DECKLY

Aplicación Web para gestión de torneos de juegos de cartas TCG

Proyecto de desarrollo
CFGS Desarrollo de Aplicaciones Multiplataforma

Eloy De La Cruz Andilla
Ismael Mehdi Zoukagh
2n DAM
2024 - 25

Resumen del proyecto y abstracción

Deckly es una aplicación web diseñada para la gestión eficiente de torneos de juegos de cartas coleccionables (TCG). La plataforma centraliza la inscripción, el seguimiento de resultados y la comunicación directa entre organizadores y participantes, superando las limitaciones de métodos tradicionales como WhatsApp y hojas de cálculo. Deckly ofrece una solución profesional y escalable que agiliza la organización de torneos al gestionar datos de manera eficiente y proporcionar una interfaz de usuario intuitiva y accesible para todos.

La aplicación permite a los organizadores crear y administrar torneos de forma eficaz, mientras que los participantes se benefician de un proceso de inscripción sencillo, el seguimiento del progreso y funcionalidades de interacción. Entre las características principales se incluyen el registro e inicio de sesión de usuarios, la creación de torneos, la inscripción, los perfiles personalizados y un potente buscador de torneos. Esta herramienta integral no solo simplifica las tareas administrativas, sino que también fomenta la profesionalización del sector TCG, impulsando su crecimiento y adaptación a las demandas evolutivas de la comunidad.

En la fase inicial, el proyecto empleó la metodología waterfall para establecer un cronograma claro y estructurado. A medida que avanzó el desarrollo, se incorporaron elementos de scrum para mejorar la asignación de tareas y la colaboración del equipo. Al combinar ambos enfoques, Deckly une una planificación exhaustiva con una ejecución ágil, garantizando una solución bien organizada y adaptable.

En última instancia, Deckly tiene como objetivo transformar la gestión de torneos de TCG mediante una plataforma moderna y centralizada que satisfaga las necesidades del mercado presentes y futuras.

Palabras clave

TCG.
Gestión de torneos.
Plataforma centralizada.
Microservicios.
Aplicación escalable.
Comunicación en tiempo real.

Project summary and abstract

Deckly is a web application designed for the efficient management of trading card game (TCG) tournaments. The platform centralizes registration, result tracking, and direct communication between organizers and participants, overcoming the limitations of traditional methods such as WhatsApp and spreadsheets. Deckly offers a scalable, professional solution that streamlines tournament organization by efficiently handling data and providing an intuitive user interface accessible to all.

The application enables organizers to create and manage tournaments effectively, while participants benefit from straightforward registration, progress tracking, and interaction features. Key functionalities include user registration and login, tournament creation, enrollment, personalized profiles, and a robust tournament search function. This comprehensive tool not only simplifies administrative tasks but also fosters professionalism in the TCG sector, encouraging growth and adaptation to the evolving demands of the community.

Initially, the project employed a waterfall methodology to set a clear, structured timeline. As development progressed, elements of scrum were introduced to improve task allocation and team collaboration. By blending both approaches, Deckly combines thorough planning with agile execution, ensuring a well-organized and adaptable solution.

Ultimately, Deckly aims to transform TCG tournament management by delivering a modern, centralized platform that meets current and future market needs.

Keywords

TCG.
Tournament management.
Centralized platform.
Microservices.
Scalable application.
Real-time communication.

Índice

1.- Introducción

El propósito que tiene el proyecto es el de desarrollar una aplicación especializada en la gestión y organización de torneos de juegos de cartas coleccionables (***TCG***).

La aplicación está diseñada para centralizar la administración de inscripciones, seguimiento de resultados y comunicación directa entre organizadores y jugadores, superando las limitaciones de los métodos tradicionales (como el uso de *WhatsApp* y hojas de cálculo).

El enfoque que se le quiere dar a la aplicación es el de la escalabilidad y la profesionalización, ya que es una carencia del sector que se puede llegar a suplir con esta solución.

Con todo esto se busca optimizar el proceso de organización de torneos, permitiendo un manejo más eficiente de los datos y ofreciendo una experiencia de usuario intuitiva y moderna a la que todo el mundo pueda tener acceso.

1.1.- Contexto y Justificación

En la actualidad, la gestión de torneos de juegos de cartas coleccionables (*TCG*) todavía se realiza de forma rudimentaria y con herramientas no especializadas que incrementan la probabilidad de errores.

Estos últimos años los *TCG* han experimentado un gran crecimiento en su base de jugadores, gracias al coleccionismo de ellas, pero gran parte de esta gente, se ha acabado aficionando al juego.

Es por eso que la dependencia de procesos manuales y división de la información llega a generar desconfianza en los usuarios y dificulta la escalabilidad de los eventos.

Tras identificar estas carencias, el proyecto plantea una solución integral que centraliza datos y agiliza la comunicación entre tiendas organizadoras y jugadores.

La implementación y uso de esta aplicación ayudará a la profesionalización en los procesos de administración en los diferentes torneos que se jueguen, e incluso también se fomentará la expansión del mercado al ofrecer una herramienta adaptable a las crecientes demandas del sector.

1.2.- Objetivos

El objetivo principal de este proyecto es informatizar las tiendas en el ámbito de la creación y gestión de torneos, ya que puede aportar varios beneficios significativos, como pueden ser la eficiencia y la precisión, relacionado con reducir errores humanos y/o pérdida de datos, o la centralización de todos los juegos en una misma aplicación web.

Un listado de objetivos específicos son los siguientes:

- Registro e inicio de sesión
- Capacidad de crear torneos por parte del organizador
- Posibilidad de unirse a torneos en el bando de los jugadores
- Perfil personal para los dos tipos de usuarios
- La implementación de un buscador de torneos

Estos son algunos de los objetivos que abarcaría la aplicación, pero no son los únicos ya que conforme el proyecto ha ido evolucionando, se han encontrado nuevas implementaciones a hacer al proyecto que se han acabado sumando a éste.

1.3.- Estrategia y planificación del proyecto

El proyecto se concibió ante la ausencia de aplicaciones especializadas en la organización de torneos *TCG*, más allá de herramientas parciales como *Companion* para *Magic: The Gathering* o *RK9* para *Pokémon* (las cuales son eficaces para sus juegos pero no tienen una visión general), detectando una oportunidad para que Deckly ocupe ese espacio. Hasta la fecha, iniciativas similares se han quedado en soluciones parciales o han resultado ineficaces.

Con *Deckly* se quiere llevar todo este tema más allá, ayudando con la gestión de torneos y la centralización de la información y de todos los juegos, además de informatizar las tiendas y dar un paso más hacia el cambio en el sector.

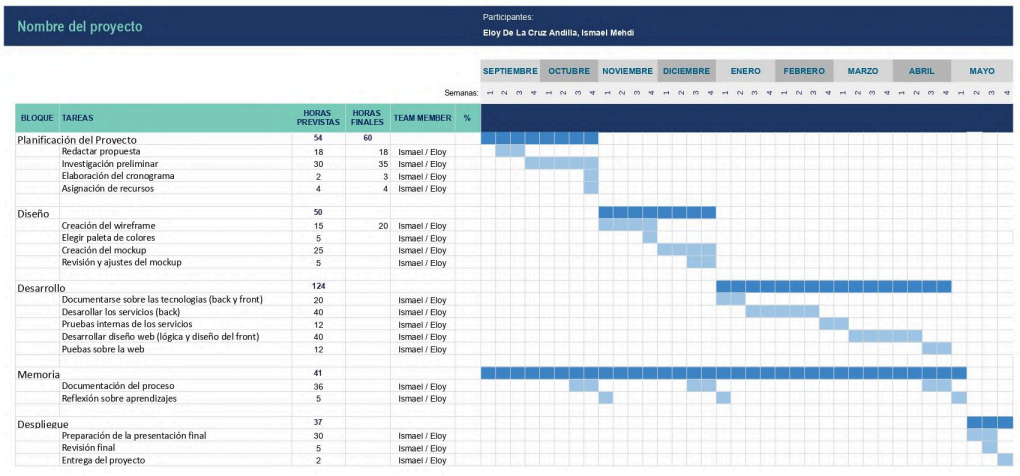
La propuesta de *Deckly* no solo resuelve las deficiencias de los métodos tradicionales, sino que también sienta las bases para un ecosistema escalable y adaptable a futuros requerimientos del sector. Todo esto es gracias a que utiliza una arquitectura moderna basada en microservicios la cual permite traer el producto de esta forma tan ampliable, ofreciendo así una experiencia de usuario superior y transformando la forma en que se gestionan los torneos de *TCG*.

1.4.- Metodología de trabajo

Cuando se habla de metodologías de trabajo en el ámbito de la informática o de la programación, se encuentran dos que reinan el mercado: la metodología ***scrum*** y la ***waterfall***. Para poner en contexto cada una de estas metodologías, waterfall es el enfoque tradicional, el cual se basa en una secuencia lineal y secuencial de actividades, mientras que la metodología scrum es un enfoque más flexible y colaborativo, el cual se basa en la entrega incremental y ciclos cortos de desarrollo.

En los últimos años la metodología *waterfall* ha ido en decadencia debido a esta nueva forma de trabajo y organización, además de que existen otras metodologías parecidas a *scrum*, como pueden ser ***kanban*** o ***agile*** (muy parecida a *scrum*), y las empresas están enfocando sus proyectos a esta manera de trabajar, ya que el sector no para de evolucionar y hay que estar preparado para los cambios repentinos.

En una primera fase se optó por la metodología *waterfall*, dado que el plazo de entrega era fijo y resultaba necesario planificar con detalle cada etapa temporal.



Nombre del proyecto

Participantes:
Eloy De La Cruz Andilla, Ismael Mehdi

SEPTIEMBRE | OCTUBRE | NOVIEMBRE | DICIEMBRE | ENERO | FEBRERO | MARZO | ABRIL | MAYO

Semanas: 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4

BLOQUE	TAREAS	HORAS PREVISTAS	HORAS FINALES	TEAM MEMBER	%
Planificación del Proyecto		54	60		
	Redactar propuesta	18	18	Ismael / Eloy	
	Investigación preliminar	30	35	Ismael / Eloy	
	Elaboración del cronograma	2	3	Ismael / Eloy	
	Asignación de recursos	4	4	Ismael / Eloy	
Diseño		50			
	Creación del wireframe	15	20	Ismael / Eloy	
	Elegir paleta de colores	5		Ismael / Eloy	
	Creación del mockup	25		Ismael / Eloy	
	Revisión y ajustes del mockup	5		Ismael / Eloy	
Desarrollo		124			
	Documentarse sobre las tecnologías (back y front)	20		Ismael / Eloy	
	Desarollar los servicios (back)	40		Ismael / Eloy	
	Pruebas internas de los servicios	12		Ismael / Eloy	
	Desarrollar diseño web (lógica y diseño del front)	40		Ismael / Eloy	
	Puebas sobre la web	12		Ismael / Eloy	
Memoria		41			
	Documentación del proceso	36		Ismael / Eloy	
	Reflexión sobre aprendizajes	5		Ismael / Eloy	
Despliegue		37			
	Preparación de la presentación final	30		Ismael / Eloy	
	Revisión final	5		Ismael / Eloy	
	Entrega del proyecto	2		Ismael / Eloy	

Figura 1. Imagen del cronograma hecho con metodología *waterfall.*

Matizar también que, cuando se entra en la fase de desarrollo, se utilizó la metodología *scrum* para el reparto de tareas, pero su uso fue más bien para que cada miembro del equipo se comprometa con sus tareas y era una forma más de organización. Cada uno de los sprints se compone de dos semanas cada uno.

