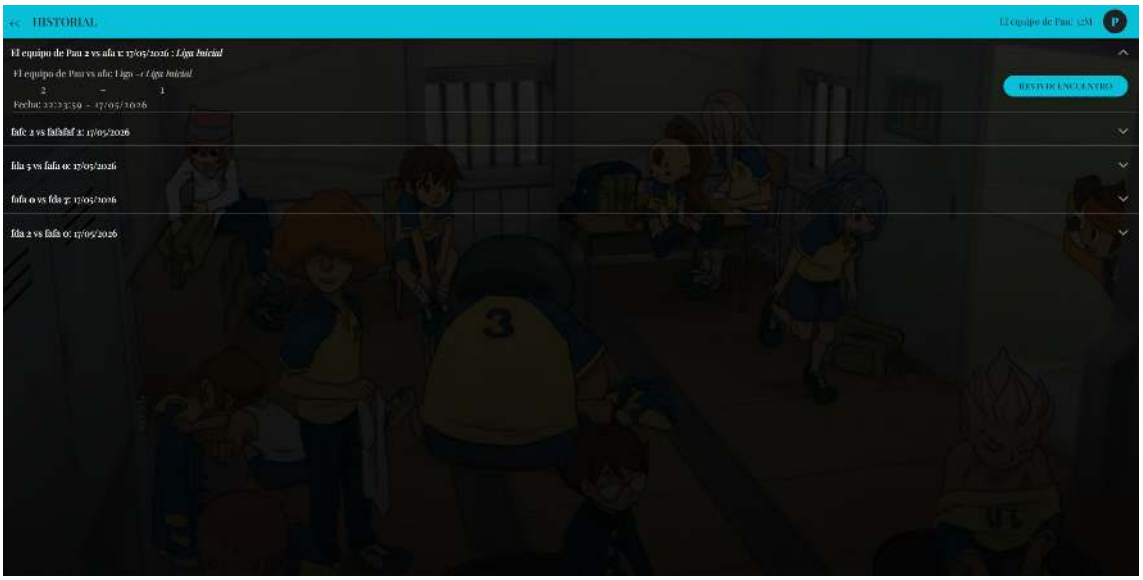
En el mercado, han de salir hasta 20 jugadores aleatorios, en los cuales puedas ver sus estadísticas (Cuanto ataque tienen, que tan bien defienden, y la valoración global de estos) a la par de el precio de cada uno.



También ha de haber un botón para poder ficharlos, con otra pantalla de confirmación, para evitar posibles errores.

Una vez fichado, la carta ya no se ha de poder fichar, y se tiene que explicar porque, por ejemplo, diciendo que ya tiene un equipo.

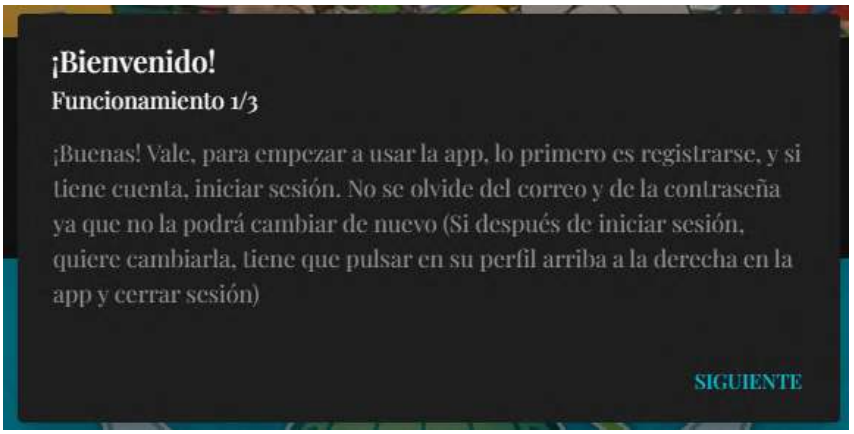
3.2.8 Historial:



