



Trabajo De Fin de Grado

**Desarrollo Aplicación Android de Consulta de Datos
de A1 Padel haciendo uso de Spring Framework,
Web Scraping, Docker, Azure, Kubernetes y Android
Studio.**

Alberto Navarro Vega

Grado en Desarrollo de Aplicaciones Multiplataforma
Mayo, 2024

ÍNDICE

1. Descripción del proyecto
2. Plan de trabajo
3. Explicación del Back-End
 - 3.1 Arquitectura.
 - 3.1.1 Microservicios
 - 3.1.2 Bases de datos
 - 3.1.3 Spring Security
 - 3.1.4 Testing
 - 3.1.5 GitFlow
 - 3.2 Automatización.
 - 3.2.1 Dockerizacion
 - 3.2.2 GitHub Actions
 - 3.2.3 Azure Functions
 - 3.3 Despliegue en Azure.
 - 3.3.1 Recursos
 - 3.3.2 Kubernetes
4. Explicación del Front-End
5. Guia de Uso
 - 5.1 Desde la propia Aplicación
 - 5.2 Desde Postman
6. Diagramas
7. Historia del proyecto
8. Justificación
9. Objetivos
10. Motivación
11. Conclusión

1. Descripción del proyecto.

Este proyecto trata de una aplicación Android en la cual puedes consultar todos los torneos y resultados de partidos que se han disputado en toda la historia de la liga de A1 Pádel, además de un ranking (Hasta el Top50) tanto masculino como femenino. Ambas funcionalidades constan de datos actualizados semanalmente gracias a técnicas de Web Scraping Automatizadas con Scripts en Python y usando Azure Functions.

Entrando en una breve explicación más técnica:

- BackEnd está realizado usando Spring Framework (Boot, Data, Security, Web, Cloud...) uniendo estos servicios con una API Gateway y apoyándome en Eureka Server para el balanceo de carga.
- El CI/CD usando GitHub Actions, Docker y AKS de Microsoft Azure.
- El despliegue se realiza en Microsoft Azure.
- FrontEnd se ha desarrollado con AndroidStudio con Java.

2. Plan de trabajo

Para un plan de trabajo organizado, he estructurado la siguiente división de tareas para un enfoque más claro y conciso. Este plan de trabajo tiene una estimación de 12 semanas aproximadamente, es decir desde el 18 de marzo al 12 de junio.

1. Estudio de las tecnologías a usar
2. Web Scraping.
 - 2.1. Documentación y aprendizaje.
 - 2.2. Obtención de todos los datos de un torneo y el ranking.
 - 2.3. Automatización para todos los partidos y ambos ranking.
 - 2.4. Insertar los datos directamente en MySQL.
 - 2.5. Automatizar el proceso con Azure Functions.
3. Base de Datos.
 - 3.1. Diseño de la Base de Datos.
 - 3.2. Creación de la base de datos.
 - 3.3. Introducción de los datos obtenidos en la Tarea 2.
4. Desarrollo de las APIs con Spring Boot.
 - 4.1. Documentación.
 - 4.2. Creación del proyecto Spring Boot.
 - 4.3. Diseño e implementación de los recursos.
 - 4.4 API Gateway + Eureka server
5. Desarrollo de la Aplicación Android.
 - 5.1. Diseño de la estructura de la Aplicación Android.
 - 5.2. Mostrar los datos de la manera más visual posible.
6. Realización de pruebas de la aplicación desarrollada y despliegue.
7. Escritura de memoria y preparación de la defensa

Diagrama de Gantt Previo

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S12
Tarea 1											
Tarea 2											
2.1											
2.2											
2.3											
2.4											
2.5											
Tarea 3											
3.1											
3.2											
3.3											
Tarea 4											
4.1											
4.2											
4.3											
4.4											
Tarea 5											
5.1											
5.2											
Tarea 6											
Tarea 7											

Diagrama de Gantt Final

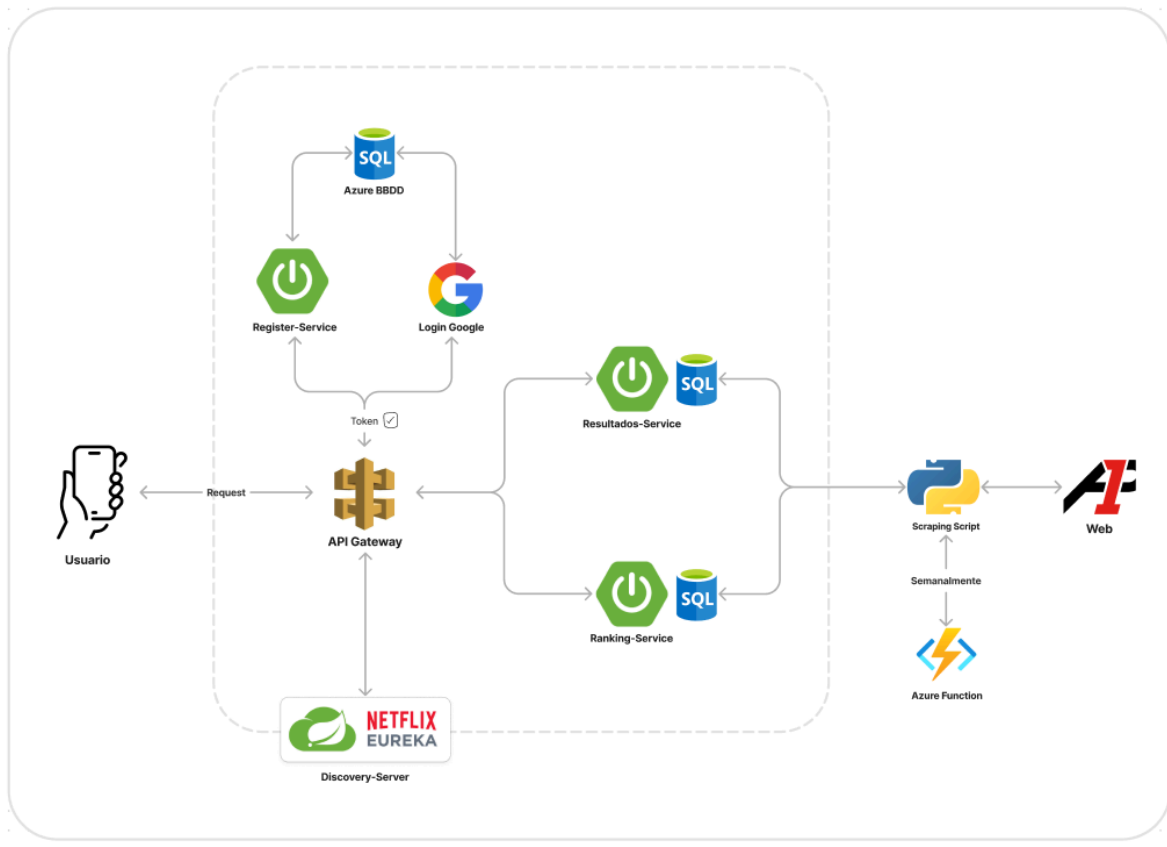
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
Tarea 1											
Tarea 2											
2.1											
2.2											
2.3											
2.4											
2.5											
Tarea 3											
3.1											
3.2											
3.3											
Tarea 4											
4.1											
4.2											
4.3											
4.4											
Tarea 5											
5.1											
5.2											
Tarea 6											
Tarea 7											

Tras la finalización de este proyecto, se observa como finalmente he dedicado más tiempo al diseño y testeo de la aplicación (Tareas 5 y 6) pero por otro lado he dedicado menos tiempo en el desarrollo de las APIs con Spring Boot (Tarea 4) y escritura de la memoria (Tarea 7).

3. Explicación del Back-End.

Para la explicación del BackEnd, haré una división de tres puntos fundamentales en el desarrollo del proyecto: **Arquitectura, Automatización y Despliegue.**

3.1 Arquitectura



3.1.1 Microservicios

API Gateway

El artefacto denominado api-gateway se encarga del enrutamiento correcto de los microservicios. Para su funcionamiento, este proyecto depende de varias bibliotecas, incluidas Spring Cloud Starter Gateway, Spring Boot DevTools, Spring Boot Configuration Processor, Project Lombok, Spring Boot Starter Test, Spring Cloud Starter Netflix Eureka Client, jjwt-api, jjwt-impl, jjwt-jackson y Spring Boot Starter Webflux.

En el archivo AppConfig, se define un bean de RestTemplate. Este bean se utiliza para realizar solicitudes HTTP desde el API Gateway a los microservicios que están detrás de él.

El AuthenticationFilter es un filtro de autenticación que se encarga de verificar las solicitudes entrantes. Este filtro extiende la clase AbstractGatewayFilterFactory, una clase base para la creación de filtros personalizados en Spring Cloud Gateway. El método apply() implementa la lógica de filtrado, verificando si la solicitud está

asegurada y contiene un token de autorización en la cabecera. Si la solicitud está asegurada y no contiene el token, se lanza una excepción. Luego, el filtro valida el token JWT utilizando JwtUtil. Si el token no es válido, se lanza una excepción indicando acceso no autorizado. Finalmente, el filtro llama a `chain.filter(exchange)` para pasar la solicitud al siguiente filtro en la cadena.