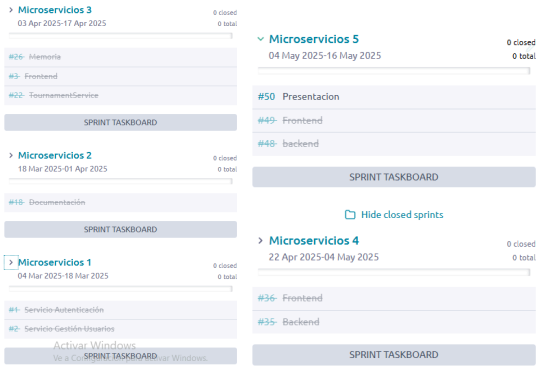


Figura 2. Imagen de los *sprints* realizados.

La adopción de *waterfall* permitió establecer un cronograma claro para todo el desarrollo, mientras que la posterior incorporación de *scrum* facilitó la distribución de tareas y el seguimiento continuo del progreso.

Se pensó en utilizar *scrum* antes, pero al no saber las problemáticas a enfrentar, se decidió apostar por el método tradicional, y bien entrados en el desarrollo, se hizo esta nueva implementación de metodología.

2.- Descripción del proyecto

2.1.- Análisis de requerimientos

- Login/Registro.
- Perfil de usuario: mostrar y permitir modificar datos de tu perfil.
- Creación de torneos (organizador).
- Búsqueda de torneos mediante filtros como la localidad (jugador) .
- Mostrar información del torneo creado.
- Inscripción a torneos (jugador).
- Con el torneo iniciado el organizador debe poder seleccionar los jugadores que pasan de ronda.
- Todos los participantes pueden ver los cambios en el torneo en tiempo real.

2.2.- Requisitos no funcionales.

- Interfaz intuitiva.
- Autenticación mediante ***JWT***.
- Cifrado de contraseñas.
- Multiplataforma (Web, Android)

2.3.- Tecnologías

2.3.1.- Tecnologías propuestas

Al inicio, durante la planificación del proyecto se consideraron varias opciones para diferentes aspectos del proyecto, donde entraban el ***frontend***, ***backend***, base de datos y ***UX/UI*** . A continuación veremos qué tecnologías y software fueron candidatos para los aspectos mencionados anteriormente.

2.3.1.1.- Frontend

Para la parte frontal o visual del proyecto, que es con la que interactúa el usuario final, se investigaron dos tecnologías las cuales son de las más demandadas del mercado actualmente: *React* y *Vue* con *Ionic*.

Criterio	***React***	***Ionic + Vue***
Facilidad de aprendizaje	Sintaxis JSX puede ser confusa al inicio, pero tiene gran documentación.	Vue es más intuitivo y con una curva de aprendizaje más suave.
Ecosistema	Amplio, con Next.js, React Native y muchas bibliotecas de terceros.	Vue es más estructurado, e Ionic proporciona componentes móviles listos para usar.
Orientación del framework	Enfocado en aplicaciones web con flexibilidad para móviles	Diseñado para aplicaciones móviles híbridas y web.
Manejo del estado	Redux, Zustand, Context API (puede ser más complejo de manejar).	Vuex o Pinia, generalmente más simples y con mejor integración nativa.
Estilo y UI	Usa CSS-in-JS, Styled Components o Tailwind	Ionic ofrece un set de componentes UI predefinidos con estilos adaptables a iOS y Android.
Rendimiento	Muy rápido gracias a Virtual DOM, pero puede requerir optimización manual.	Vue + Ionic es eficiente, aunque el uso de WebView en móviles puede ser un poco más pesado.
Modularidad y escalabilidad	Más flexible, pero requiere estructurar el código manualmente para escalar bien.	Más estructurado con Vue y el sistema de componentes de Ionic.
Comunidad y soporte	Muy grande, con muchos recursos, pero también con muchas formas distintas de hacer lo mismo.	Comunidad creciente, documentación clara y un ecosistema más unificado.

2.3.1.2.- Backend

El *backend* es donde se encuentra toda la lógica de negocio y se encargará de recibir, procesar y devolver datos. Para construir la ***API*** del proyecto se investigaron las 3 opciones más usadas a día de hoy en la parte del servidor: *Spring Boot*, *Express.js* y *ASP.NET*.

Criterio	***Spring Boot***	***Express.js***	***ASP.NET Core***
Curva de aprendizaje	Alta (ecosistema amplio, convenciones propias de Spring). Ideal para desarrolladores con experiencia en Java.	Moderada (JavaScript/TS accesible, pero gestión asíncrona y middlewares añaden complejidad).	Moderada-alta (C# es intuitivo, pero patrones específicos de .NET y Entity Framework tienen su carga).
Arquitecturas	Soporte nativo para MVC, microservicios (Spring Cloud), y eventos. Ideal para sistemas complejos.	Flexible para APIs REST y microservicios, pero sin convenciones fuertes. Riesgo de código desorganizado en proyectos grandes.	Soporte robusto para MVC, Clean Architecture y microservicios. Integración sencilla con Azure.
Escalabilidad	JVM optimizada para concurrencia. Spring WebFlux permite escalamiento reactivo.	Escalado horizontal eficiente, pero limitado por el modelo single-thread de Node.js.	Alto rendimiento multihilo. Async/await bien integrado. Mejor desempeño en carga sostenida.
Soporte para bases de datos	JPA/Hibernate simplifica SQL. Amplio soporte para NoSQL.	ORMs disponibles (Sequelize, Mongoose), pero requieren configuración manual. Menos integración nativa.	Entity Framework Core ofrece alto nivel de abstracción con LINQ. Soporte estable para SQL/NoSQL.
Seguridad	Spring Security es potente pero complejo de configurar. OAuth2/JWT bien soportados.	Depende de middlewares externos (Passport.js). Requiere más trabajo para implementar seguridad robusta.	Identity Framework integrado. Soporte nativo para OAuth2, políticas de autorización claras.

2.3.1.3.- Base de datos

La base de datos es la herramienta que se encarga de almacenar de forma física toda la información de nuestro proyecto. Investigamos tres bases de datos, dos son ***SQL*** relacionales: *MySQL*, *PostgreSQL* y una es ***NoSQL***: *MongoDB*. Estas tres opciones constan en el top 5 actual de las bases de datos más usadas en el mundo.

Criterio	***MySQL***	***PostgreSQL***	***MongoDB***
Modelo de datos	Relacional (tablas y SQL)	Relacional (más flexible y avanzado que MySQL)	NoSQL (documentos JSON)
Velocidad en consultas	Muy rápido en lecturas y consultas simples	Óptimo para consultas complejas y grandes volúmenes de datos	Muy rápido en búsquedas flexibles y datos semi estructurados
Integridad de datos	Alta (claves foráneas y restricciones)	Muy alta (mejor manejo de transacciones y ACID)	Baja (no tiene integridad referencial nativa)
Casos de uso	Aplicaciones web tradicionales, CMS, eCommerce	Data warehouses, sistemas bancarios, análisis de datos	Aplicaciones en tiempo real, big data, microservicios
Facilidad de uso	Fácil de aprender y configurar	Más complejo, pero más potente	Fácil de usar si vienes de JavaScript
Popularidad y comunidad	Muy popular, gran comunidad	Menos popular que MySQL.	Popular en el mundo NoSQL, gran soporte en startups
Licencia	Open-source (con versión comercial: MySQL Enterprise)	Open-source (PostgreSQL License, más permisiva)	Open-source (MongoDB SSPL, con restricciones en la nube)

2.3.1.4.- UI/UX

Es importante tener una herramienta de diseño adecuada para poder trabajar sobre ello. Es por eso que se investigó sobre *Figma* y *Photoshop*, para ver cual de estas herramientas se ajusta mejor a las necesidades del proyecto.

Criterio	***Figma***	***Photoshop***
Tipo de herramienta	Diseño UI/UX, prototipado, diseño interactivo, ideal para web y móvil.	Edición de imágenes, retoque fotográfico (menos centrado en UI/UX).
Colaboración	Excelente, trabajo en tiempo real, varios diseñadores pueden editar al mismo tiempo.	No tiene colaboración en tiempo real. Se puede trabajar en la nube, pero no es su fuerte.
Facilidad de uso	Fácil de aprender, especialmente para el diseño de interfaces web/móviles.	Más complejo, pero muy poderoso para edición de imágenes detalladas.
Prototipado	Excelente, permite crear prototipos interactivos con transiciones y animaciones.	No tiene herramientas de prototipado nativas (se debe usar software adicional).
Accesibilidad	Totalmente basado en la web (trabaja en cualquier dispositivo con acceso a internet).	Necesita ser instalado y es más pesado en recursos (requiere PC o Mac potente).
Plugins	Amplio ecosistema de plugins, especialmente para UI/UX y prototipado.	Gran cantidad de plugins, pero más enfocados en edición de imágenes.
Costo	Ofrece una versión gratuita.	Software de pago.

2.3.2.- Tecnologías elegidas

2.3.2.1.- *Frontend*: *VUE + IONIC*

Para el frontend, finalmente se consideró trabajar con *Vue* e *Ionic*, principalmente por la compatibilidad con librerías que resultan útiles para el desarrollo del proyecto; su sencilla curva de aprendizaje; tecnologías bien estructuradas; gran capacidad y facilidad para la paginación, además de que es una gran opción para una aplicación multiplataforma.

Se descartó directamente la opción de usar *HTML*, *CSS* y *JS* sin ningún tipo de ***framework*** porque, existiendo herramientas especializadas en solucionar y mejorar esta parte del código e ir más rápido en el desarrollo de la parte frontal de la aplicación, era esencial para adoptar un ritmo consistente en esta parte del proyecto.

2.3.2.2.- *Backend*: *SPRING BOOT*

Tras el análisis de las opciones disponibles, se terminó de confirmar *Spring Boot* como el *framework* principal para el *backend*. Se tuvo en cuenta su facilidad de integración con bases de datos; su soporte nativo para la arquitectura de microservicios; una buena escalabilidad, ya que es un framework diseñado para programas de gran volumen, y por último ofrece un sistema de seguridad potente (***Spring Security***), que pese a su complejidad, se simplifica gracias a las opciones que ofrece la tecnología.

2.3.2.3.- Base de datos: *MYSQL y MONGODB*

Para la base de datos, *MySQL* es la escogida por la confiabilidad, la fácil integración con *Spring Boot,* muy rápida para consultas simples además de que tiene una curva de aprendizaje muy buena. Debido a que es una base de datos muy utilizada en el mercado, muchos servicios de hosting tienen esta base de datos incorporada.

Además de *MySQL*, se incorporará *MongoDB* en uno de los microservicios (servicio de emparejamiento), dado que el modelo de documentos de *MongoDB* permite almacenar la estructura de bracket y los resultados de las rondas de forma más eficiente y flexible. Al tratarse de datos semiestructurados y de crecimiento dinámico, *MongoDB* facilita las consultas sobre colecciones de partidas y rondas sin necesidad de redefinir esquemas rígidos, lo que agiliza tanto el desarrollo como el rendimiento en las operaciones de lectura y escritura.