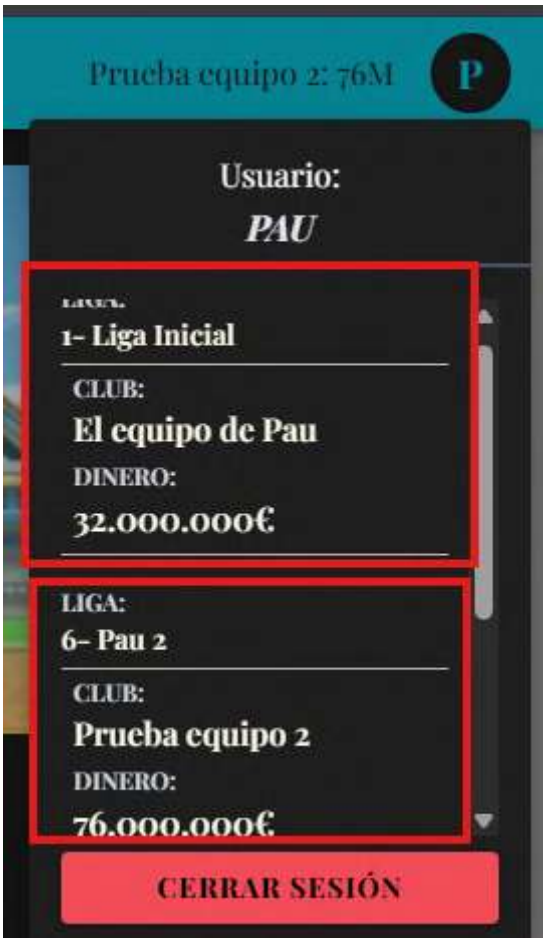
En esta pantalla sirve para ver todos los partidos jugados, con el momento en el que se vivió ese encuentro, los dos equipos implicados, y el resultado final. También ha de haber un botón que sirva, para ir a la pantalla de partidos, y revivir ese encuentro en el que ha dado al botón

3.2.9 Header:

En el header ha de poner el título de la página en la que te encuentras, para facilitar al usuario en qué página está, también, en el home concretamente ha de haber el botón de ayuda, que sirva específicamente para poder ver como funciona la aplicación en sí.



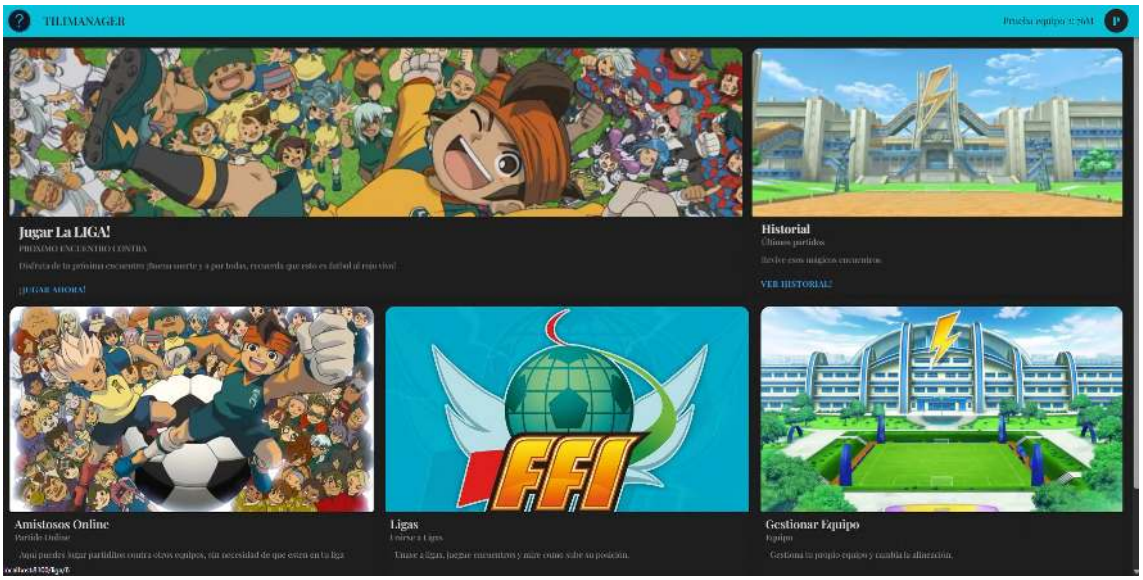
También, arriba a la derecha tiene que estar tu perfil, con la posibilidad de cambiar entre equipos si está en más de una liga, lo que permite gestionar los equipos del usuario de una manera bastante rápida y sencilla. También ha de haber un botón para cerrar sesión, y en caso de que no tenga sesión iniciada, un botón para iniciar sesión.



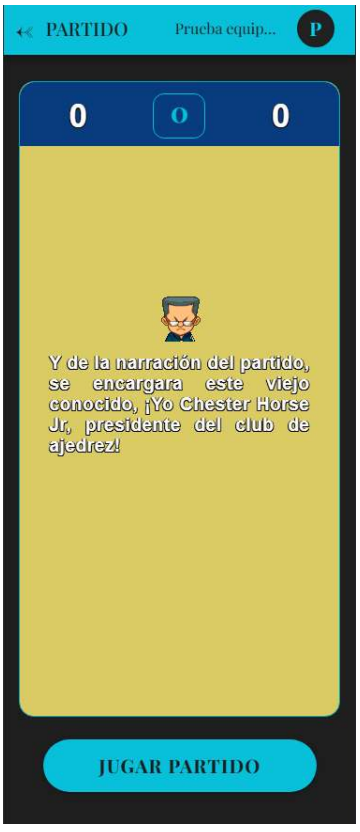
Equipo 1 y Equipo 2, de la Liga Inicial y liga Pau 2.