La clase `RouteValidator` define una lista de rutas que no requieren autenticación, siendo accesibles públicamente sin necesidad de un token de autorización. El predicado `isSecured` verifica si una solicitud necesita autenticación; si la ruta coincide con alguna de las rutas públicas, se considera no asegurada y no se aplica el filtro de autenticación.

`JwtUtil` proporciona métodos para validar tokens JWT. El método `validateToken` se encarga de validar un token JWT utilizando la clave secreta definida en `SECRET`. La clave secreta se obtiene mediante el método `getSignKey()`, que decodifica la clave en formato Base64.

En el archivo de configuración `application.yml`, se definen las rutas y los microservicios correspondientes a los que se dirigirán las solicitudes. Cada ruta está asociada a un microservicio específico y se especifica el filtro de autenticación `AuthenticationFilter` para las rutas que requieren autenticación JWT. Esta configuración garantiza que las solicitudes se dirijan correctamente a los microservicios adecuados y que las rutas protegidas se autenticuen debidamente utilizando tokens JWT.

Discovery-Server

The screenshot displays the Spring Eureka web interface. The top navigation bar includes the Spring Eureka logo and links for HOME, LAST 1000 SINCE STARTUP, and a search bar. The main content area is divided into several sections:

- System Status:** A table showing environment (test), data center (default), current time (2024-06-06T07:13:21 +0000), uptime (00:20), lease expiration enabled (true), renews threshold (8), and renews (last min) (16).
- DS Replicas:** A table showing the replica for localhost.
- Instances currently registered with Eureka:** A table listing applications, AMIs, availability zones, and status.
- General Info:** A table showing system metrics like total-avail-memory, num-of-cpus, and current-memory-usage.

Environment	test	Current time	2024-06-06T07:13:21 +0000
Data center	default	Uptime	00:20
		Lease expiration enabled	true
		Renews threshold	8
		Renews (last min)	16

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - deployment-services-fcb65758c-776s5-api-gateway-8084
RANKING-SERVICE	n/a (1)	(1)	UP (1) - deployment-services-fcb65758c-776s5-ranking-service-8081
REGISTER-SERVICE	n/a (1)	(1)	UP (1) - deployment-services-fcb65758c-776s5-register-service-8083
RESULTADOS-SERVICE	n/a (1)	(1)	UP (1) - deployment-services-fcb65758c-776s5-resultados-service-8082

Name	Value
total-avail-memory	106mb
num-of-cpus	1
current-memory-usage	40mb (37%)

El artefacto discovery-server actúa como un servidor de descubrimiento de servicios y balanceador de carga, utilizando Eureka Server. Las dependencias necesarias para su funcionamiento incluyen Spring Cloud Starter Netflix Eureka Server, Spring Boot DevTools, Spring Boot Configuration Processor, Project Lombok y Spring Boot Starter Test.

El servidor de descubrimiento Eureka es esencial en la arquitectura de microservicios, ya que permite que los microservicios se registren y descubran otros servicios en el entorno.

En la clase principal, DiscoveryServerApplication, se configura y se arranca el servidor de descubrimiento Eureka. Esta clase está anotada con `@SpringBootApplication`, que la define como una aplicación Spring Boot, y `@EnableEurekaServer`, que habilita el servidor de descubrimiento Eureka en la aplicación. El método `main()` inicia la aplicación Spring Boot, lo que a su vez pone en marcha el servidor de descubrimiento Eureka.

La configuración del servidor de descubrimiento Eureka se define en el archivo `application.properties`. En este archivo, se especifican las siguientes configuraciones:

- **spring.application.name=discovery-server:** Establece el nombre de la aplicación como "discovery-server".
- **eureka.client.register-with-eureka:** false: Indica que este servidor de descubrimiento no se registrará con otro servidor de descubrimiento.
- **eureka.client.fetch-registry:** false: Indica que este servidor de descubrimiento no recuperará registros de otros servidores de descubrimiento.
- **server.port:** 8761: Especifica el puerto en el que se ejecutará el servidor de descubrimiento Eureka.

Estas configuraciones determinan el comportamiento y la funcionalidad del servidor de descubrimiento Eureka, permitiendo una gestión eficiente de los microservicios y el balanceo de carga en el entorno.

Ranking-Service

El artefacto ranking-service se encarga de gestionar el ranking de jugadores, proporcionando endpoints para obtener todos los jugadores y filtrarlos por género. Este servicio depende de varias bibliotecas, incluidas Spring Boot Starter Web, Spring Data JPA, Spring Boot DevTools, Project Lombok y Spring Cloud Starter Netflix Eureka Client.

En el archivo de configuración `application.properties`, se establece la conexión a la base de datos, ya sea desplegada en Azure o utilizando una imagen de Docker. Esto

garantiza que el servicio pueda interactuar con la base de datos de manera eficiente y fiable.

El controlador `JugadorController` maneja las solicitudes HTTP relacionadas con los jugadores. Este controlador proporciona endpoints para obtener todos los jugadores y filtrarlos por género, devolviendo las respuestas como `ResponseEntity` con la lista de jugadores o un código de error en caso de fallo.

La clase `Jugador` define la estructura de los jugadores en el sistema de ranking. Esta clase contiene atributos como nombre, apellidos, número de ranking, puntos, URL de imagen y URL de bandera, que representan las propiedades esenciales de un jugador.

La interfaz `JugadorRepository` extiende `JpaRepository` para realizar operaciones de base de datos relacionadas con la entidad `Jugador`. Además, incluye un método personalizado para buscar jugadores por género, facilitando la filtración de jugadores en función de este atributo.

El servicio de jugadores está compuesto por `JugadorService` y `JugadorServiceImpl`. `JugadorService` declara métodos para obtener todos los jugadores y filtrarlos por género, mientras que `JugadorServiceImpl` implementa estos métodos utilizando `JugadorRepository` para interactuar con la base de datos y proporcionar la lógica de negocio necesaria.

Finalmente, se configura el registro en un servidor Eureka para el descubrimiento de servicios, estableciendo la URL del servidor Eureka en `eureka.client.serviceUrl.defaultZone`. Esta configuración permite que el servicio se registre y sea descubierto por otros servicios en el entorno de microservicios.

En resumen, el servicio de ranking de jugadores proporciona una API REST para gestionar jugadores en un sistema de ranking, facilitando operaciones como la obtención de todos los jugadores y el filtrado por género, asegurando una gestión eficiente y organizada de los datos de los jugadores.

Resultados-Service

El artefacto "resultados-service" es un microservicio que proporciona una API para gestionar información sobre partidos y torneos. Este servicio utiliza varias dependencias como Spring Boot Starter Data JPA, Spring Boot Starter Web, Spring Boot DevTools, MySQL Connector/J, Project Lombok, Spring Boot Starter Test, Spring Cloud Starter Netflix Eureka Client, JUnit Jupiter Engine y Mockito Core.

Dentro de este microservicio, se definen las entidades JPA "Partido" y "Torneo". La entidad "Partido" está anotada con `@Entity` y `@Table` para mapearla a una tabla en la base de datos, e incluye campos como `ronda`, `fecha_hora`, `pareja_1`, `pareja_2` y

resultado. La entidad "Torneo" también está anotada con `@Entity` y `@Table`, e incluye campos como ciudad, tipo y fechatorneo. Para acceder a los datos de estas entidades en la base de datos, se utilizan los repositorios JPA "PartidoRepository" y "TorneoRepository", que extienden `JpaRepository` y proporcionan métodos como `findAll()` y `findByTorneoId()`.

En cuanto a la lógica de negocio, "PartidoService" define métodos como `getAllPartidos()` y `getPartidosByTorneo(Long torneoId)`, mientras que "TorneoService" define métodos como `getAllTorneos()` y `getTorneosByYear(int year)`. Las implementaciones de estos servicios, "PartidoServiceImpl" y "TorneoServiceImpl", manejan la lógica de negocio utilizando los repositorios correspondientes para acceder y manipular los datos.

Este conjunto de archivos forma un sistema Java/Spring Boot que ofrece una API robusta para administrar información sobre partidos y torneos, utilizando una base de datos relacional a través de JPA y Spring Data, y apoyado por herramientas de prueba y desarrollo ágiles como JUnit y Mockito.

Register-Service

El artefacto "register-service" es un microservicio dedicado al registro y login de usuarios. Este servicio se construye utilizando varias dependencias como Spring Boot Starter Data JPA, Spring Boot Starter Security, Spring Boot Starter Validation, Spring Boot Starter Web, Spring Boot DevTools, MySQL Connector J, Project Lombok, Spring Boot Starter Test, Spring Security Test, Spring Cloud Starter Netflix Eureka Client, jjwt-api, jjwt-impl, jjwt-jackson, dev.samstevens.totp, y el jacoco-maven-plugin. Dentro de este microservicio, se manejan modelos de persistencia, destacando el modelo "User", que representa los datos de un usuario, incluyendo atributos como id, nombre, apellidos, edad, correo, dirección, teléfono, contraseña, estado, rol, mfaEnabled, secret, createDate y editionDate. Además, se implementan DTOs como "UserDtoRegister" y "UserDto" para facilitar el registro y la representación de datos de usuario.