2.3.2.4.- *UI/UX*: *FIGMA*

La herramienta seleccionada para llevar a cabo el diseño de la web, teniendo en cuenta *UI/UX* ha sido *Figma*, ya que ofrece muchas facilidades para trabajar en equipo (permite editar al mismo tiempo y tiene guardado en la nube), y es una herramienta que existe en entorno web, y no es necesario descargar ningún programa. Además de todo esto, se pueden crear prototipos interactivos y ofrece una gran accesibilidad, ya que se puede trabajar desde cualquier dispositivo.

2.4.- Estructura del proyecto

Este proyecto ha sido diseñado bajo una arquitectura desacoplada y basada en microservicios, lo que permite dividir el sistema en componentes pequeños, independientes y escalables que interactúan entre sí de manera eficiente además de que aporta una facilidad en cuanto al mantenimiento de estos. En lugar de seguir un modelo convencional como lo es el monolítico, donde toda la lógica del negocio se encuentra integrada en un único bloque, se ha optado por una arquitectura microservicios.

2.4.1.- ¿Qué son los microservicios?

Los microservicios son una arquitectura de *software* donde una aplicación se divide en pequeños servicios independientes que pueden ser desplegados, desarrollados y gestionados de manera autónoma. Cada microservicio está enfocado en una única funcionalidad o conjunto de funcionalidades específicas dentro de la aplicación, y se comunica con el cliente (*Frontend*) a través de ***API Restful***. Cada microservicio puede tener su propia base de datos y lógica de negocio, lo que permite un desarrollo y despliegue más rápido.

2.4.2.- ¿Por qué usar microservicios?

En el diseño temprano del proyecto se dio el caso en el que habían muchas ideas interesantes; implementar algunas de ellas era un gran desafío para un desarrollo temprano. En vez de descartar estas ideas, se investigó qué arquitectura permitiría tener una demo inicial preparada para ser escalable. Ahí es donde entra esta arquitectura que, aparte de ofrecer esa escalabilidad autónoma por servicio, ofrece otros más beneficios, donde algunos de ellos serían la buena tolerancia a fallos: si un servicio falla y se cae el resto de la aplicación, seguirá funcionando siempre y cuando no utilice funcionalidades del servicio que se ha caído. Otra ventaja a destacar es que dos desarrolladores se pueden dedicar a desarrollar microservicios diferentes de forma autónoma permitiendo no estar controlando que hará el otro desarrollador para evitar conflictos.

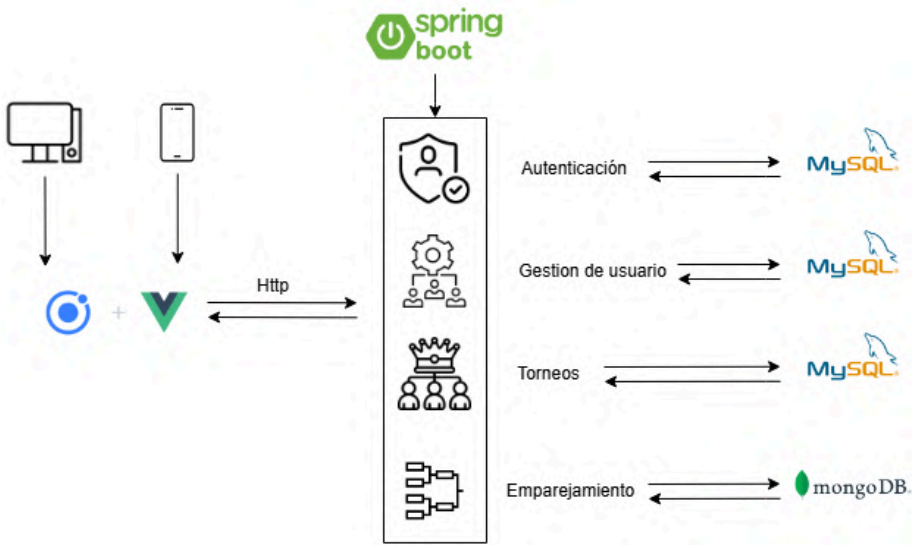


Figura 3. Estructura del proyecto separada en microservicios y las tecnologías implementadas.

2.5.- Componentes

Este apartado va destinado a hablar de los componentes que integran el proyecto, separando la explicación por microservicios. Todos los microservicios siguen una misma estructura, que se divide en varias carpetas: modelo, servicio, entidades, repositorio y configuración.

2.5.1- Autenticación

Este microservicio es responsable de la autenticación del sistema. Permite a los usuarios registrarse y autenticarse verificando y validando las credenciales del usuario sin importar el rol. Para la seguridad de este microservicio se utiliza *Spring Security* y *JWT* para la autenticación y mantener sesiones abiertas.

2.5.1.1.- Cómo funciona JWT con *Spring Security*

2.5.1.1.1.- Registro de usuario

El primer paso en el microservicio de autenticación es permitir que los usuarios se registren. Para ello, el sistema debe aceptar las credenciales del usuario y almacenarlas de manera segura.

Spring Security puede utilizar ***PasswordEncoder*** para el cifrado de contraseñas. Es importante remarcar que se cifra antes de guardarse en la base de datos, lo que garantiza que nunca se almacenarán contraseñas en texto plano.

Al registrarse, se crean los datos del usuario en una base de datos *MySQL*, pero aún no hay un ***token*** de *JWT*, ya que solo se genera al momento de la autenticación.

2.5.1.1.2.- Inicio de sesión

Cuando el usuario ya está registrado, puede iniciar sesión. Este proceso involucra verificar las credenciales y, si son correctas, generar un *token* para el usuario autenticado. Este token se firma con una clave secreta propia y se le envía al usuario. Este se almacenará temporalmente en el navegador y cada vez que el usuario trate de hacer una petición a un ***endpoint*** protegido, junto a la consulta deberá enviar en el encabezado ***Authorization*** el token que se le generó previamente al iniciar sesión.

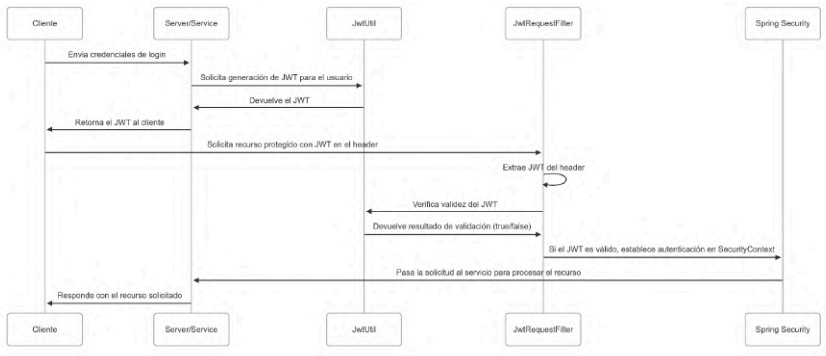


Imagen ampliada: Diagrama de secuencia JWT.png

Figura 4. Diagrama de secuencia de autenticación con *JWT* en el proyecto.

2.5.1.2.- Base de Datos

En la base de datos para el microservicio de autenticación existe una una única tabla en *MySQL* que se encarga de almacenar estas credenciales para el inicio de sesión. Aparte de las credenciales para la correcta identificación del usuario, se almacena el rol del usuario, ya sea para los *player* y *organizer***.** Esto es importante ya que permitirá acceder a diferentes vistas y funcionalidades dependiendo del rol.

El algoritmo utilizado para el cifrado de la contraseña es ***BCrypt*** con la clase ***BCryptPasswordEncoder*****.**

USER			
Long	id	PK	
String	role		not null
String	password		not null
String	email		unique, not null
LocalDateTime	createdAt		not null
String	username		unique, not null

Figura 5. Tabla *User* del microservicio ***AuthService***.

2.5.1.3.- Estructura general del microservicio *AuthService*

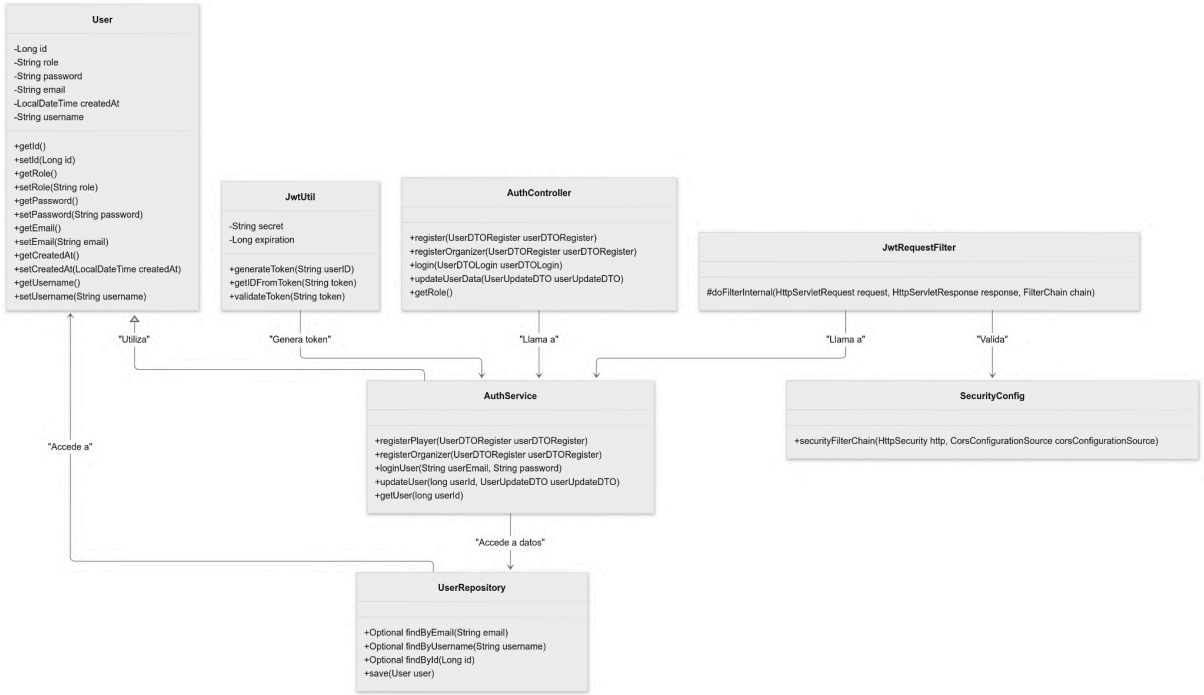


Imagen ampliada: Diagrama de clases authservice.png

Figura 6. Diagrama de clases de la estructura general del micro servicio *AuthService*

2.5.2.- Gestión de usuario

El objetivo de este microservicio es gestionar toda la información relacionada con los jugadores y organizadores. Permite crear, leer, actualizar y eliminar información de los usuarios a través de una ***API Rest***, interactuar directamente con la base de datos para almacenar y recuperar información de los usuarios. Además este servicio se encarga de guardar imágenes localmente en el servidor, las cuales son utilizadas por los usuarios como imagen de perfil.

2.5.2.1.- Gestión y almacenamiento de la información de los usuarios

Cuando un usuario se registra en la aplicación, este añade más credenciales o información de la que el servicio *AuthService* almacena. Esto se debe a que dicho servicio solo guarda los datos mencionados anteriormente (ver Diagrama ER, tabla *User* del microservicio AuthService).

Cuando se quiere registrar un usuario desde la aplicación, en realidad se están realizando dos peticiones:

1. La primera es al *AuthService*. Una vez que este valida y almacena los datos, devuelve un código 200, es decir, da el "ok" para continuar.