3.2.10 Responsive:

Toda la aplicación ha de ser funcional tanto como para pantallas de ordenador, como para dispositivos móviles, hay que tener en cuenta que la aplicación es multiplataforma, entonces alguien podría jugar un partido desde su ordenador con pantalla de 2160 pixeles contra otra persona que juegue en su móvil de 375 píxeles, es por eso que es muy importante que esté bien escalado.


TILIMANAGER
Jugar La LIGA!
Historial
Amistosos Online
Ligas
Gestionar Equipo


TILIMANA...
Prueba equip...
Jugar La LIGA!
PROXIMO ENCUENTRO CONTRA
Disfruta de tu próxima encuentro ¡Buena suerte y a por todas, recuerda que esto es futbol al rojo vivo!
¡JUGAR AHORA!
Historial
Últimos partidos
Revive esos mágicos encuentros
VER HISTORIAL!


PARTIDO
Prueba equip...
0
0
0
Y de la narración del partido, se encargara este viejo conocido, ¡Yo Chester Horse Jr, presidente del club de ajedrez!
JUGAR PARTIDO


EQUIPO
Prueba equip...
74
Panda
73
Oughtry
74
Styx
74
Mouseman
75
Train
74
Mooney
69
Willy Glass
Prueba equipo 2
PORTERO

4 Conclusiones

4.1 Conclusiones generales del proyecto

El desarrollo de este simulador y gestor de ligas de Inazuma Eleven ha permitido extraer lecciones muy valiosas sobre el diseño y la creación de aplicaciones interactivas. La principal conclusión es que el éxito de un sistema basado en datos y simulaciones depende por completo de una buena planificación previa en la estructura de la información, a la par que una muy buena gestión del tiempo. Conseguir que convivan un mercado de fichajes económicamente seguro y un motor de partidos dinámico demuestra que la clave de un buen software no está solo en una interfaz atractiva, sino en asegurar que los datos no se dupliquen y fluyan de forma rápida hacia el usuario.

En el plano académico y profesional, este proyecto ha supuesto un gran crecimiento. Ha servido para consolidar conceptos fundamentales como el control de transacciones para evitar fallos económicos, la automatización de tareas lógicas que dependen del reloj en tiempo real y el filtrado de datos antes de enviarlos a la pantalla. Además, se ha obtenido una visión global de cómo crear un producto desde cero, aportando herramientas clave para resolver problemas de organización de datos, optimización de pantallas para móviles (UX) y gestión fluida de eventos visuales.

4.2 Cumplimiento de los objetivos

A continuación, se detalla el nivel de cumplimiento de los objetivos planteados al inicio del proyecto:

Objetivo 1: Diseñar un sistema de ligas virtuales independientes donde los datos estén aislados.

Estado: Conseguido.

Argumentación: Se ha logrado que cada liga funcione como una burbuja cerrada. Un mismo futbolista del juego puede pertenecer a un equipo en la "Liga A" y estar libre en el mercado de la "Liga B", sumando goles y estadísticas de forma totalmente independiente en cada una de ellas.

Objetivo 2: Crear un mercado de fichajes automatizado y financieramente seguro.

Estado: Conseguido.

Argumentación: Las compras y ventas funcionan sin errores. El sistema impide que los clubes gasten más dinero del que tienen y bloquea las operaciones en el momento de la compra para que dos usuarios no

puedan adquirir al mismo jugador a la vez. Además, la oferta de futbolistas se renueva sola automáticamente.

Objetivo 3: Desarrollar un motor de simulación de partidos basado en eventos cronológicos.

Estado: Conseguido.

Argumentación: Se descartó la idea de ofrecer un resultado directo y frío. En su lugar, el sistema calcula el partido minuto a minuto mediante probabilidades, guardando cada suceso (goles, tarjetas, jugadas) en una línea de tiempo para que la pantalla pueda "reproducir" el encuentro como si fuera una película.

Objetivo 4: Construir una interfaz de usuario adaptativa, reactiva y tipo cuadro de mandos (Dashboard).

Estado: Conseguido.

Argumentación: La pantalla principal organiza de forma limpia una gran cantidad de datos (navegación, dinero del club y clasificaciones) sin agobiar al usuario. Al cambiar de liga en el menú desplegable, toda la información se actualiza al instante en cascada. Además, el diseño se adapta perfectamente a pantallas de móviles y tablets de forma cómoda y táctil.