Para la gestión de excepciones, se utiliza "CustomExceptionHandler", que maneja excepciones como `UsernameNotFoundException` y `UsuarioNoHabilitadoExeption`. La configuración y el cliente HTTP se gestionan a través de "AppConfig", que configura un bean `RestTemplate`. Adicionalmente, se incluyen clases como "AuthResponse" para representar respuestas de autenticación, "LoginRequest" para solicitudes de inicio de sesión y "RegisterRequest" para solicitudes de registro. La autenticación basada en JWT se maneja con "JWTAuthenticationFilter", que extrae y valida tokens. Las implementaciones del servicio de autenticación y gestión de tokens JWT se realizan en "AuthServiceImpl" y "JWTServiceImpl", respectivamente. Finalmente, "AuthController" actúa como el controlador para la gestión de autenticación.

3.1.2 Bases de datos

Para almacenar todo los datos que me proporciona el script de WebScraping hecho con Python, he creado 3 bases de datos **MySQL** las cuales son:

- **ResultadosMaster BBDD:** En esta BBDD almaceno todos los torneos que se han jugado en esta liga de pádel y también todos los partidos de estos torneos. Consta de 2 tablas que son:

Table: **partidos**

Columns:

id	bigint AI PK
fecha_hora	varchar(255)
pareja_1	varchar(255)
pareja_2	varchar(255)
resultado	varchar(255)
ronda	varchar(255)
torneo_id	bigint

Table: **torneos**

Columns:

id	bigint AI PK
ciudad	varchar(255)
fechatorneo	varchar(255)
tipo	varchar(255)

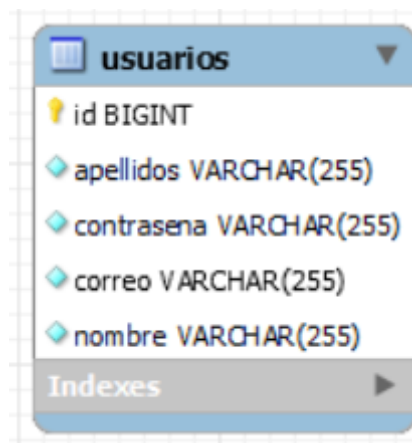
- **Top50Players BBDD:** En esta BBDD almaceno el top 50 de jugadores de ambos géneros con más puntuación de esta liga. Esta BBDD consta de 1 sola tabla:

Table: jugadores

Columns:

id	bigint AI PK
apellidos	varchar(255)
bandera_url	varchar(255)
genero	varchar(255)
imagen_url	varchar(255)
nombre	varchar(255)
numero_ranking	bigint
puntos	varchar(255)

- **Usuarios BBDD:** Como su nombre indica, en esta BBDD almaceno todos los usuarios de la aplicación. Consta de 1 sola tabla:



Para conseguir todos los datos necesarios para este proyecto, he creado dos Scripts de **Web Scraping con Python**. Uno para la obtención de todos los resultados y partidos de esta liga y otro para el ranking 50 de ambos géneros.

Aquí hay una breve explicación de estos Scripts:

Ambos scripts son herramientas de extracción y procesamiento de datos que utilizan diferentes técnicas para obtener información específica de sitios web y almacenarla en una base de datos MySQL. Aunque tienen objetivos diferentes, comparten el uso de bibliotecas como `mysql.connector` para interactuar con la base de datos, **BeautifulSoup** para analizar el HTML de las páginas web y `requests` para realizar solicitudes HTTP.

El **primer** script se enfoca en la extracción de datos de torneos de pádel desde un sitio web específico. Utiliza la biblioteca **BeautifulSoup** para analizar el HTML de la página web de A1 Padel, extraer información como la ubicación, el tipo de torneo y los datos de los partidos, y luego insertar estos datos en mi base de datos MySQL utilizando `mysql.connector`. También utiliza la biblioteca **requests** para realizar solicitudes HTTP a las páginas web de los torneos.

Por otro lado, el **segundo** script se centra en extraer datos de **clasificaciones** de jugadores de pádel de la página web de A1 Padel y almacenarlos en una base de datos MySQL. Utiliza la biblioteca **Selenium** para interactuar con la página web y obtener datos dinámicos que no están presentes en el HTML estático. Luego, utiliza **BeautifulSoup** para analizar el HTML de las páginas web de las clasificaciones, extraer información como el nombre del jugador, su ranking y puntos, así como las URLs de las imágenes del jugador y la bandera de su país. Finalmente, utilizo `mysql.connector` para insertar estos datos en la base de datos MySQL.

3.1.3 Spring Security

Para garantizar la seguridad de mi aplicación, he implementado Spring Security utilizando tokens JWT. Este enfoque incluye varios métodos importantes. Primero, el método **getToken** se encarga de generar un token JWT para el usuario especificado. Este método toma como parámetro los detalles del usuario y devuelve un token JWT en forma de cadena.

Otro aspecto crucial es la verificación de la validez del token. El método **isTokenValid** se utiliza para comprobar si un token JWT es válido en relación con los detalles del usuario proporcionados. Este método recibe el token JWT y los detalles del usuario como parámetros y devuelve un valor booleano: `true` si el token es válido y `false` en caso contrario.

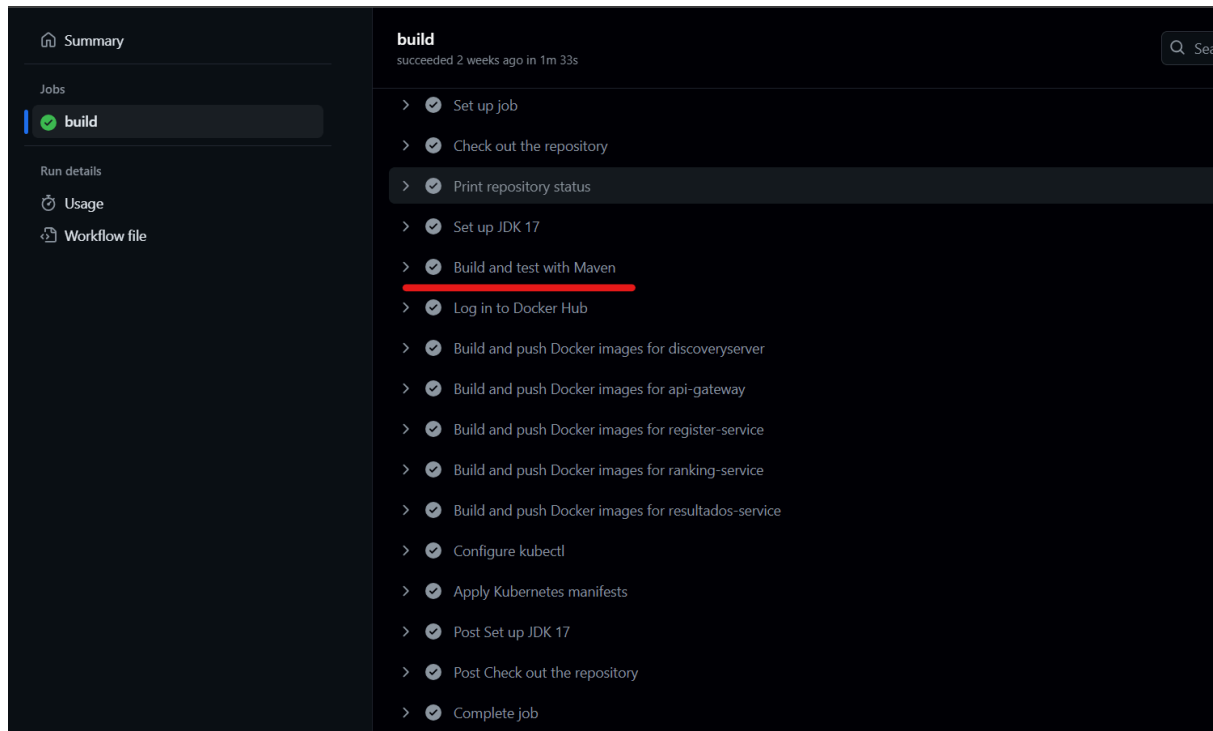
Finalmente, para asegurar la integridad de los tokens, el método **getKey** proporciona la clave utilizada para firmar los tokens JWT. Este método no requiere parámetros y devuelve la clave de firma en forma de objeto `Key`. Estos métodos combinados aseguran que mi aplicación mantenga un alto nivel de seguridad utilizando Spring Security y JWT.

3.1.4 Testing

Para tener un código que cubra todos los casos de uso posibles, desarrollé test unitarios con las librerías que nos proporciona JUNIT. Cree test en los servicios

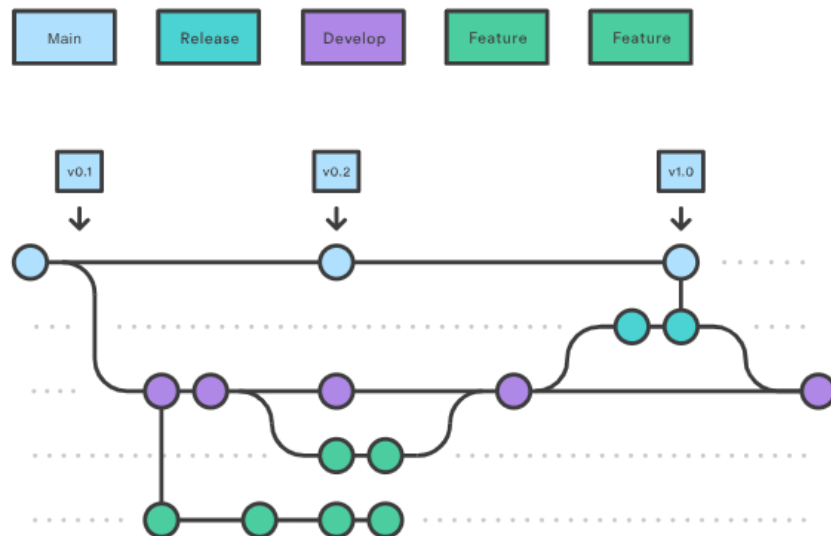
funcionales, como son principalmente Ranking-Service, Register-Service y Resultados-Service.