2. La segunda petición envía el resto de los datos del usuario, junto con el *token* en el encabezado (*Authorization*). Esta información es recibida por el servicio que gestiona los datos del usuario, llamado ***UserManagementService***, asegurándose de que el ID del usuario sea el mismo tanto en *AuthService* como en *UserManagementService*.

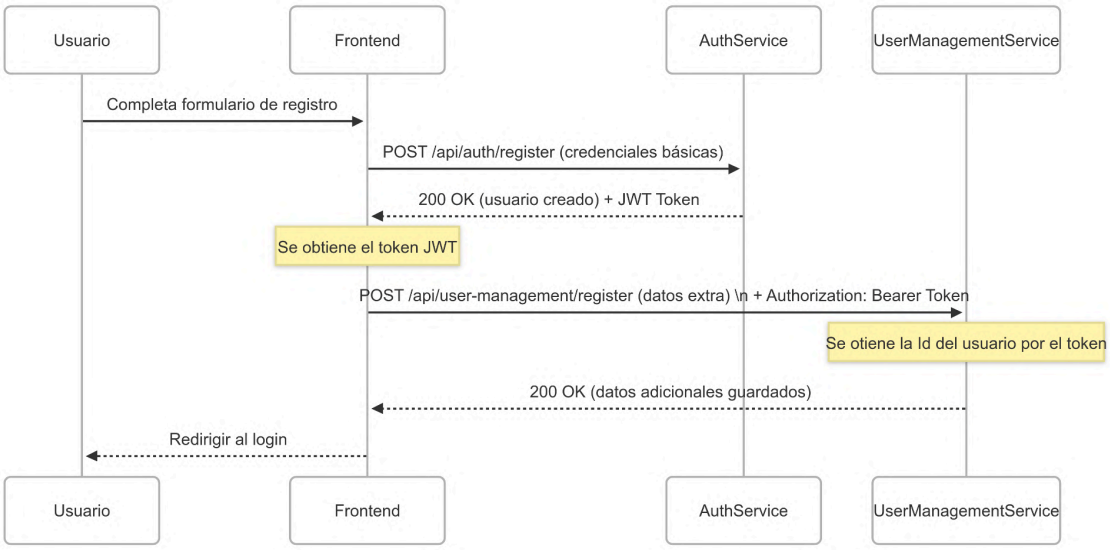


Imagen ampliada: Diagrama de secuencias registro de usuario.png

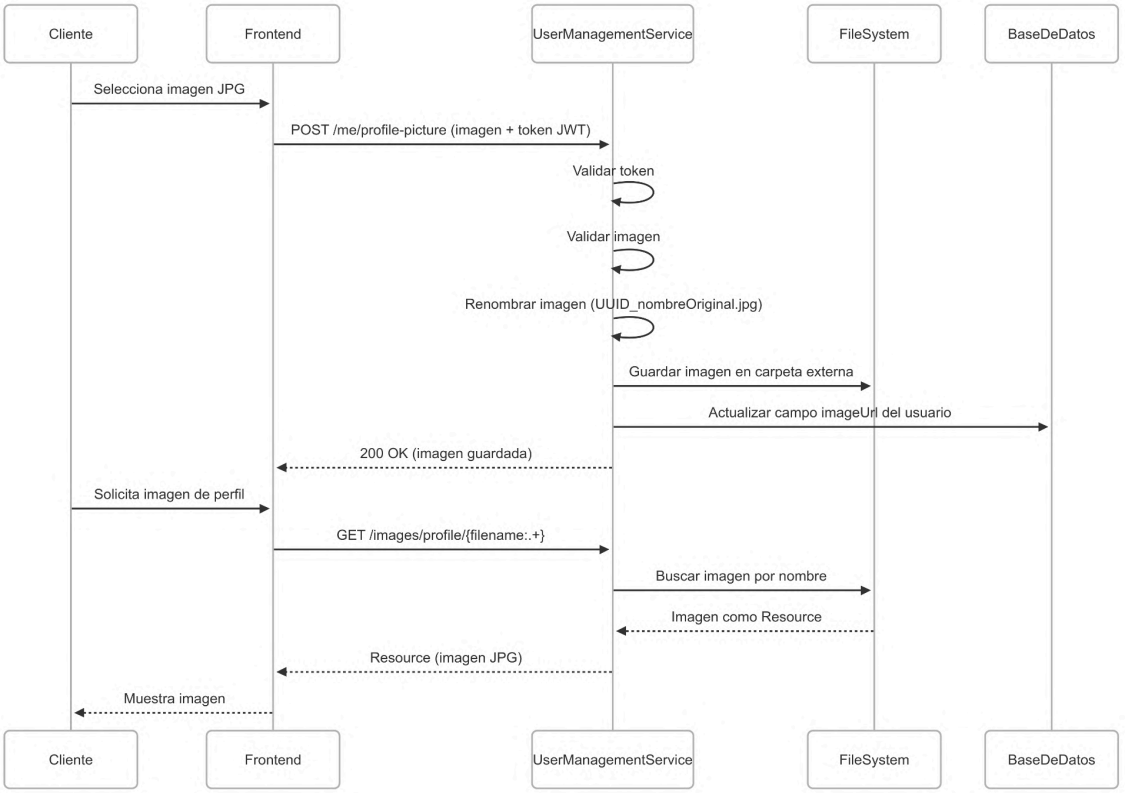
Figura 7. Interacción entre *AuthService* y *UserManagementService* para registrar un usuario.

2.5.2.2.- Almacenado y recuperación de imágenes

La imagen se manda en formato *JPG* desde el cliente, por protocolo ***HTTP*** más las validaciones de seguridad que se mencionaron anteriormente, el microservicio se encarga de darle un nombre nuevo al fichero: *"uuid random_nombre original del archivo".* Este nombre se almacena en la base de datos junto a los datos de ese propio usuario (columna *imageUrl*) y después se almacena la imagen localmente en el servidor.

Aclarar que la carpeta que almacena las imágenes no está dentro de ningún microservicio, si no que se guardan en disco.

Para recuperar la imagen hay un *endpoint* con un método ***GET*** preparado en ***UserController*** al que se le envía por ***params*** el nombre del archivo, donde justamente es el nombre que tienes almacenado en la base de datos en el campo *image_url***.** Este método busca la imagen por el nombre en el fichero donde se almacenan las imágenes y la devuelve como un ***Resource.***



Ampliar imagen: Diagrama de secuencias subida y recuperacion de imagenes.png
Figura 8. Almacenamiento y recuperación de imágenes.

2.5.2.3.- Base de datos

En la base de datos del microservicio de gestión de usuarios nos encontramos con una tabla con especializaciones: una tabla padre *User* de la que heredan información las tablas *organizer* y *player*, dado que de esta manera se consigue la división de roles. La razón de la estructura es que ambos roles comparten muchos campos, pero hay ciertos campos donde solamente unos de los roles lo alberga, y no se encuentra en el otro. Es por eso que es conveniente seguir esta estructura. Todo esto está almacenado en una base de datos *MySQL*.

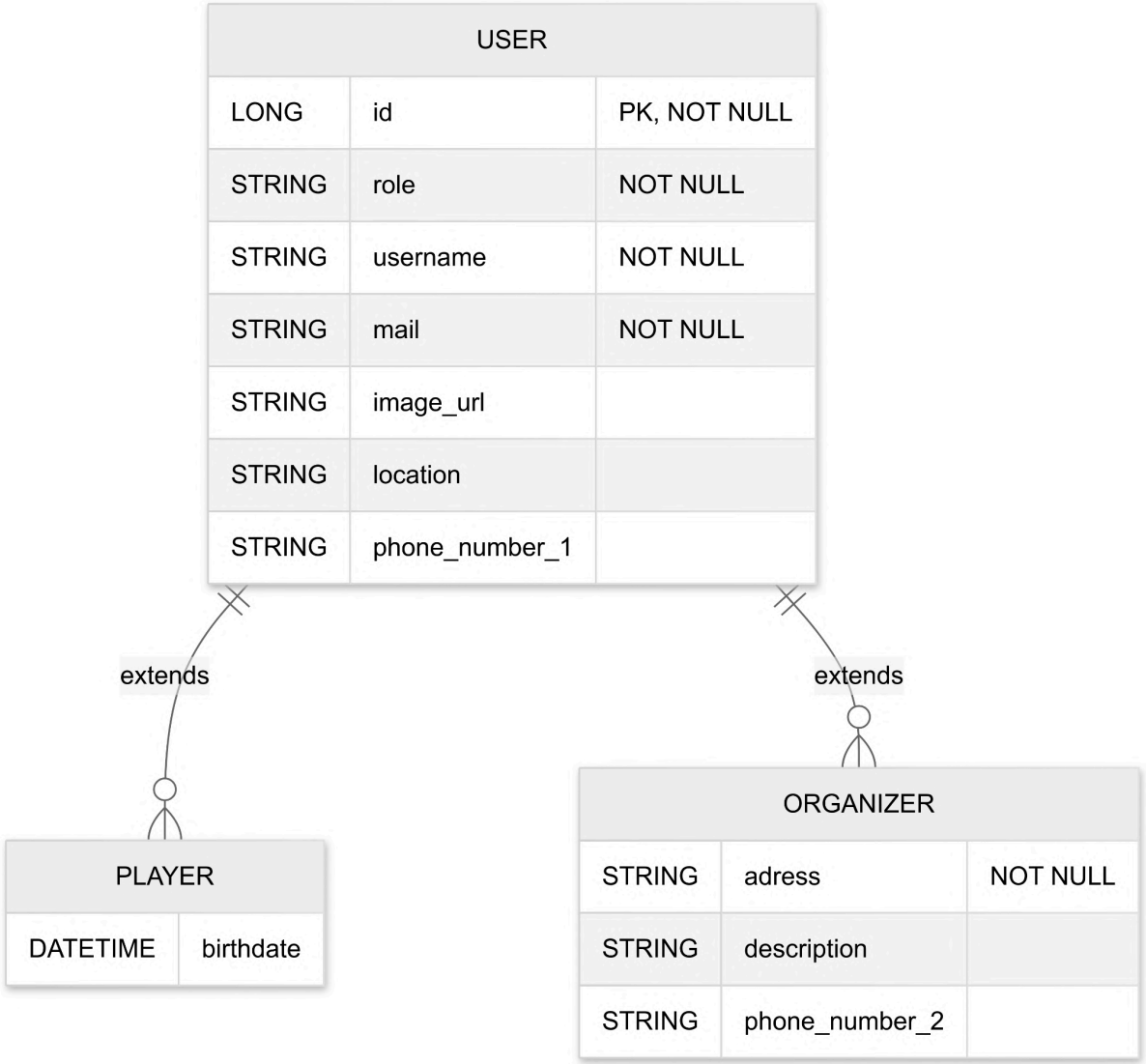


Imagen ampliada: UserManagementService db.png

Figura 9. Diagrama ER de la base de datos de *UserManagementService*.

