.UBAfiuba
FACULTAD DE INGENIERÍA

Ingeniería de Sodftware I
(TA046- 7509- 9520) Curso Montaldo

Trabajo Práctico grupal
Proyecto Gestión y Reserva de Fútbol 5

24 de junio de 2025

Joaquin Chamo	Andrea Figueroa	Leticia Figueroa	Federico Honda
110748	110450	110510	110766

Candela Piccin	Santiago Varga	Siyu Zheng	Juan Wienberg
109760	110561	111273	110753

1. Introducción

Como parte de la cursada de la materia Ingeniería de software el día jueves 22 de Mayo se nos fue encargado como equipo de desarrollo la realización de un MVP de un *sistema de gestión y reservas de fútbol 5*, mediante el cual pudiésemos demostrar lo aprendido en los temas que la materia imparte, particularmente siendo escencial el dominio de la metodología de desarrollo SCRUM, concicimiento de la arquitectura API REST, buenas prácticas, entre otros.

En el presente informe detallaremos los avances obtenidos a lo largo de los diferentes Sprints y como estos permitieron entregas de valor constante así como la oportunidad de acceder a feeback temprano, potenciando nuestro trabajo a través de las metodologías ágiles que caracterizan a SCRUM. Además, expondremos los problemas y complicaciones que encontramos no solo técnicamente, sino a nivel organizativo, de esta forma, pretendamos se entienda cómo el equipo afrontó y superó estos mismos en pos del cumplimiento de los objetivos requeridos.

Finalemnte sacando de foco el desarrollo del proyecto para detallar el producto de desarrollado: Presentaremos los diagramas de la arquitectura del proyecto mediante el modelo de vistas 4+1, incluyendo como agregado ejemplos de código que exponen puntos importantes abarcados en el proyecto (test unitarios, definición de los endpoints en la plataforma de Swagger, etc).

2. Herramientas para el desarrollo

Durante el desarrollo del proyecto nos hicimos de diversas herramientas tanto de comunicación como de planificación y organización en pos de mejorar estos aspectos. Nos parecieron significativamente importantes a la hora de la puesta en común de ideas, la división de tareas y generaron mayor claridad a la hora de encarar cada parte del proyecto.

En primer lugar utilizamos ***Jira***, plataforma recomendada por la cátedra la cual facilita la visualización de los sprints a llevar a cabo. Por un lado tenemos el ***backlog***, donde proporcionamos todas las historias de usuario propuestas. Luego, se les podía dar forma de tarea, puliéndolas y teniendo en cuenta los criterios de aceptación. De esta manera, pudimos estimar, mediante la técnica de poker planning, la complejidad o dificultad de cada tarea, para también ser conscientes de que tan prioritarias eran las mismas.

Pasando a canales de comunicación, nos pareció que lo mejor era tener un canal de ***discord*** aparte para mantenernos informados de los avances de cada participante y también poder realizar consultas internas sobre distintos temas. A su vez, facilitó las reuniones semanales ya sea del grupo de trabajo completo o de los diferentes subgrupos encargados de diferentes partes del proyecto.

Siguiendo con el apartado de la planificación y organización, sentimos que el Jira nos quedó un poco acotado en función del volumen de trabajo y la cantidad de integrantes. Por lo tanto decidimos crear, mediante google sheets, algunas tablas las cuales representaban distintas partes del proyecto y como este avanzaba.

Por un lado creamos una tabla con los endpoints realizados en los cuales se establecieron los responsables de cada uno para poder hacer consultas a los integrantes más actualizados sobre el asunto. También hay una columna con el sprint al cual pertenecen dichos endpoints para una mayor claridad. Luego, consideramos importante realizar una tabla con las diferentes historias de usuario a desarrollar, acompañadas del sprint correspondiente al cual pertenecen, los responsables de las mismas y el estado actual en el que se encuentran a nivel de implementación. Como detalle a aclarar, estas herramientas fueron de gran utilidad para la subdivisión de las tareas en el sector del backend. Es decir, no aparece casi ninguna mención al frontend. Esto sucede ya que al haber solo dos integrantes del equipo en el frontend y seis en el backend, nos pareció más urgente y prioritario para la organización realizar esto solo para esta parte del proyecto.

Por último, para el apartado de las ***retro***, debíamos tener una forma de conservar lo debatido en cada una de las reuniones una vez finalizado cada sprint, para así llegar a las conclusiones que nos ayudarían a seguir avanzando en un trabajo de esta magnitud. Por eso consideramos oportuno la creación de un ***miro***, plataforma la cual nos daba múltiples opciones a la hora de crear una

retrospectiva que ayude al futuro desarrollo de los demás sprints. Con esta herramienta pudimos poner en común en cada ***retro*** lo que nos pareció positivo, negativo o consejos y comentarios para mejorar el siguiente tramo del proyecto.