Estos test se pasan automáticamente en el WorkFlow de GitHub Actions cada vez que se realiza un commit del código al repositorio de GitHub.



3.1.5 GitFlow

Ramas de publicación



Para el correcto control de versiones en el desarrollo de este proyecto, he seguido el modelo de creación de ramas GitFlow. Creando mi rama de desarrollo Feature, Develop, Release y por último Main.

3.2 Automatización

Para facilitar el desarrollo, el testeo y la implementación continua de nuevas funcionalidades he hecho uso de Docker, GitHub Actions y Azure Functions. Todas estas herramientas las explico detalladamente a continuación.

3.2.1 Dockerización

En este apartado, se detalla el proceso de dockerización utilizando Docker Compose para gestionar los contenedores de nuestra aplicación. El archivo `docker-compose.yml` especifica la configuración necesaria.

La versión de Docker Compose utilizada es la 3.3. En la sección `services`, se definen los servicios que se ejecutarán como contenedores Docker, incluyendo:

- **register-service:** Este contenedor gestiona el servicio de registro y login de la aplicación.
- **ranking-service:** Contenedor para el servicio de ranking de la aplicación.

- **resultados-service:** Otro contenedor dedicado al servicio de resultados.
- **api-gateway:** Contenedor del servicio de API Gateway, encargado de redirigir las solicitudes a los otros servicios.
- **discovery-server:** Contenedor del servicio Eureka para el balanceo de carga de los servicios.

Además, se define una red específica para los contenedores en la sección networks.

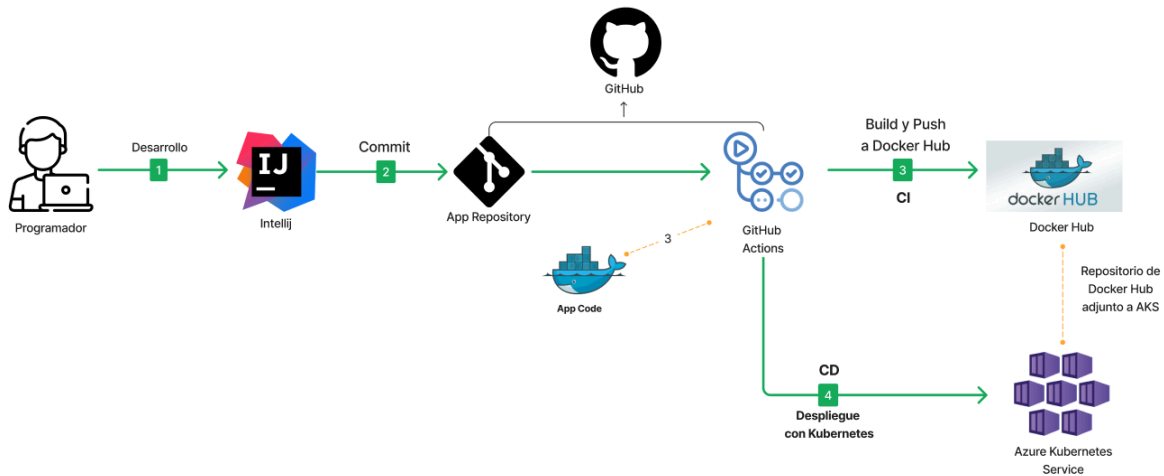
Al implementar estos contenedores, pueden surgir varios problemas:

- Puerto en uso: Si otro programa está utilizando el puerto especificado, asegúrate de liberarlo o cambia el puerto en el archivo docker-compose.yml.
- Problemas de conexión del microservicio a la base de datos: Asegúrate de que todos los atributos necesarios para la conexión estén configurados correctamente, y que las credenciales (DATABASE_USERNAME y DATABASE_PASSWORD) coincidan con las configuradas en la base de datos, incluso si el microservicio no se comunica directamente con la base de datos.
- Problemas de comunicación entre el cliente y el servidor: Verifica que las rutas en tu controlador estén configuradas correctamente y que cada contenedor esté utilizando el archivo .jar correspondiente al iniciar.
- Problemas de recursos insuficientes: Si los contenedores se detienen inesperadamente, revisa si Docker tiene suficientes recursos asignados. Puedes ajustar esta configuración en Docker Desktop y ejecutar docker system prune -a para eliminar datos antiguos que puedan causar conflictos.
- Errores de compilación: Si encuentras errores al construir las imágenes Docker, verifica que todas las dependencias estén configuradas correctamente y que el código esté actualizado. Ejecuta mvn clean install para asegurarte de que todas las dependencias se resuelvan adecuadamente.

Implementar y mantener la dockerización correctamente asegurará un entorno de ejecución estable y eficiente para mi aplicación.

3.2.2 GitHub Actions

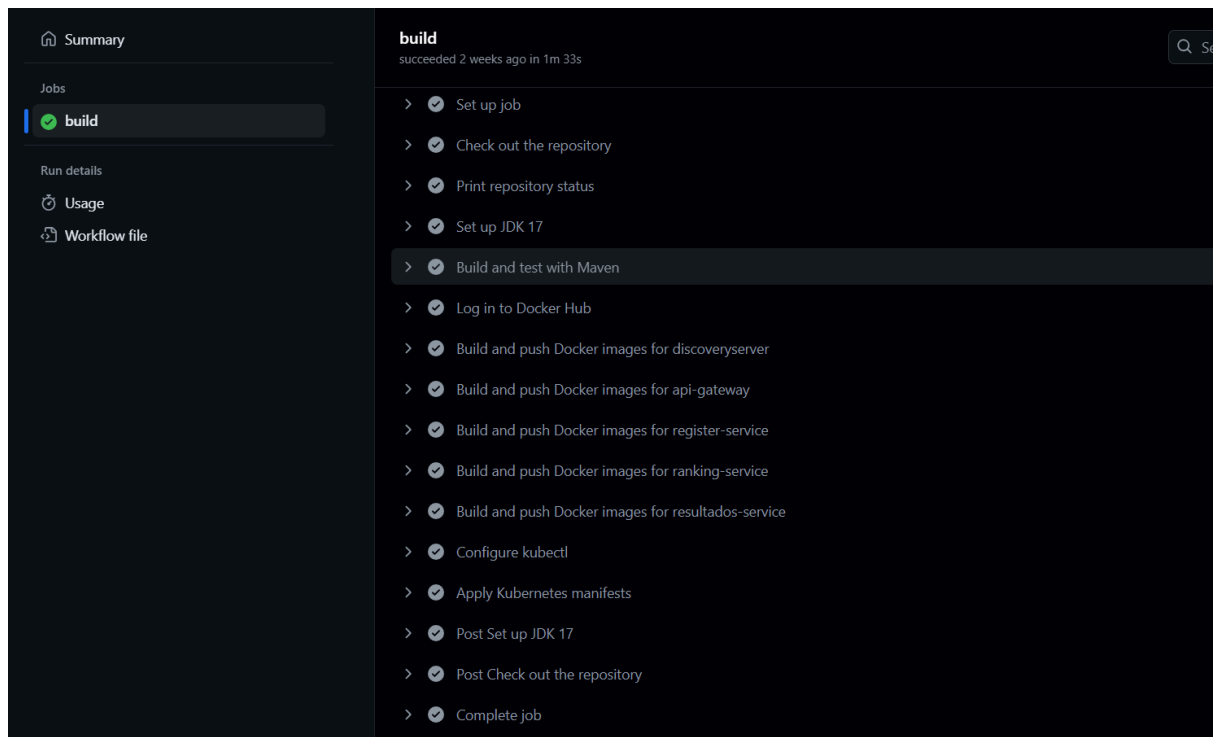
Automatización/Despliegue



Gracias a esta herramienta, he conseguido que por cada commit hecho a mi rama de mi repositorio de Github, se ejecuten automáticamente los siguientes procesos: Build, Test Unitarios, Subida de imágenes Docker a Docker Hub y finalmente despliegue en Azure usando Kubernetes.

Para automatizar procesos en mi repositorio, utilizo GitHub Actions creando workflows específicos. Primero, hay que crear una estructura de carpetas en la raíz del proyecto: una carpeta `.github` y dentro de esta, otra carpeta llamada `workflows`. En esta ubicación se definen los workflows necesarios, utilizando archivos con la extensión `.yaml`.

Una vez configurado, se puede observar la ejecución de los workflows en la pestaña Actions del repositorio en GitHub. Para gestionar usuarios y contraseñas de manera segura, se debe ir a la pestaña Settings del repositorio, luego a Secrets and Variables, y finalmente seleccionar Actions. Aquí podremos definir las credenciales sin tener que incluirlas directamente en el código.