2.5.2.4.- Estructura general del microservicio *UserManagementService*

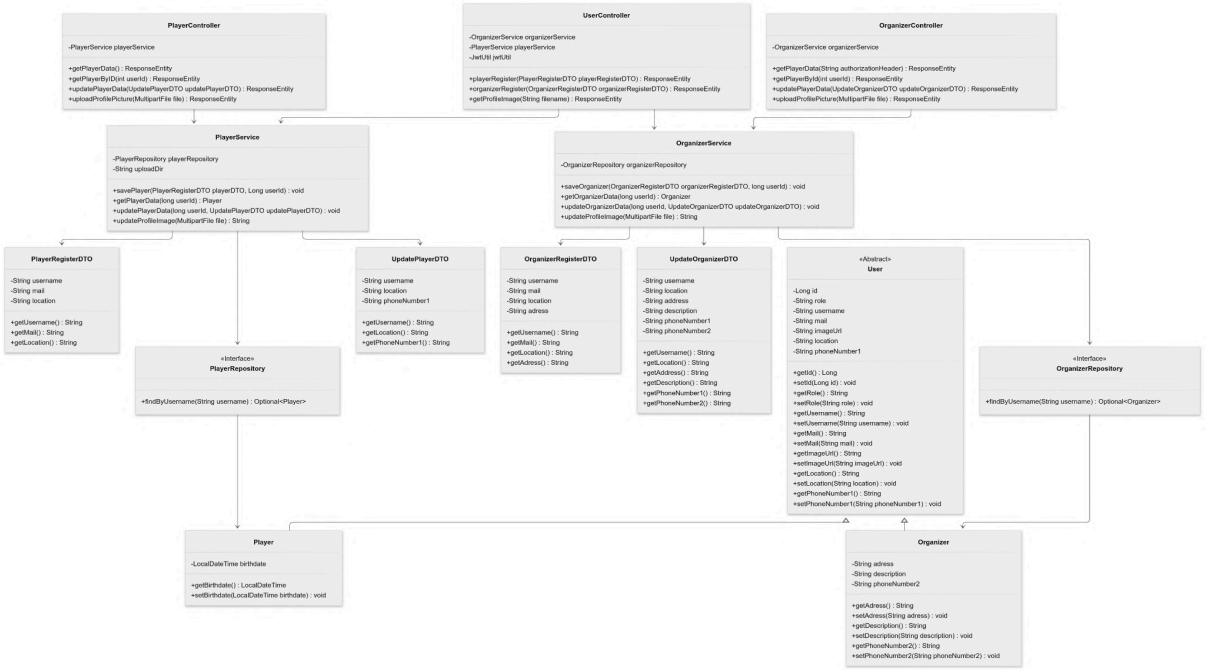


Imagen ampliada: Estructura general de UserManagementService.png

Figura 10. Estructura general del microservicio *UserManagementService*.

2.5.3.- Torneos

Permite la creación y gestión de torneos por parte del organizador, y que los jugadores se inscriban a estos torneos. También cuenta con un buscador de torneos aplicando ciertos filtros. Los registros de los torneos acabados son accesibles mediante un historial. Este microservicio permite que tanto jugadores como organizadores puedan añadir eventos a su calendario.

2.5.3.1.- Estado de los torneos

Estos torneos cuentan con 3 estados y cada torneo solo puede encontrarse en un estado diferente. Los estados son: creado, activo y cerrado.

- El estado “creado” es cuando el organizador crea el torneo y este queda a la espera de que llegue la fecha indicada para ser iniciado. Durante este tiempo los jugadores podrán inscribirse a ese torneo y los organizadores podrán editar algunos datos, como la fecha y la hora de inicio. Aclarar que el organizador pone una cantidad máxima de jugadores que pueden inscribirse.

- El estado “activo” es cuando ha llegado la fecha de inicio del torneo y el organizador ha confirmado el iniciar el torneo. En este estado es cuando el torneo se “activa” y se puede acceder al bracket (árbol del torneo) y ver los cambios en tiempo real.

- El estado “cerrado” se da en el instante en el que se elige el ganador del torneo. Automáticamente por detrás, el torneo se marca como cerrado y pasa a la lista del historial tanto del organizador como de los jugadores que participaron en él.

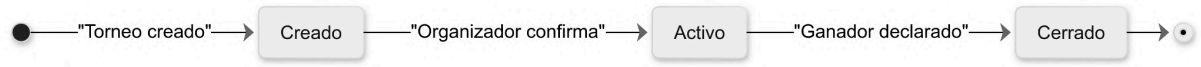


Imagen ampliada: Diagrama de estado de un torneo.png
Figura 11. Diagrama de estados de torneos.Diagrama de estados de los torneos.png

2.5.3.2.- Base de datos

En la base de datos de este microservicio nos encontramos con dos tablas relacionadas, *Tournament* y *PlayerRegistration.* Aparte de estas dos existe la tabla *Events,* que es la que almacena los eventos de jugadores y organizadores**.** La idea detrás de esta estructura es que por cada torneo, exista una lista de inscripciones que la llenan los jugadores que se van inscribiendo respetando la capacidad máxima que marca el organizador.

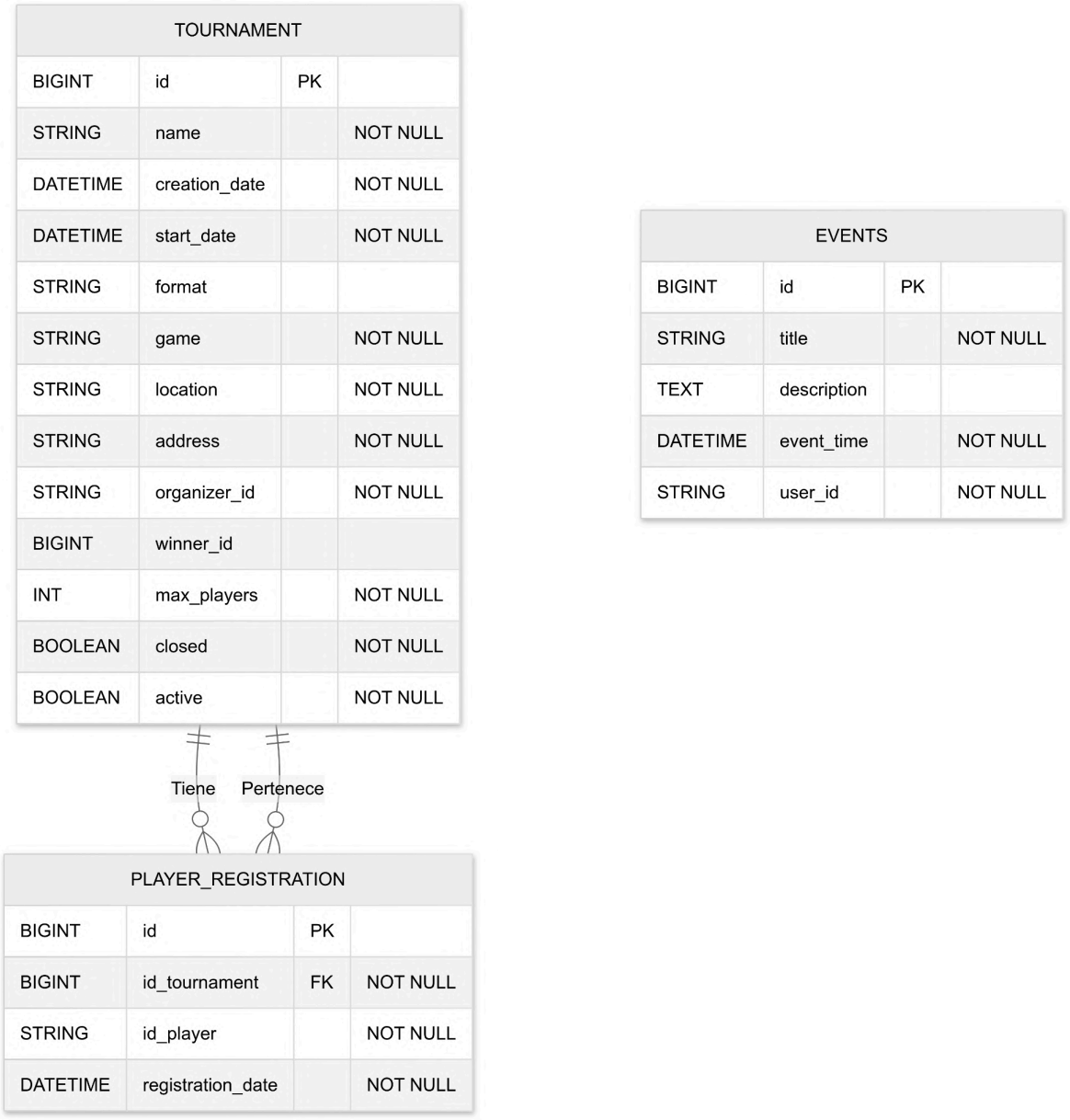


Imagen ampliada: Diagrama ER TournamentService.png
Figura 12. Diagrama ER ***TournamentService*.**

2.5.3.3.- Estructura general del microserivcio *TournamentService*.

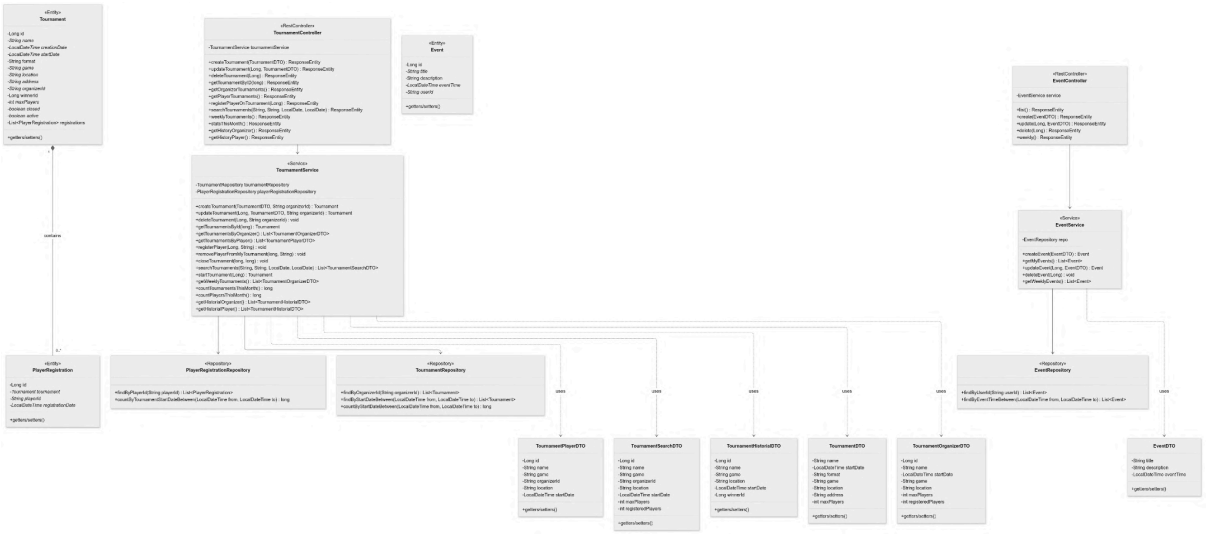


Imagen ampliada: Diagrama de clases TournamentService.png

Figura 13. Diagrama de clases de la estructura del microservicio *TournamentService*.

2.5.4.- Emparejamiento

Al iniciarse el torneo, y pasar a estado “activo”, este microservicio recibe la lista de jugadores asociada al torneo que se va a iniciar y la deja desordenada, haciendo así que los emparejamientos sean aleatorios. Seguidamente construye todo el bracket del torneo en un *JSON* donde se guarda información de que jugadores han ganado cada partido y han pasado de ronda y quienes han quedado descalificados. Este documento que está guardado en una base de datos MongoDB, recibe cambios según el organizador vaya especificando qué jugador ha ganado cada partido. Todo esto se ve reflejado en el cliente en tiempo real con ayuda de ***WebSockets***.