Una aclaración que nos parece pertinente hacer es referente al *flow* utilizado en la plataforma de GitLab. En este caso, optamos por un ***Monorepo*** con ***GitFlow*** simplificado, ya que nos pareció lo más acertado crear una rama dev para no enviar todos los avances directamente a master y poder tener un mayor control a la hora de ofrecer entregas de valor libres de errores e integraciones incompletas. Es simplificado ya que no tenemos varias releases sino una sola rama de la cual derivan las subramas de cada tarea a realizar.

3. Desarrollo en Sprints

3.1. Sprint 1

Este primer Sprint se caracterizó principalmente por tener algunos problemas de organización en cuanto a tiempos y entregas parciales, falta de comunicación entre los distintos subgrupos en los cuales nos dividimos y la poca claridad a la hora de encarar las tareas propuestas. Al ser la primera vez para todos en encarar un proyecto de este estilo, al comienzo intentamos entender lo pedido y establecer prioridades. De igual forma, este mismo sirvió para una primera división entre backend y frontend, comprendiendo los objetivos de cada parte y decidiendo subdividir cada tarea en una de backend y otra de frontend.

Además, agregamos al Jira las columnas de Testing e Integrando. De esta manera logramos tener una mayor claridad con respecto al estado de cada tarea y se estipuló que únicamente se puedan integrar tareas que ya se encuentren debidamente probadas. Por otro lado, nos permitió tener en claro cuando una tarea estaba completada y solo faltaba su Testing o si ya esta ya estaba probada y solo faltaba su integración al resto del proyecto.

A nivel de implementación concreta, en este primer Sprint se sentaron las primeras bases, tanto en frontend como en backend, para el registro, el inicio de sesión y la visualización de las canchas de forma simplificada.

3.2. Sprint 2

En este segundo Sprint logramos atacar de forma eficiente los problemas surgidos en el primero, mejorando la productividad a partir de una mejor distribución de las tareas y la subdivisión de estas en subtareas atómicas. Otro apartado positivo a resaltar es que propusimos el miércoles (anteultimo dia del sprint) como el dia limite para pasar tareas a integración, de forma tal que quedó el tiempo suficiente para integrar los avances y poder entregar valor a partir de una rama estable con los últimos avances.

Como aspecto negativo podemos mencionar que hubo algunos inconvenientes con la base de datos manejada y cómo manipularla. También surgieron errores en el backend a la hora de la integración con el frontend, siendo eso algo que no debería suceder teniendo en cuenta que cada entidad y característica debía ser debidamente probada antes.

Por otro lado, la comunicación mejoró muchísimo con respecto al sprint anterior, de forma que siempre había alguien para responder algún inconveniente o consulta acerca de algún apartado del proyecto. En el apartado de implementación nos concentramos en arreglar los errores surgidos a partir del primer sprint, terminar de integrar y desarrollar la lógica de las canchas y comenzar las historias de usuario referidas a los equipos.

3.3. Sprint 3

En la tercera semana de desarrollo logramos utilizar de forma óptima las herramientas elegidas para mantener al día a cada uno de los integrantes del equipo y conocer el estado de cada parte del proyecto. Aunque nos encontramos con gran cantidad de criterios de aceptación a cumplir en las diversas historias de usuario, pudimos desarrollarlos manteniendo al tanto a los demás sobre los avances realizados.

Otro apartado a mencionar es que empezamos a definir los endpoints a realizar desde el principio del sprint. Esto mejoró muchísimo tanto la planificación como la integración de cada entidad a implementar, teniendo en cuenta así la comunicación entre backend y frontend. También empezamos a hacer una limpieza en las ramas de GitLab para prepararnos para el último sprint.

Se avanzó y completó la entidad de equipos, se determinaron nuevas entidades como Booking, Time Slot y Tournaments. También se completaron los endpoints que refieren a las invitaciones a otros jugadores.

3.4. Sprint 4

En el último sprint pudimos concretar todas las historias de usuario restantes que eran requeridas para el fin del proyecto. Se trató, más que nada, de concretar, probar, integrar y acomodar las entidades y características establecidas al final del tercer sprint y principios del último.

Un inconveniente que tuvimos fue el hecho de que en la aplicación de Jira nunca iniciamos el sprint 4, por lo tanto seguramente no se llegue a apreciar con claridad una diferencia entre los últimos dos sprints, aunque de igual forma la plataforma fue utilizada y actualizada constantemente.