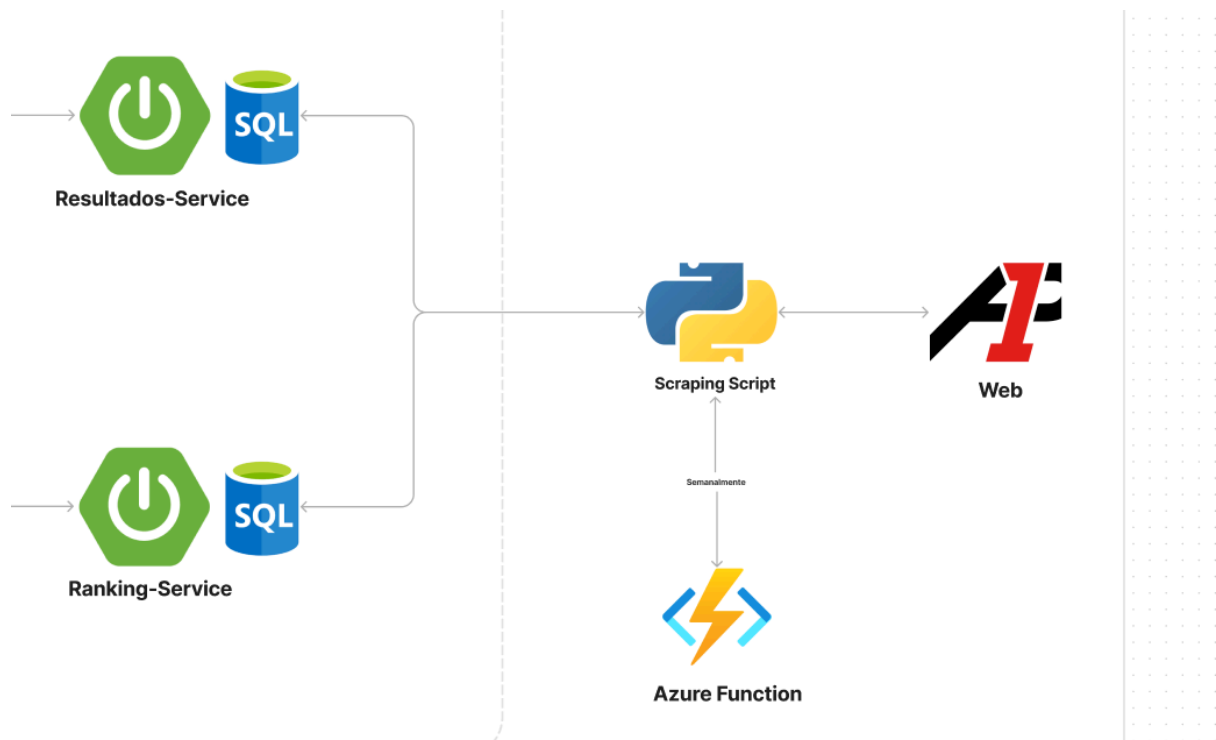
El archivo de workflow en GitHub Actions, con extensión .yml, define las acciones a automatizar en mi repositorio. Este archivo incluye varios elementos clave:

- **Nombre del archivo:** Se ubica en .github/workflows dentro del repositorio.
- **Eventos de activación:** Permite especificar cuándo se debe activar el workflow, como por ejemplo, al hacer push a una rama específica o al crear una nueva versión.
- **Jobs:** Un workflow puede contener uno o más "jobs", que son unidades de trabajo independientes que se ejecutan en entornos separados. Cada job consta de una serie de pasos para realizar tareas específicas.
- **Pasos (steps):** Cada job está compuesto por varios pasos, que son las acciones individuales realizadas durante la ejecución del job. Estos pasos pueden incluir la compilación del código, la ejecución de pruebas, el despliegue de la aplicación, entre otros.
- **Acciones (actions):** Se pueden utilizar acciones predefinidas de GitHub o acciones personalizadas para llevar a cabo tareas específicas en los pasos. Las acciones son unidades reutilizables de código que facilitan la implementación de workflows.
- **Variables de entorno y secretos:** Se pueden definir variables de entorno y secretos para almacenar información sensible, como contraseñas o tokens de acceso, y utilizarlas en el workflow.

- **Flujo de ejecución:** Cuando un evento desencadena el workflow, GitHub Actions ejecuta los jobs y pasos definidos en el archivo según las especificaciones proporcionadas.
- **Registro de ejecución:** Tras completar la ejecución del workflow, es posible revisar el registro de ejecución para ver el resultado de cada paso y diagnosticar cualquier problema que pueda haber ocurrido.

Estos elementos permiten automatizar y gestionar eficazmente las tareas en el repositorio utilizando GitHub Actions, asegurando un flujo de trabajo más eficiente y seguro.

3.2.3 Azure Functions



Azure Functions es un servicio de computación sin servidor que te permite ejecutar pequeños fragmentos de código (funciones) en la nube sin tener que gestionar la infraestructura subyacente. Las funciones se pueden desencadenar por eventos como solicitudes HTTP, mensajes en colas, cambios en bases de datos, y más.

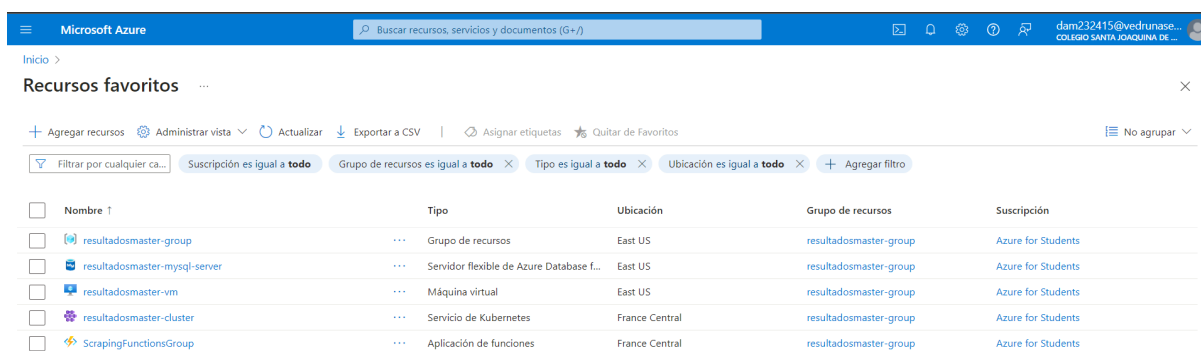
Dentro de este servicio, hay diferentes tipos de funciones. Para mi proyecto, he hecho uso de dos HTTP Trigger. Esta función es un tipo de desencadenador que permite que una función se ejecute en respuesta a una solicitud HTTP. Esto es útil cuando deseas que tu función actúe como un endpoint web o en mi caso, scrapear los datos de una URL cada cierto tiempo.

Mi función se ejecuta todos los lunes a las 15:00 ya que ese día a esa hora, las finales del torneo anterior ya se habrán realizado y estarán los datos disponibles en la web oficial de A1Padel.

3.3 Despliegue en Azure

Para poder hacer uso de la aplicación en cualquier momento, he usado las cuentas de Microsoft Azure que nos ha proporcionado el centro para hacer un despliegue de todos los recursos necesarios para un funcionamiento correcto de la aplicación.

3.3.1 Recursos



Microsoft Azure					
Inicio >					
Recursos favoritos					
+ Agregar recursos Administrar vista Actualizar Exportar a CSV Asignar etiquetas Quitar de Favoritos No agrupar					
Filtrar por cualquier ca... Suscripción es igual a todo Grupo de recursos es igual a todo Tipo es igual a todo Ubicación es igual a todo + Agregar filtro					
Nombre ↑	Tipo	Ubicación	Grupo de recursos	Suscripción	
resultadosmaster-group	Grupo de recursos	East US	resultadosmaster-group	Azure for Students	
resultadosmaster-mysql-server	Servidor flexible de Azure Database f...	East US	resultadosmaster-group	Azure for Students	
resultadosmaster-vm	Máquina virtual	East US	resultadosmaster-group	Azure for Students	
resultadosmaster-cluster	Servicio de Kubernetes	France Central	resultadosmaster-group	Azure for Students	
ScrapingFunctionsGroup	Aplicación de funciones	France Central	resultadosmaster-group	Azure for Students	

Para desplegar la aplicación en Azure, es necesario crear y configurar varios recursos. En primer lugar, se crea un servidor MySQL, una máquina virtual, un Servicio de Kubernetes y Azure Functions.

Una vez creados estos recursos, se pasa a su configuración. Para el servidor MySQL, se accede a la sección Redes y activé las Reglas del firewall. Aquí, se modifican las reglas según mis necesidades, introduciendo al menos la IP de la máquina virtual. En el caso de la máquina virtual, hay que dirigirse a Redes/Configuración de red y cree una ACL del puerto/Regla de puerto de entrada, añadiendo reglas para cada puerto que se fuera a utilizar en el proyecto.

Para el servicio de kubernetes, se realiza toda la instalación conveniente. Finalmente, se configura Azure Functions según los requisitos de mi proyecto, siguiendo las guías de Azure para la implementación y configuración de estas funciones.

Estos pasos me aseguraron que los recursos estén correctamente configurados para el despliegue de la aplicación en Azure, proporcionando un entorno robusto y seguro para su funcionamiento.