4.3 Valoración de la metodología y planificación

El proyecto se ha llevado a cabo siguiendo un desarrollo por fases, lo que ha permitido probar y asegurar el funcionamiento de cada módulo antes de pasar al siguiente. Durante la primera mitad del calendario se cumplió el ritmo previsto en el diseño de las bases de datos y en la creación de las pantallas. Sin embargo, se detectó un retraso temporal al programar la lógica de partidos y las reglas del mercado, ya que aparecieron problemas de sincronización de datos y bucles internos que obligaron a dedicar más horas de pruebas de las esperadas.

Para garantizar el éxito del proyecto y cumplir con la fecha de entrega, se organizaron las tareas prioritarias. Se optó por simplificar algunas vistas secundarias del historial y centrar los esfuerzos en un sistema de limpieza de datos intermedio que aligerara la comunicación entre el servidor y las pantallas. Estos cambios en la estrategia fueron muy acertados, ya que solucionaron los errores inesperados de la red y mejoraron drásticamente la velocidad de la aplicación, logrando un resultado final muy estable y fluido.

4.4 Visión a futuro

Gracias a la estructura organizada y limpia del proyecto, quedan las puertas abiertas a nuevas mejoras para continuar expandiendo el simulador en el futuro:

- **Sistema de negociación directa:** Permitir que los usuarios intercambien o negocien jugadores directamente entre ellos haciendo contraofertas económicas, en lugar de depender únicamente de los futbolistas libres que ofrece el mercado general.
- **Gestión táctica avanzada:** Introducir la opción de elegir alineaciones formales (como un 4-3-3 o un 5-4-1), logrando que la posición natural de cada futbolista influya directamente en el éxito de las jugadas del motor de partidos.
- **Notificaciones en tiempo real:** Implementar avisos inmediatos para que los usuarios reciban alertas en sus dispositivos cuando otro jugador supere una oferta en el mercado o cuando empiece una nueva jornada de liga.
- **Inteligencia artificial para los rivales automáticos (Bots):** Mejorar las decisiones de los equipos controlados por el sistema para que realicen fichajes de forma autónoma basándose en sus necesidades deportivas y en el dinero que tengan disponible.

5. Glosario

Fantasy: Es un juego móvil, que trata sobre gestionar un equipo de fútbol basado en los equipos reales de LALIGA (Liga Española de fútbol de primera división), en la cual mediante se van jugando los partidos reales, se les pone nota a los jugadores, y dependiendo de qué tan bien jueguen, tendrás una mejor puntuación, o una peor.

Online Soccer Manager: Es otro juego móvil, en el cual también con jugadores reales, deberás crear tu equipo, entrenarlos y jugar partidos contra otros jugadores con sus respectivos equipos.

4-4-3: Formación de fútbol en la que hay 4 defensas, 4 mediocentros y 3 delanteros.

API REST: Interfaz que conecta el navegador y el servidor para intercambiar información como el mercado o las clasificaciones.

Backend: Parte lógica del servidor que procesa las simulaciones, gestiona el mercado y asegura los datos por detrás.

DTO (Data Transfer Object): Objetos intermedios que limpian y transforman los datos antes de enviarlos, protegiendo la información y agilizando la red.

Frontend: Parte visual y cuadro de mandos fosc donde el usuario interactúa, consulta la tabla y gestiona su club.

JSON: Formato ligero de texto utilizado para enviar los datos (saldos, plantillas, eventos) entre el servidor y el navegador.

Lógicas Transaccionales: Sistema de seguridad del mercado que actualiza saldos y propietarios a la vez, impidiendo dinero negativo o jugadores duplicados.

Motor de Simulación: Algoritmo que calcula el desarrollo de los partidos minuto a minuto basándose en estadísticas y probabilidades.

Plataforma Multiliga: Sistema que permite a un jugador tener estadísticas, dueños y valores independientes según la liga en la que compite.

Diseño Adaptable (Responsive): Maquetación que reordena y adapta menús y tablas automáticamente para que la web sea cómoda en móviles

6. Bibliografia

1. Documentación Angular: https://angular.dev/overview

2. Documentación Ionic: https://ionicframework.com/docs

3. Documentación PostgreSQL:

https://www.postgresql.org/docs/18/index.html

4. Aprender sobre Spring Boot:

https://www.youtube.com/@LaGeekipediaDeErnesto

https://www.youtube.com/watch?v=BRtPNl0aBkA&list=PLyvsggKtwbLVOPuOGn9J1Ie9RD7r7LcWD

5. Stack Overflow (Sitio para pedir ayuda a otros programadores):

https://stackoverflow.com

6. Sitio para diagramas:
https://mermaid.ai/app/projects

7. Documentación Spring Boot:
https://spring.io/projects/spring-boot

7 Anexos

Informe de usabilidad a un usuario de más de 50 años(para ver qué tan amigable es la aplicación para todo el mundo):
https://docs.google.com/spreadsheets/d/1-PIC-IA3cpSRsovOKcVqNm3OtZzW4okyib9rcelbo7s/edit?usp=sharing