2.5.4.1.- ¿Que son los *WebSockets?*

WebSockets es un protocolo de comunicación que permite una conexión persistente y bidireccional entre el cliente y el servidor. A diferencia del *HTTP*, donde el cliente hace una petición y espera una respuesta, con *WebSockets* ambos pueden enviarse datos entre sí en cualquier momento, sin necesidad de volver a abrir la conexión.

2.5.4.2..- ¿Por qué usar WebSockets?

A la hora de planificar el proyecto se tenía claro que el estado del torneo pudiese ser modificado en tiempo real. Se quería que el jugador tuviese respuesta inmediata y supiera en cada momento que está pasando en tiempo real. Se investigó qué protocolo podría ayudar a cumplir con este objetivo, ya que a pesar de existir otras alternativas unidireccionales, como lo es *Server-Sent Events (SSE),* se ha optado por este protocolo ya que a futuro si se requerirá de esa bidireccionalidad en la que el jugador le mande una confirmación al organizador en tiempo real. En resumen *websockets* cumple con los mínimos necesarios además de que sienta la base de futuras implementaciones.

2.5.4.3..- Implementación de WebSockets en este microservicio

En el instante en el que el torneo pasa a estar “activo”, después de arreglar el emparejamiento, se crea la estructura del torneo y se almacena en una base de datos *MongoDB*. A continuación se abre un canal de comunicación con *WebSocket* en el que se envían los datos de la estructura del torneo . Cuando se accede a la pantalla del bracket, automáticamente se conecta a ese canal de comunicación y se reciben los datos actualizados. Cada vez que el organizador seleccione el ganador de cada partido, el microservicio actualiza los datos con el ganador, donde lo almacena en la base de datos y emite por el canal los datos actualizados*,* haciendo así que todos los que estén conectados a este canal de comunicación reciban en tiempo real la estructura del torneo modificada.

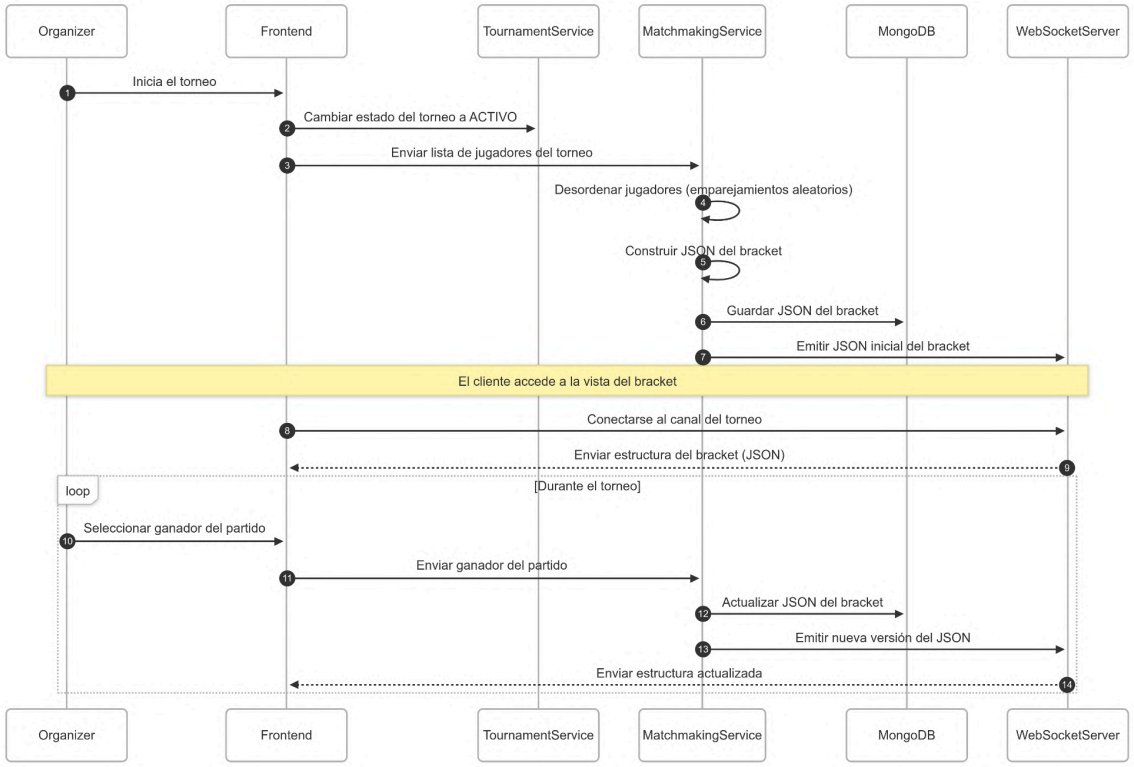


Imagen ampliada: Diagrama de secuencias MatchmakingService.png
Figura 14. Diagrama de secuencias que refleja el funcionamiento del torneo desde el momento que se inicia.

2.5.4.4.- Base de datos

La base de datos es muy diferente respecto a los anteriores microservicios, Aquí se trabaja con la base de datos *MongoDB***,** ya que se requiere de una estructura muy concreta que sería muy difícil construir en una base de datos relacional por tablas como lo es *MySQL*.

2.5.4.4.1.- ¿Por qué es más sencillo con MongoDB?

Es muy importante respetar cierta estructura ya que la librería encargada de mostrar el bracket en el *frontend* la requiere. *MongoDB* es una base de datos *NoSQL* orientada a documentos, por lo que es más sencillo y eficiente almacenar un *JSON* con una forma específica. En *MySQL* se traduciría en múltiples tablas relacionadas entre sí y con cierta estructura que harían las consultas más complejas.

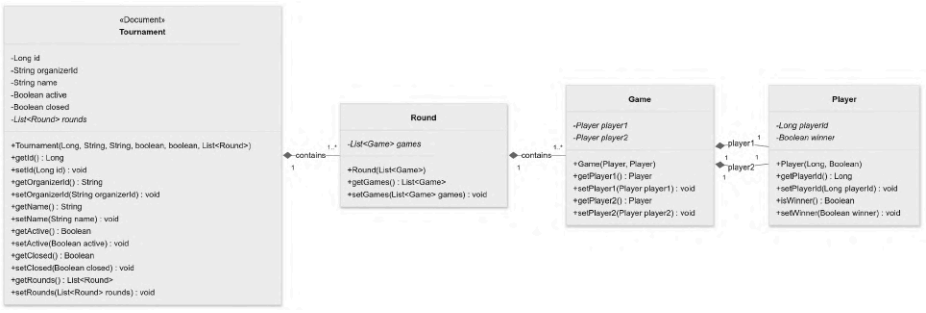


Imagen ampliada: Diagrama de clases Tournament en MatchmakingService.png
Figura 15. Diagrama de clases de la estructura de torneo en ***MatchmakingService*.**

2.5.4.5.- Estructura general del microserivcio *MatchmakingService*.

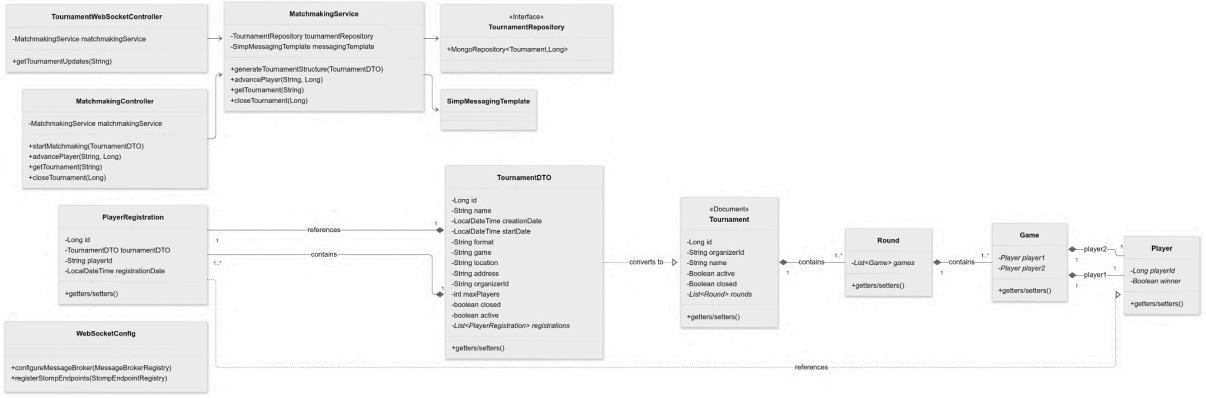


Imagen ampliada: Diagrama de clases del microservicio MatchmakingService.png

Figura 16. Diagrama de clases de la estructura general del microservicio ***MatchmakingService***.

2.6.- Funcionalidades

2.6.1.- Registro de jugador

Cualquier persona que quiera participar en los torneos publicados puede registrarse llenando un formulario en la página web del proyecto.

2.6.2.- Registro de organizador

Los organizadores, que por lo general son tiendas donde se organizan estos torneos, deben solicitar a un administrador una cuenta “organizador”. Esto se debe a que pensamos que no cualquiera puede acceder a una cuenta de este tipo, ya que todo el recorrido de torneos debería ser exclusivo de lugares especializados. El administrador debe verificar que la persona que se ponga en contacto sea una entidad de este tipo para poder ofrecer acceso a una cuenta “organizador”.

2.6.3.- Inicio de Sesión

Los usuarios, sin importar si su cuenta es jugador u organizador, utilizan un mismo login, donde el servidor mediante la información de la base de datos, sabe responder qué rol tiene el usuario que está intentando iniciar sesión.

2.6.4.- Perfil de usuario

Los usuarios pueden acceder a la información de su perfil sin importar el rol que tengan. Aclarar que el perfil de un jugador y el de un organizador no muestran la misma información, aunque tienen algunas en común. Los usuarios pueden modificar algunos datos, como la localización o el nombre de usuario.

2.6.5.- Torneos (Organizadores)

- Crear torneos con diferentes configuraciones
- Posibilidad de eliminar un jugador de su torneo si este ve que está cometiendo alguna infracción o el jugador no puede asistir al evento.
- Dar inicio al torneo cuando llega la fecha.
- Interacción con el bracket para decirle al programa que jugador pasa de ronda.

2.6.6.- Torneos (Jugadores)

- Acceso a inscribirse en torneos creados por un organizador.
- Ver en tiempo real cómo va progresando el torneo.

2.6.7.- Calendario

Tanto organizadores como jugadores tendrán a su disposición un calendario en el que se guardará de forma automática los torneos creados (organizador) o torneos a los que están inscritos (jugador). De esta manera los usuarios pueden tener una visión más amplia de los torneos que tienen pendientes.

Además de esto, tanto organizadores como jugadores, cuentan con un apartado de eventos. Para los organizadores es útil ya que aparte de torneos, también cuentan con eventos con otras finalidades. Es por eso que ambos perfiles pueden poner estos eventos en su calendario.

2.6.8.- Historial

Tanto organizadores como jugadores tendrán acceso a un historial de torneos en los que participaron anteriormente. Ahí es donde quedan registrados los resultados de los eventos en los que participaron.

2.7.- Diseño de la interfaz web

El diseño de Deckly parte de la paleta cromática inspirada en *CardMarket*, adaptando sus tonos azulados para transmitir seriedad y profesionalidad propias de un centro de gestión de torneos TCG.