3.3.2 Kubernetes

Para la configuración de Kubernetes, se crean dos archivos que son `microdeployment.yaml` y `microservice.yaml`. El primero de ellos tiene la función de desplegar los servicios configurados previamente, por otro lado `microservice` sirve para hacer públicos estos servicios y poder acceder a ellos a través de una IP externa.

Pasando a la parte de consola, los pasos son los siguientes:

1. Crear el espacio.
2. Levantar los despliegues con `microdeployment.yaml`.
3. Hacer públicos los despliegues levantando `microservice.yaml`.

4. Explicación del Front-End.

El Front-End de esta aplicación está realizado con el Framework de Android Studios. El proyecto lo he dividido en paquetes, los cuales son los siguientes:

- UI: Dentro de este paquete se encuentran los Activity y los Fragments. Dentro de estas clases se pueden encontrar el activity de Login, Registro o los fragmentos de Resultados, Ranking, Home o Contenedor. Todas estas clases realizan las funciones que sus nombres indican. En varias de ellas usó la librería Retrofit para establecer la conexión a las bases de datos.
- Registro: Dentro de este paquete se encuentran todas las clases relacionadas con la seguridad, registro y logueo de usuarios.
- Model: En este paquete se encuentran los modelos de las entidades que usó en los fragmentos y activity. Estas clases son Jugador, Partido, Torneo y User.
- Interfaces: Como su nombre lo indica dentro de este paquete se encuentran las interfaces de `RankingInterface` y `ResultadosAPI`, dentro de ella defino los métodos correspondientes para enviarle las requests a los servicios.
- Adapters: Por último, en este paquete he creado adaptadores para mostrar de una forma correcta, eficiente y dinámica las listas de jugadores o torneos en la interfaz de estas clases.

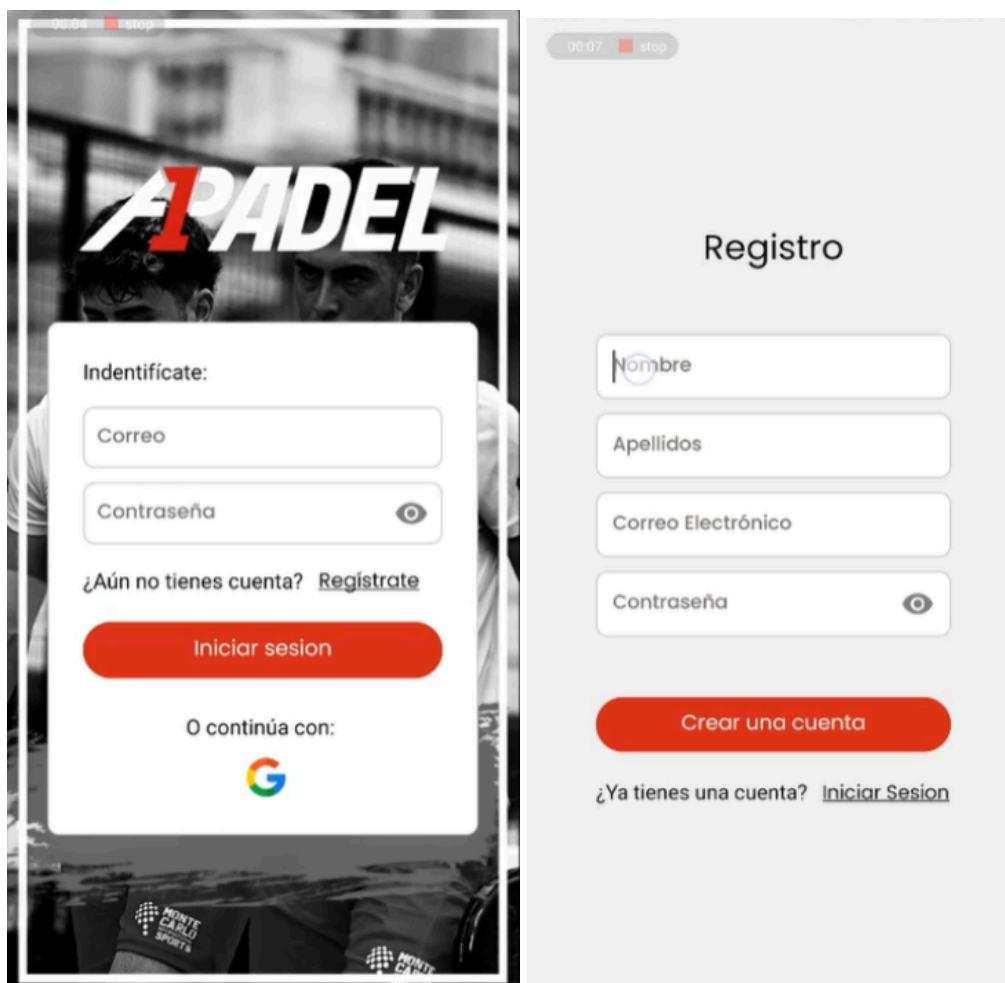
En cuanto a distribución, esta página consta de layouts como pueden ser el de **Home**, **Resultados**, **Ranking e Historia**, por otro lado los activity que son el **ContenedorFragmentActivity**, **Login y Registro**. Estos layouts y activity pueden ser visualizados en el siguiente punto.

5. Guia de uso.

5.1 Desde la Propia aplicación

Al iniciar la aplicación, se presentará una pantalla de autenticación o registro donde constan dos opciones para la identificación del usuario:

- **Login mediante cuenta de Google**, el cual te permitirá no tener que registrarte introduciendo todos tus datos.
- **Login/Registro habitual**. Para identificarse de esta forma, el usuario deberá registrarse. Se solicitará al usuario ingresar datos como su nombre, apellidos, correo electrónico y contraseña. Posteriormente, si el registro es exitoso se redirigirá al usuario a la pantalla de Login, donde deberá introducir los datos que se piden previamente introducidos en el registro.



Una vez completada la identificación correctamente, se redirigirá al usuario a la pantalla de inicio donde hay una pequeña descripción de la liga sobre la que está hecha la aplicación.



- Consultar resultados

Para consultar los resultados de cualquier torneo que haya realizado esta liga, el usuario deberá seleccionar en este orden los siguientes campos:

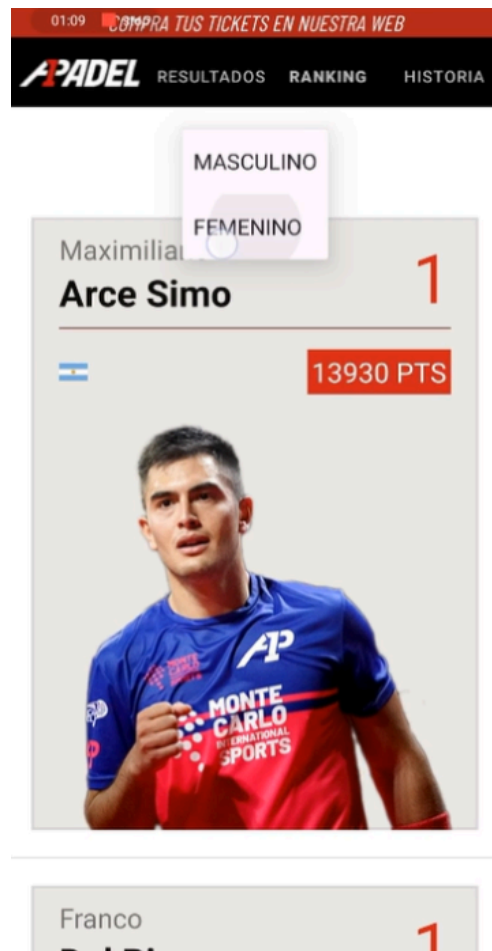
- Año del que quiere ver los torneos jugados.
- Una vez escogido el año, aparecerán debajo un Scroll con todos los torneos disponibles de ese año con información relevante sobre ellos como Fecha, Nombre del torneo y Ubicación.
- Una vez localizado el torneo, clicar sobre él y automáticamente aparecerán todos los partidos de ese torneo con información como: Ronda, Fecha y Hora, Pareja 1, Pareja 2 y Resultado.

01:05 COMPR TUS TICKETS EN NUESTRA WEB				
PADEL RESULTADOS RANKING HISTORIA				
TEMPORADA 2022				
TORNEO	FutureAPT 23 - 27 Marzo 2022 BUENOS AIRES			
	Master 27 - 3 Abril 2022 BUENOS AIRES			
Ronda	Fecha y Hora	Pareja 1	Pareja 2	Resultado
	10:15	Damar, Jordane Klein	Sassano, Fabricio Peiron	
R4	28/03/22 16:00	Juan Martín Cesca, Esteban Genoud	Javier Reiter, Matias Gonzalez	3-6 / 1-6
R4	28/03/22 16:00	Leandro Roman Augsburger, Juan Ignacio De Pascual	Marcos Andrada, Juan Pablo Andrada	6-3 / 1-6 / 7-6
R4	28/03/22 16:00	Felipe Calleja, Ramiro Valenzuela	Denis Ignacio Caluva, Emanuel Mandrile	6-2 / 7-6
R4	28/03/22 16:00	Julián Leite, Dylan Demian Cuello	Gonzalo Sassano, Fabricio Peiron	6-3 / 5-7 / 4-6
R16	29/03/22	Facundo	Miguel	2-6 / 6-4 /

- Consultar ranking

Este ranking está actualizado al Top 50 tanto masculino como femenino de la liga de A1 Padel.