También se implementaron clases de Testing dentro del backend para las diferentes entidades anteriormente mencionadas, siendo así más fácil detectar errores dentro de las mismas.

4. Arquitectura: Modelo de vistas 4+1 y Atributos de calidad

A continuación expondremos el documento de arquitectura, el cual tiene como objetivo diagramar, de forma simplificada y a modo de ejemplo, las diversas funcionalidades del proyecto según el modelo de vistas 4+1.

4.1. Vista de escenarios

Este es un modelo de algunos posibles casos de uso para los usuarios de esta plataforma, diferenciados según su rol.

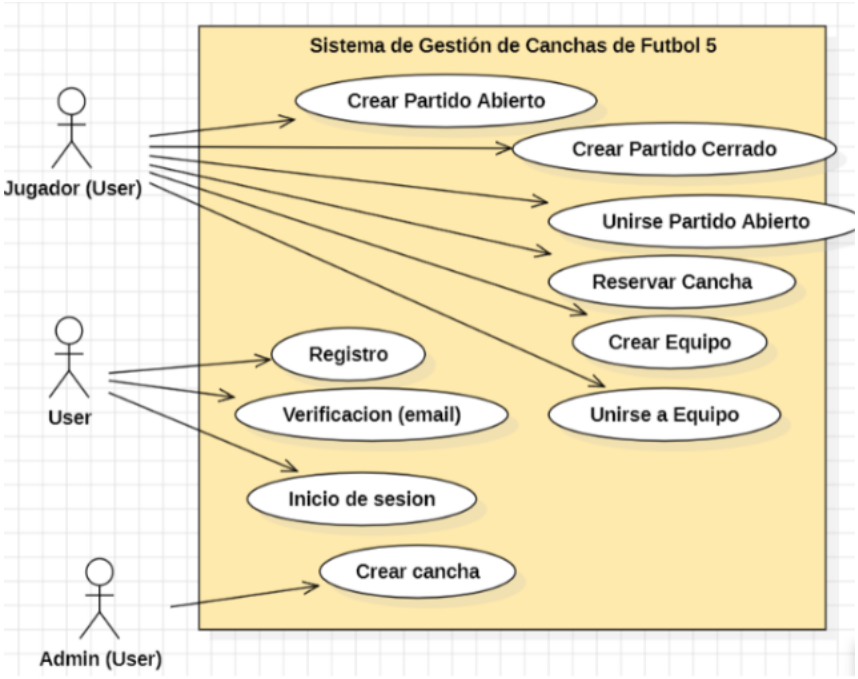


Figura 1: Obs: Para el caso del Administrador existen varias funcionalidades importantes referentes a las canchas, como puede ser eliminarlas o editar sus valores por defecto, pero en este caso se encuentra simplificado dentro de el caso de uso CREAR.

4.1.1. Atributos de calidad

- **Seguridad**: La seguridad es un atributo central en el sistema, con especial foco en la autenticación y autorización de los usuarios. Se implementan mecanismos para garantizar que solo usuarios legítimos puedan acceder a funcionalidades específicas. Por ejemplo, solo el capitán de un equipo (es decir, el usuario que lo creó) tiene permisos para invitar a nuevos miembros, evitando accesos indebidos o manipulaciones no autorizadas.
    - *Cómo se puede evaluar*: Se puede pedir a un grupo de personas que intente realizar acciones que no les corresponden (por ejemplo, que alguien que no sea capitán intente invitar miembros) y verificar si el sistema lo impide correctamente. También se puede revisar si los usuarios sienten que su información está segura mediante una breve encuesta.
- **Usabilidad**: El sistema está diseñado para ser intuitivo y fácil de usar, tanto para jugadores como para dueños de cancha que publican la disponibilidad de sus instalaciones y desean

poder acceder y entender fácilmente la información de las reservas. La interfaz debe guiar a los usuarios según su rol, mostrando solo las acciones que pueden realizar.

- *Cómo se puede evaluar*: Se pueden hacer encuestas simples a los usuarios para preguntar si entienden cómo usar el sistema y si les resulta fácil de navegar. También se puede observar cuánto tiempo les toma a los usuarios completar tareas básicas (como reservar una cancha o unirse a un equipo) y si necesitan ayuda o cometen errores durante el proceso.