2.7.1.- Inspiración

La inspiración en *CardMarket* nace en querer adoptar su mismo estilo de negocio, que es ser el refugio principal de todos los juegos *TCG,* pero en nuestro caso no como página de compra venta de cartas y artículos de juego, sino como gestor de torneos *TCG.*

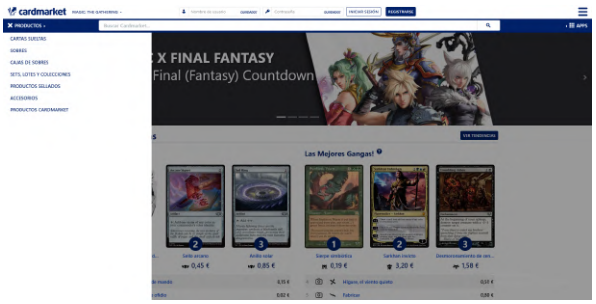


Figura 17.
Página principal de *CardMarket*, *light mode*

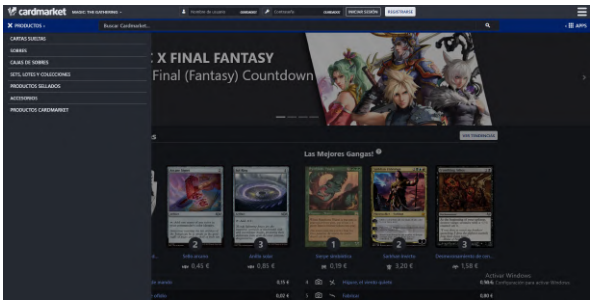


Figura 18.
Pàgina principal de *CardMarket*, *dark mode*

En el diseño de *Deckly* se optó por mantener los fondos claros para tener una mayor visibilidad, y cambiar los tonos negros por diferentes tonalidades de azul. La idea no era copiar a *CardMarket*, sino hacer que se asemejen, para que el jugador sienta que ya ha visitado la página de alguna u otra forma y se sienta cómodo, ya que la inmensa mayoría de jugadores, conocen esta página. Es por eso que hay ciertas características que han sido retocadas para dar otro enfoque diferente.

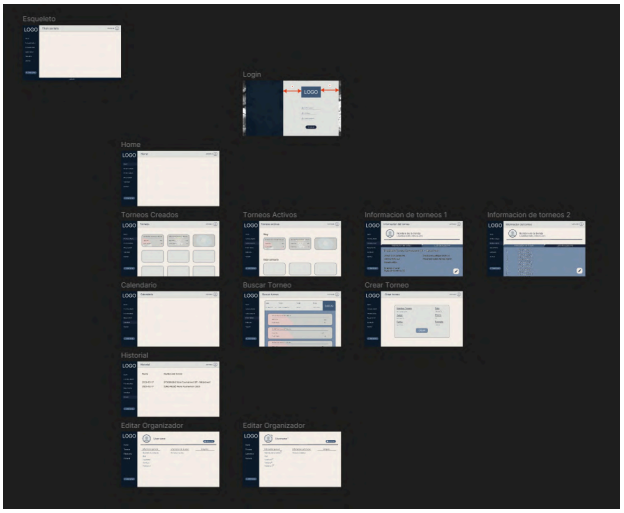
Estas características son:

- Un sidebar mas pequeño y fijo para dar sensación de control, como en un *dashboard* o un panel de control.
- El perfil del usuario en la esquina superior derecha, para asemejarse a paginas del estilo red social.
- Predominancia del color azul como símbolo de seguridad y seriedad.
- Bordes de tarjetas e interfaces redondeadas para hacer la contraparte en el aspecto de la seriedad, dando una sensación amigable. Recalcar que esta característica no anula la anterior mencionada, sino que ambas se refuerzan mutuamente.

2.7.2.- Evolución del diseño

Gracias a la planificación del proyecto se pudo desempeñar todas estas características de la forma en la que se pensaron, ya que al principio el diseño de Deckly era muy diferente al actual.

Figura 19. Mockup inicial del diseño. Ver en más detalle en anexos.



Figuras 20 y 21.
Prototipo inicial del diseño (vista organizador y jugador). Ver en más detalle en anexos.

En las imágenes se puede ver que el diseño de la web al principio tenía unos colores más anaranjados y no azules como acabaron siendo. Esto se debe a que no únicamente se utilizó *CardMarket* como fuente de inspiración para el diseño del proyecto. En este caso, la paleta de colores inicial fue tanteada por la página de *RK9*, que como se menciona más arriba (Estrategia y planificación del proyecto), es una página de torneos destinada únicamente a la *IP* de *Pokémon.* Los colores anaranjados transmiten otro tipo de sensaciones a las de seriedad y seguridad, como son entusiasmo y energía. Este enfoque era adecuado para Deckly ya que este tipo de sensaciones y emociones son las que se quiere transmitir en un torneo, pero decidimos cambiarlo por adoptar un tono más profesional, adaptándolo al servicio que ofrece Deckly. Es por eso que al final, la paleta de colores elegida fue la de tonos azules y claros.

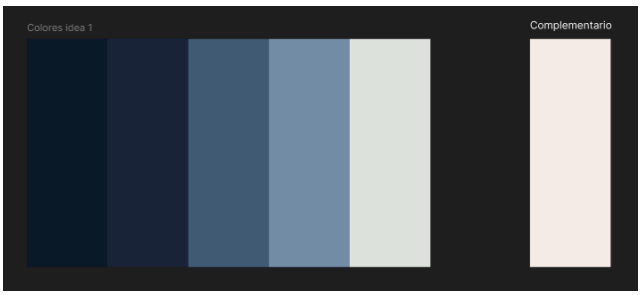
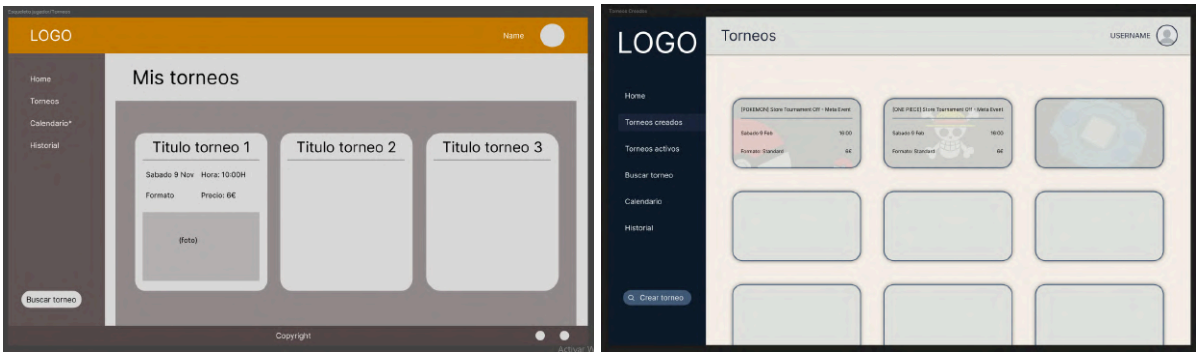


Figura 22. Paleta de colores final.



Figuras 23 y 24.
Comparativa de diseño y color entre el *mockup* y el prototipo.

2.7.3.- Diseño en profundidad

El fondo de la barra lateral emplea el color más oscuro para enmarcar la navegación, mientras que un azul intermedio se reserva a botones y enlaces secundarios. Un tono más suave se utiliza en los bordes de las tarjetas y contenedores, aportando estructura sin restar protagonismo al contenido, y los acentos ligeramente más claros marcan los estados de “hover” en elementos interactivos. El fondo general y las tarjetas se mantienen en un tono neutro muy claro, garantizando contraste y buena legibilidad.

Para facilitar la navegación y la comodidad del usuario, se ha optado por un sidebar fijo en la parte izquierda que agrupa las secciones principales de la aplicación. Esta disposición emula la mayoría de paneles de administración actuales y permite acceder en un solo vistazo a todas las funcionalidades. En la esquina superior derecha se sitúa el acceso al perfil de usuario, donde aparecen avatar y menú desplegable con opciones como “Mi perfil” y “Cerrar sesión”. De este modo, se aprovecha un patrón de uso ya familiar para quien visita aplicaciones web de gestión.

La interfaz está concebida como un gran panel de control. Cada página muestra tarjetas con la información esencial y botones claramente identificados según el nivel de prioridad. Además, se han incluido microinteracciones suaves en transiciones y efectos de hover para reforzar la respuesta visual sin distraer, generando una sensación de dinamismo y solidez.

En su conjunto, el diseño de Deckly persigue un equilibrio entre profesionalidad y sencillez. La elección de los colores, el uso de un sidebar persistente y la colocación del perfil en la zona superior siguen convenciones probadas en aplicaciones de panel, lo que reduce la curva de aprendizaje y hace que tanto organizadores como jugadores encuentren rápidamente sus secciones de interés. De esta forma, la estética y la experiencia de usuario quedan alineadas con los objetivos de eficiencia y centralización que definen el proyecto.

Con la toma de decisiones y la justificación de los cambios tomados, después de haber visto cómo eran los *mockups* y el prototipo del diseño, la web ha adoptado una identidad propia, que es lo que se quería lograr en un principio.

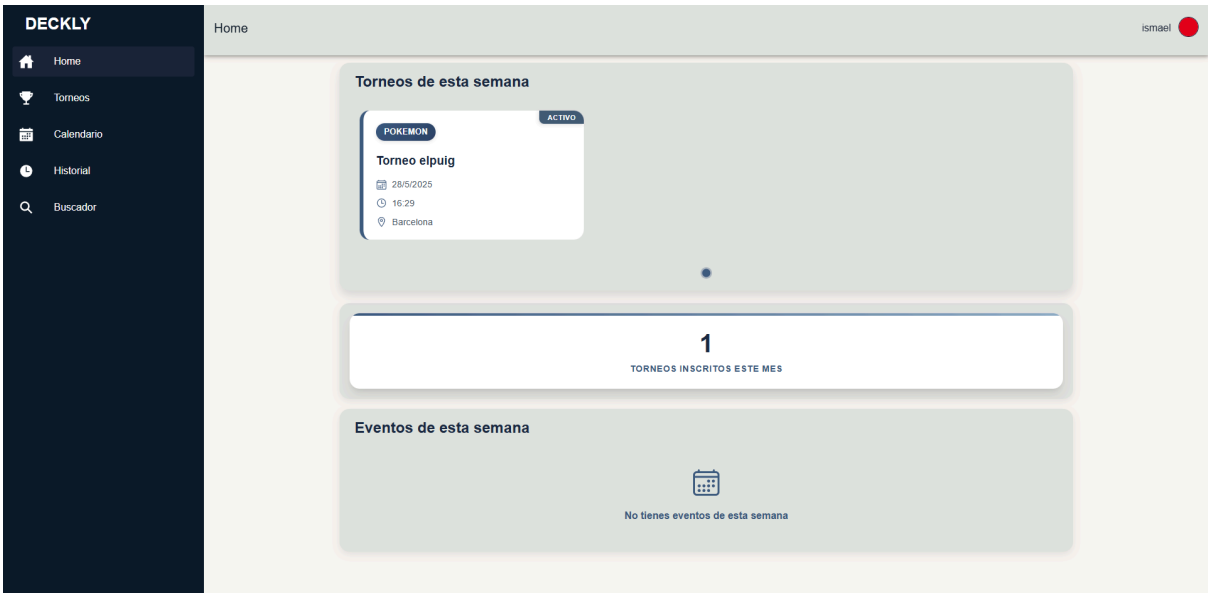


Figura 25. Diseño final de la web (Home de jugador)

Figura 26. Diseño final de la web (*Landing page*)

3.- Conclusiones

3.1.- Conclusiones generales del proyecto

La idea de Deckly nace un año atrás de su desarrollo, cuando los miembros del equipo que hoy forman parte del proyecto, expresaron ideas para aplicaciones que tenían en mente. Es gracias a esto que en una primera instancia se tuvo más tiempo para pensar en cómo se quería desarrollar la idea y de qué manera. El equipo mantuvo un largo periodo de tiempo en el que únicamente se hablaba de la planificación, los objetivos a cumplir y de posibles nuevas aportaciones al gigante que se estaba construyendo.