Para consultarlo, el usuario deberá acceder a él clicando en la opción Ranking del menú superior. Una vez ahí, por defecto aparecerá el ranking de los jugadores masculinos con información como Nombre y Apellidos, Nacionalidad, Ranking y número de Puntos. Si desea consultar el ranking femenino solo tendrá que seleccionarlo en el menú desplegable.



5.2 Desde Postman

- Registrar/Login

Registro de usuario:

URL: POST <http://98.66.187.58:8084/auth/register>

Descripción: Registra un nuevo usuario.

Cuerpo de la solicitud:

```
{
  "nombre": "admin",
  "apellidos": "admin",
  "correo": "admin@example.com",
  "contrasena": "admin"
}
```


Login de usuario:

URL: POST http://98.66.187.58:8084/auth/login

Descripción: Autentica a un usuario y devuelve un token de autenticación.

Cuerpo de la solicitud:

```
{  
  "correo": "admin@example.com",  
  "contrasena": "admin"  
}
```

- Consultar ranking

Obtener todos los jugadores:

URL: GET http://98.66.187.58:8084/api/v1/ranking

Descripción: Obtiene la lista de todos los jugadores.

***Añadir el token que genera el login en apartado Authorization>Baerer Token**

Obtener jugadores por género:

URL: GET http://98.66.187.58:8084/api/v1/ranking/{genero}

Descripción: Obtiene la lista de jugadores filtrados por género.

Parámetro de ruta: {género} - El género de los jugadores a buscar (ejemplo: "masculino" o "femenino").

***Añadir el token que genera el login en apartado Authorization>Baerer Token**

- Consultar Partidos

Obtener todos los partidos:

URL: GET http://98.66.187.58:8084/api/v1/partidos

Descripción: Obtiene la lista de todos los partidos.

***Añadir el token que genera el login en apartado Authorization>Baerer Token**

Obtener partidos por torneo:

URL: GET http://98.66.187.58:8084/api/v1/partidos/torneo/{torneold}/partidos

Descripción: Obtiene la lista de partidos de un torneo específico.

Parámetro de ruta: {torneold} - El ID del torneo cuyos partidos se quieren obtener.

***Añadir el token que genera el login en apartado Authorization>Baerer Token**

- Consultar Torneos:

Obtener todos los torneos:

URL: GET <http://98.66.187.58:8084/api/v1/torneos>

Descripción: Obtiene la lista de todos los torneos.

***Añadir el token que genera el login en apartado Authorization>Baerer Token**

Obtener torneos por año:

URL: GET <http://<tu-dominio>/api/v1/torneos/year/{year}>

Descripción: Obtiene la lista de torneos de un año específico.

Parámetro de ruta: {year} - El año de los torneos a buscar.

***Añadir el token que genera el login en apartado Authorization>Baerer Token**

6. Diagramas.

Para una mejor visualización de los diagramas, puedes visitar la página de FigJam
(Recomiendo entrar):

<https://www.figma.com/board/OWgEbq6kO5ArnPZ2uWYJiO/Diagramas-TFG?node-id=0-1&t=XO9feNKKWkWBNW6Q-1>

Diagrama de casos de uso

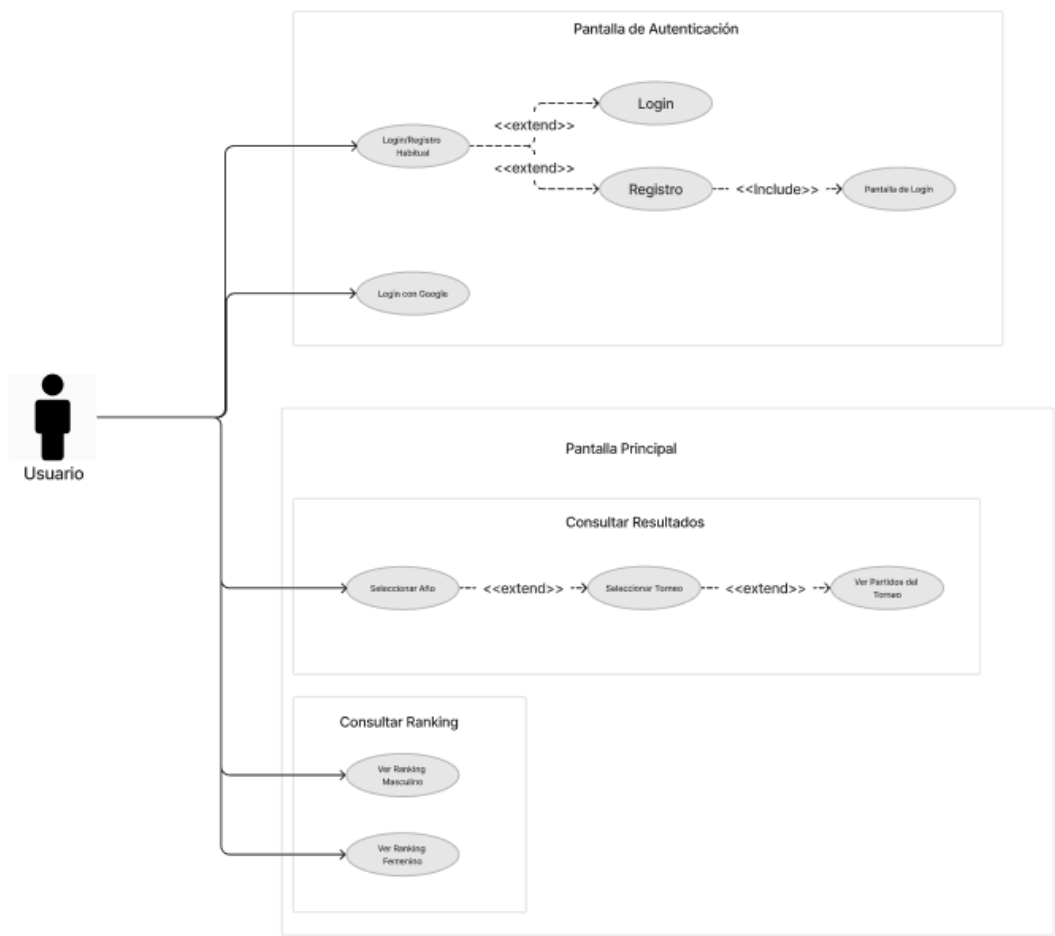


Diagrama de clases

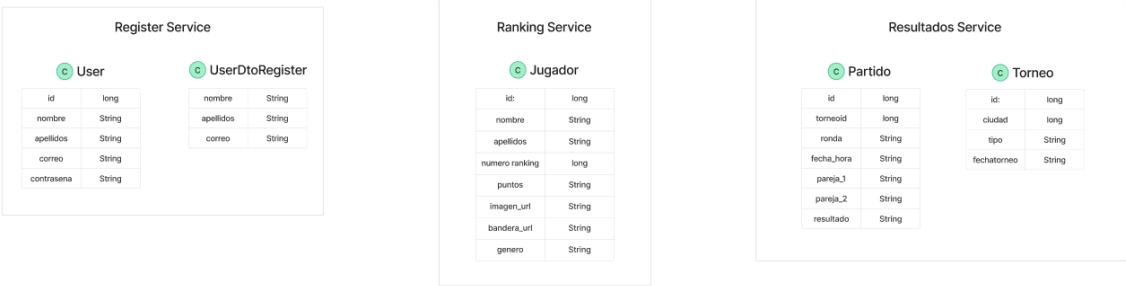
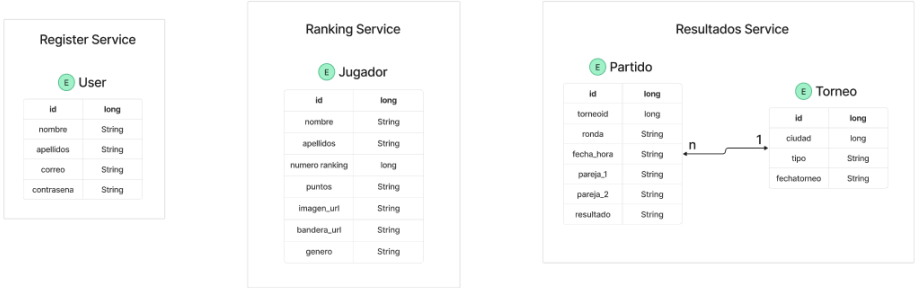