4.2. Vista Lógica

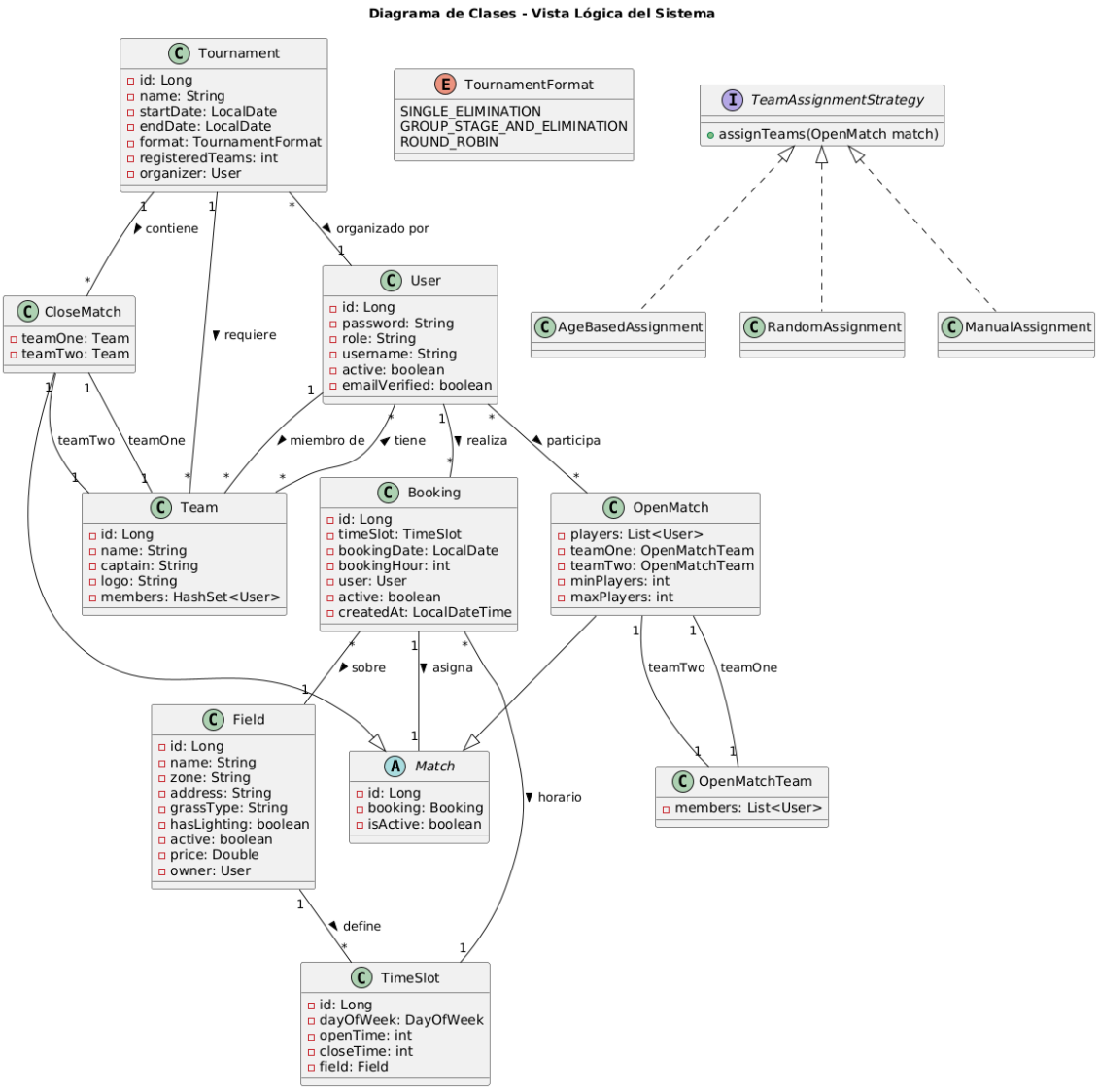


Figura 2: Diagrama de clases

4.2.1. Módulos principales

User

Representa a los usuarios registrados.
Tipos: `USER`, `ADMIN`.
Operaciones:

- Registro y login (JWT).
- Verificación de email.
- Obtener perfil.

Match

Gestiona partidos de fútbol.
Subtipos:

- `OpenMatch`: creado por un usuario, jugadores individuales se pueden unir.
- `CloseMatch`: requiere dos equipos ya conformados.

Operaciones:

- Crear, listar, obtener partido.
- Asignación de equipos por estrategia (manual, aleatoria, por edad).

Field

Representa una cancha.
Administrada por usuarios `ADMIN`.
Características:

- Nombre, dirección, zona, tipo de césped, iluminación, fotos.

Operaciones:

- Crear, listar, editar, activar/desactivar.

Booking

Representa una reserva de cancha para cierto horario y día.
Asociada a un `User` y un `Field`.
Operaciones:

- Crear reserva.
- Validar disponibilidad.