En un inicio, la idea era mucho más grande, puesto que no se pretende desarrollar como proyecto final de curso. Es por eso que se tuvo que hacer recortes para cumplir con el plazo de entrega, ya que muchas de estas tecnologías eran desconocidas por el equipo, y hubo que dedicar tiempo para aprender sobre ellas y lograr que todo tuviese cohesión en tanto a conocimientos, porque otras tantas ya eran conocidas de antemano y no se quería que la disparidad de conocimientos sobresaliera de forma muy brusca.

Fue una gran idea hacer este proyecto, porque gracias a ello se han estudiado tecnologías muy útiles e interesantes las cuales están a la orden del día en tanto a conocimientos demandados por el sector.

Es cierto que al principio surgieron dudas sobre las elecciones escogidas, como los microservicios, ya que era una idea muy distante de la cual el equipo no tenía mucha información, y conforme se avanzaba en el proyecto, se planteó la idea de pasarlo a monolítico. Esto no acabó siendo así gracias a la ambición de los integrantes del proyecto por aprender cómo realmente funciona este tipo de estructura de diseño. Al final se logró ver todo el potencial que aportaba y se apostó por seguir adelante con ello.

Pese a todas las adversidades que han ido apareciendo por el camino, el resultado final es digno de la alegría del equipo, ya que se llegó a cumplir todos y cada uno de los objetivos que se han mencionado.

3.2.- Consecución de los objetivos

Como se mencionó anteriormente, el proyecto contaba con muchas ideas, pero no eran realistas. Es por eso que se tuvo que hacer una separación entre *core* o necesario, y opcional o deseado. Gracias a esto, el equipo contaba con metas realistas las cuales fueron alcanzadas de manera idónea, ya que a cada una de estas metas u objetivos, se le dedicó su tiempo y esfuerzo en ser estudiada y aprendida.

Con todo esto dicho, finalmente sí que surgieron propuestas que se acabaron añadiendo al proyecto; así que no solamente el propósito se mantuvo en la línea de los objetivos, si no que también a medida que surgían nuevas ideas las cuales podrían implementarse, sin interferir gravemente en el desarrollo, se añadieron.

3.3.- Valoración de la metodología y planificación

Gracias a que se pudo invertir gran parte del tiempo del proyecto en la investigación y planificación del mismo, se tenían las metas más presentes a la hora de construir el proyecto.

Pero si que es cierto que el equipo se encontró con un problema, que finalmente fue una virtud, que fue cuando entró en la fase de desarrollo. La metodología que se adoptó en un inicio en el proyecto fue la de *waterfall*, la cual permitió ver todos los meses desde otro prisma y se pudo organizar de manera en que todo tuviese su tiempo de preparación y maduración. No fue hasta que el periodo de desarrollo apareció, que el equipo se sintió perdido por cuestiones de volumen de trabajo y diferentes formas de abarcar al proyecto. Es por eso que se optó por integrar una metodología en mitad de todo el proceso, que fue la de *scrum*, con la que los integrantes del equipo ya estaban entrenados. Al principio sirvió más bien de bloc de notas con tareas apuntadas, pero a medida que se iba trabajando, se iba adoptando más la metodología de trabajo de *scrum,* y se empezó a destinar tiempo a pequeñas sesiones para poner al equipo al dia, se empezó a trabajar por *sprints* y todo fluyó de una mejor manera.

La adopción de esta metodología brindó al proyecto de más profundidad y de más capas de esfuerzo y trabajo, las cuales se han podido ver reflejadas en el proyecto. En este aspecto, la valoración que se aporta a la metodología de trabajo y planificación empleada, es excelente.

3.4.- Visión de futuro

Deckly nació de una idea que se tuvo no porque fuese una buena idea de proyecto, si no como una solución a dificultades que enfrenta el sector. Es por ello que en el futuro, Deckly se seguirá desarrollando hasta llegar al punto de poder sacar la herramienta al mercado para hacer frente a estas adversidades.

Una de las primeras cuestiones a tratar, será la de replantear ideas que han surgido durante el proyecto, y rescatar algunas que ya estaban concebidas de antes, para saber cómo seguir avanzando a partir del punto actual y saber qué nuevos objetivos se quieren cumplir y llevar a cabo.

4.- Glosario

Agile: Enfoque de desarrollo de software que promueve la colaboración, la adaptabilidad al cambio y entregas incrementales.

API: Interfaz de Programación de Aplicaciones que permite la comunicación y el intercambio de datos entre sistemas de forma estandarizada.

API Rest/Restful: Arquitectura de diseño de APIs basada en *HTTP* y recursos identificados por URLs; utiliza métodos como *GET, POST, PUT* y *DELETE*.

AuthService: Servicio responsable de gestionar la autenticación de usuarios.

Authorization: Proceso que determina qué recursos o acciones puede realizar un usuario tras haber sido autenticado.

Backend: Parte de la aplicación que gestiona la lógica, la base de datos y la comunicación en el servidor, inaccesible directamente para el usuario.

BCrypt: Algoritmo de hashing adaptativo diseñado para proteger contraseñas frente a ataques de fuerza bruta.

BCryptPasswordEncoder: Implementación de *Spring Security* que utiliza *BCrypt* para codificar y verificar contraseñas.

Bracket del torneo: Diagrama que muestra los enfrentamientos entre participantes y el avance de cada uno según sus resultados.

Diagrama ER: Representación gráfica de entidades y sus relaciones en una base de datos (Entidades–Relaciones).

Diagrama de estados: Modelo *UML* que describe los distintos estados de un objeto y las transiciones entre ellos.

Diagrama de secuencia: Diagrama *UML* que muestra la interacción entre objetos en el tiempo mediante mensajes ordenados.

Endpoint: Punto final de comunicación en una *API*, definido por una *URL* y un método *HTTP* para acceder a un recurso.

Frontend: Parte de la aplicación o sitio web con la que interactúan directamente los usuarios, incluye *UI, HTML, CSS* y *JavaScript.*

Framework: Conjunto de bibliotecas y estructuras predefinidas que facilitan el desarrollo de software.

Ionic: Framework para crear aplicaciones móviles híbridas usando tecnologías web (*HTML*, *CSS* y *JavaScript*) y *Cordova/Capacitor*.

Java: Lenguaje de programación orientado a objetos muy utilizado en aplicaciones empresariales, *Android* y sistemas distribuidos.

JSON (JavaScript Object Notation): Formato ligero y fácil de leer para intercambio de datos, basado en pares clave–valor y estructuras de listas.

JWT (JSON Web Token): Token compacto en formato *JSON* para transmitir información verificada entre partes, común en autenticación.

Kanban: Sistema visual de gestión de tareas con tableros y tarjetas que ayuda a controlar y optimizar el flujo de trabajo.

MatchmakingService: Servicio encargado de emparejar jugadores o participantes.

Microservicios: Arquitectura que divide una aplicación en servicios pequeños, independientes y desplegables por separado.

NoSQL: Categoría de bases de datos no relacionales que almacenan datos en formatos flexibles como documentos, grafos o pares clave–valor.

PasswordEncoder: Interfaz de *Spring Security* para codificar contraseñas antes de almacenarlas y verificarlas luego.

Params: Parámetros que se envían en una petición *HTTP*, pueden ir en la *URL* (*query*), encabezados o cuerpo (*body*).

protocolo HTTP: Conjunto de reglas para la transferencia de información en la web, basado en solicitudes y respuestas cliente–servidor.

Resource: Representación de una entidad o dato accesible a través de una *API*, generalmente mapeada a una URL específica.

Scrum: Marco de trabajo ágil que organiza el proyecto en iteraciones cortas llamadas sprints, con roles y ceremonias definidas.

SQL: Lenguaje de consulta estructurado para gestionar y manipular bases de datos relacionales.

Spring Security: Módulo de *Spring* que proporciona autenticación, autorización y protección contra amenazas en aplicaciones Java.

TCG (Trading Card Game): Juego de cartas coleccionables en el que los jugadores construyen mazos personalizados y compiten entre sí.

Token: Cadena de caracteres que representa un permiso o sesión de acceso, usada para autenticar y autorizar peticiones.

TournamentService: Servicio que gestiona la lógica de creación de torneos o competiciones.

UI/UX: "*User Interface*" (diseño de la interfaz) y "*User Experience*" (experiencia percibida por el usuario al interactuar).

UserController: Componente en patrón *MVC* que recibe y gestiona las peticiones *HTTP* relacionadas con usuarios.

UserManagementService: Servicio encargado sobre datos de usuarios.

5.- Bibliografía

Learn Hibernate and Spring (As A Total Beginner) Tutorial. (2025). Udemy. https://www.udemy.com/course/spring-hibernate-tutorial/?couponCode=ST15MT20425G2

spring-boot. (2024, July 26). FreeCodeCamp.org. https://www.freecodecamp.org/news/tag/spring-boot/

freeCodeCamp.org. (2024, February 6). Spring Boot & Spring Data JPA – Complete Course. YouTube. https://www.youtube.com/watch?v=5rNk7m_zlAg

Ivana Soledad Rojas Córsico. (2023, July 7). ¿Cómo crear el login? Spring Boot 3 + Spring Security 6 + JWT Authentication #backend. YouTube. https://www.youtube.com/watch?v=nwqQYCM4YT8

Spring Boot :: Spring Boot. (n.d.). Docs.spring.io. https://docs.spring.io/spring-boot/index.html

kamilwylegala. (2024, February 29). GitHub - kamilwylegala/vue-tournament-bracket: Vue component for rendering single elimination tournament brackets. GitHub. https://github.com/kamilwylegala/vue-tournament-bracket

OpenAI. (2025). ChatGPT. ChatGPT; OpenAI. https://chatgpt.com/

Spring Boot 2025. (2025). Spring Boot 2025. https://mp3uf6-spring.super.site/

Mermaid | Diagramming and charting tool. (n.d.). Mermaid.js.org. https://mermaid.js.org/

6.- Anexos

Proyecto:
https://github.com/IsmaMH779/proyecto_final

Mockup inicial del diseño:
https://www.figma.com/design/U3yS6io79CLZ4He8vpTzxs/Web-concept?node-id=0-1&t=ys0HiM5PfsOouXph-1

Paleta de colores:
https://www.figma.com/design/MHyp6YQZ6vE4qWz8FukfR2/Dise%C3%B1o-Deckly?node-id=0-1&p=f&t=rrAWG2dmc91TCVS0-0

Vista jugador (Prototipo inicial del diseño):
https://www.figma.com/design/MHyp6YQZ6vE4qWz8FukfR2/Dise%C3%B1o-Deckly?node-id=1-2&p=f&t=rrAWG2dmc91TCVS0-0

Vista organizador (Prototipo inicial del diseño):
https://www.figma.com/design/MHyp6YQZ6vE4qWz8FukfR2/Dise%C3%B1o-Deckly?node-id=1-3&p=f&t=rrAWG2dmc91TCVS0-0