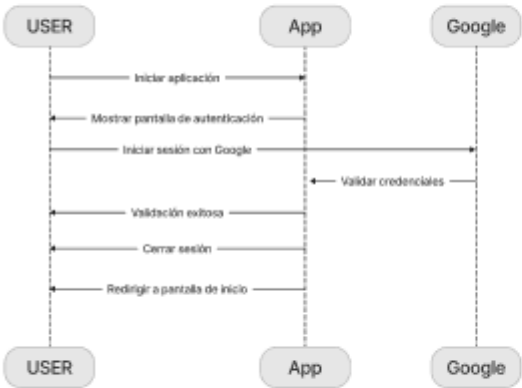
Diagrama de Entidad-Relación



Casos de prueba

Flujo de autenticación y registro

Caso 1: Inicio de sesión con cuenta de Google

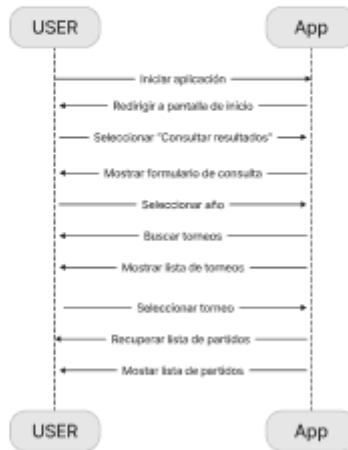


Caso 2: Registro y inicio de sesión habitual



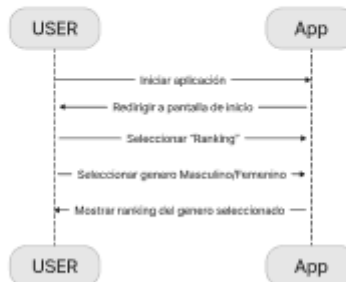
Flujo de consulta de resultados

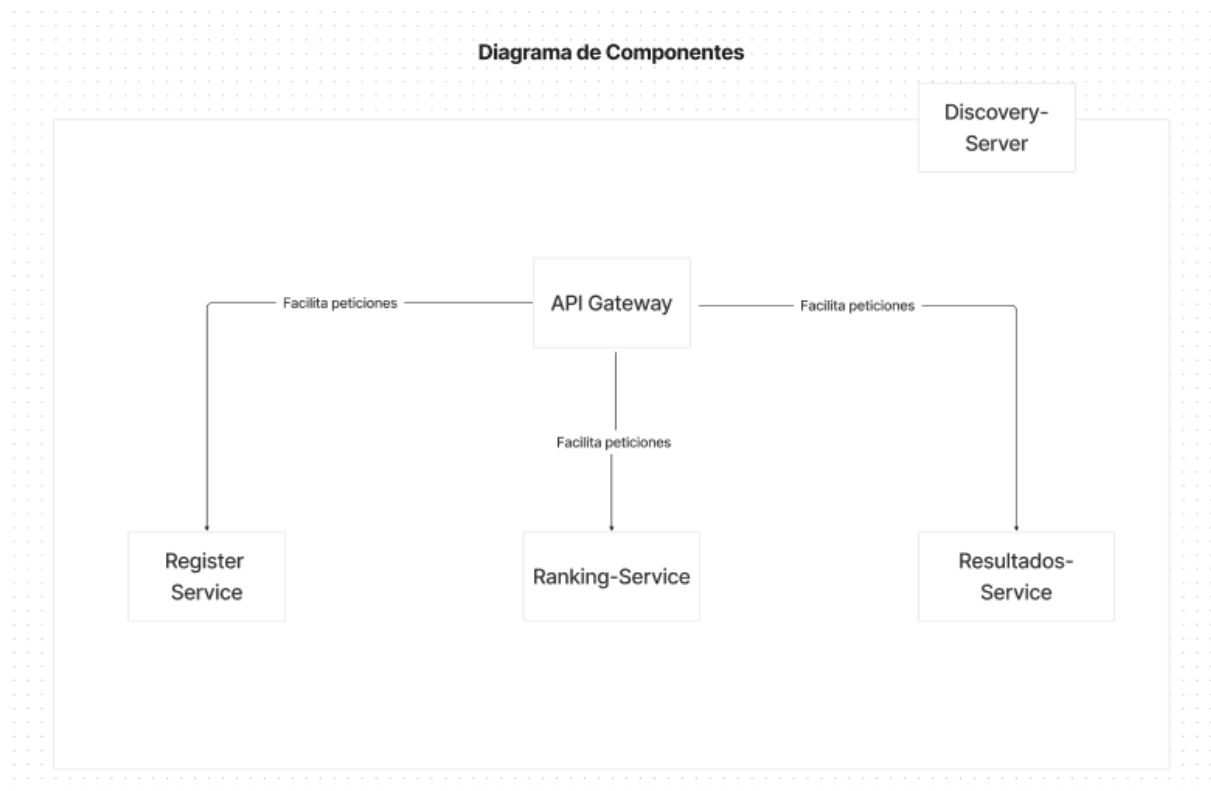
Caso 1: Consultar resultados de un torneo específico



Flujo de consulta de ranking

Caso 1: Consultar uno de los dos rankings





7. Historia del proyecto.

En este apartado pretendo contar un poco todas las fases del proyecto, tal cual han pasado de una forma parecida a la de un diario.

El proyecto “comenzó” aproximadamente en Octubre de 2023, dando inicio a mi 2º Año de Desarrollo de Aplicaciones Multiplataforma.

¿Porque escribo el “comenzó” entre comillas? Sobre estas fechas, como comento, entró en mi cabeza el pensamiento de que debía tener una idea clara para así poder empezar cuanto antes para poder ir haciendo todo a un ritmo tranquilo, y así fue.

Llegué a la conclusión de que mezclar la programación y el pádel era una buena idea, desde un principio la función de esta aplicación iba a ser la de poder crear cuadros de torneos introduciendo X cantidad de jugadores. La aplicación generaría todas las rondas (Dieciseisavos, Octavos, Cuartos...), además de eso, se podría ir apuntando los resultados para así poder tener un registro de estos partidos y torneos.

Finalmente esta idea la abandoné y me decante por la segunda opción, una aplicación que proporcionará todos los resultados de todos los partidos y el ranking de toda la historia de una de las ligas de pádel más importante en la actualidad, A1 Pádel.

Una vez con la idea clara, comencé a pensar cómo hacerlo. Me surgieron preguntas como ¿Lenguajes de programación? ¿Android, Escritorio, Multiplataforma? ¿Bases de datos? Estuve aproximadamente un par de meses estudiando, probando y analizando cómo dar respuestas a estas preguntas. Probe bases de datos como MongoDB, Firebase y lenguajes como Flutter, Kotlin, Angular, etc. Al final me decante por una aplicación Android con Java y

MySQL, ya que estas tecnologías eran con las que más cómodo me sentía en ese momento.

Con estas respuestas más o menos encontradas, llegué a la altura del curso donde nos enseñaron los microservicios, esto fue lo que me dio el último empujón para poder empezar a tocar código.

A mediados de Febrero de 2024, una vez hechas las entregas de todos los proyectos finales de la segunda evaluación, volví con el desarrollo del proyecto. Aprendí conceptos y buenas prácticas como API Gateway, Eureka server, modularización de microservicios, etc.

Poco a poco iban avanzando las semanas y el proyecto iba tomando forma, a mediados de Abril el proyecto estaba casi terminado. Tenía hecho todo el BackEnd, y quedaban por terminar detalles del FrontEnd y todo el despliegue. Con esto último, me puse durante las últimas semanas de Abril y mediados de Mayo. Oficialmente, este proyecto lo acabé la segunda semana de Mayo de 2024, desde esa fecha a la actualidad, he estado y estoy repasando detalles, mejorando documentación y preparándome la defensa del mismo.

8. Justificación.

¿Por qué he realizado este proyecto?

Pienso que un TFG debe tener una parte creativa pero también resolutive. Con esto quiero decir que, debes crear algo en gran parte desde 0 por ti y que a la misma vez resuelva la inexistencia de algo que en tu día a día crees que sería útil.

En mi caso, soy un jugador habitual de este deporte y seguidor de esta gran liga de pádel. Ocasionalmente me gusta consultar los resultados de torneos que por un motivo u otro no puedo llevar al día y estar informado sobre que pareja gana, pierde etc. El problema surge cuando yo quiero consultar estos datos, esta liga consta de una web en la cual en mi opinión, debes pasar muchos filtros para conseguir encontrar estos resultados. Con mi aplicación he pretendido tener algo simple, pero útil. Donde en pocos segundos, tengas disponibilidad total de estos datos.

9. Objetivos.

Los objetivos principales durante el desarrollo de mi proyecto han sido los siguientes:

- Plasmar totalmente mi idea en el proyecto y al mismo tiempo poder personalizarlo a mi gusto.
- Abarcar un rango de tecnologías amplio, para a la misma vez de ir desarrollando ir aprendiendo.
- Llevar una organización y rutina diaria. Para poder cumplir con los plazos de una manera tranquila y holgada.

10. Motivación.

Mi motivación ha sido poder crear algo funcional hecho totalmente desde mi ingenio, mezclando dos aspectos importantes de mi vida como son el desarrollo de software y el pádel.

Para enfocarlo de una manera más laboral, desde un principio he tenido la mentalidad que este proyecto me lo ha encargado un cliente final. Esto me ha hecho poder tomármelo de una manera más seria y pienso que me ha ayudado notablemente.

11. Conclusión.

Era la primera vez que me enfrentaba a un proyecto de esta magnitud. Desde un principio sabía que llevando una buena organización y que sería capaz de realizarlo, tras muchas horas de aprendizaje y trabajo duro he podido sacarlo adelante además de plasmar todo lo aprendido durante mi etapa cursando estos estudios.

Soy consciente de todas las posibles mejoras del proyecto por lo que en el futuro lo seguiré mejorando. Con este proyecto me ha quedado claro que con disciplina, organización y esfuerzo todo lo que te propongas es posible. Como dicen muchos "Todo lo que puedas imaginar, lo puedes programar".