Team

Equipos creados por usuarios `USER`.
Un usuario puede ser miembro de uno o varios equipos.
Operaciones:

- Crear equipo.
- Agregar/Eliminar miembros.

Tournament

Organización de torneos que agrupan partidos cerrados.
Planificado para manejar clasificaciones, fechas, resultados.

Email

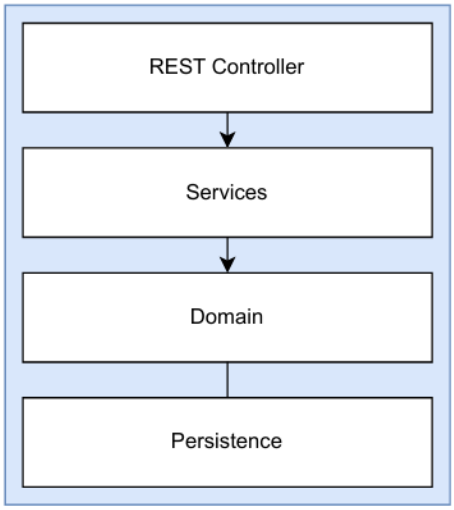
Servicio transversal para notificaciones.
Operaciones:

- Confirmación de cuenta.
- Notificación de partidos asignados.
- Recordatorios de reservas.

Timeslot

Representa los horarios posibles de reserva en cada cancha.
Ayuda a validar disponibilidad.
Podría incluir restricciones de horario por zona/temporada.

4.2.2. Arquitectura Interna por Capas

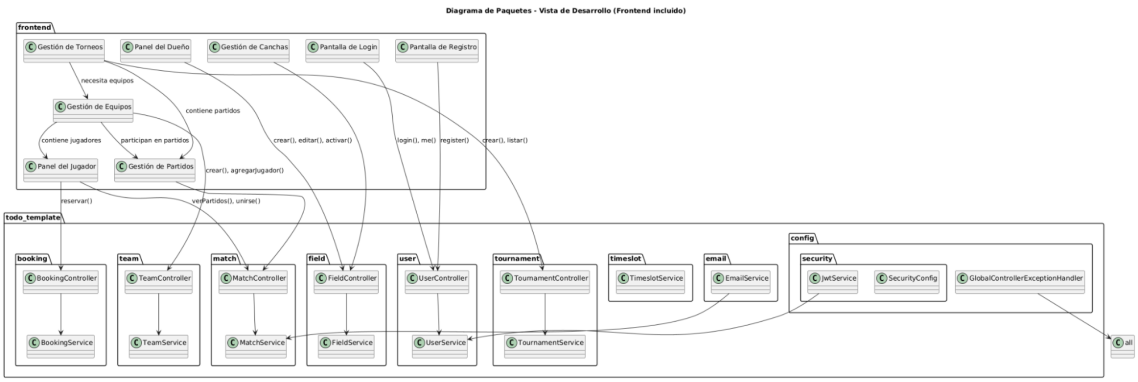


4.2.3. Relaciones Clave entre Entidades

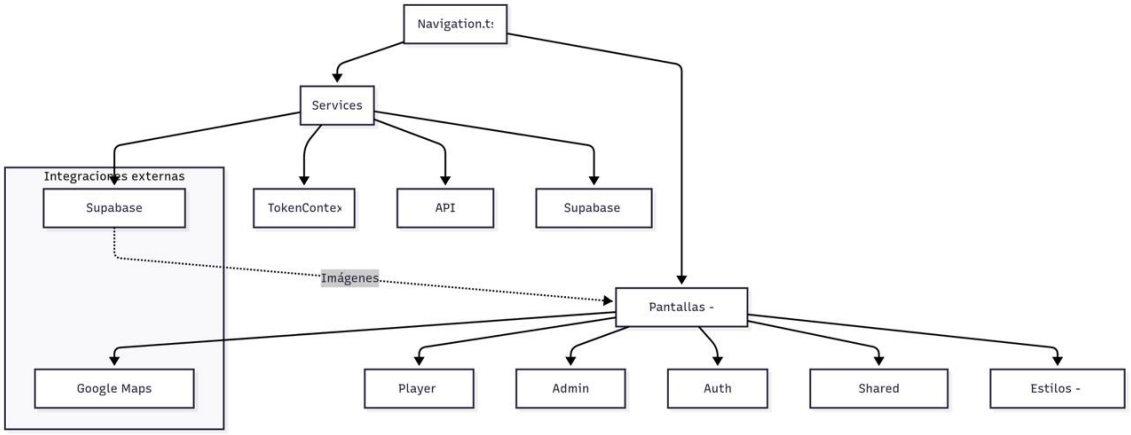
- **User** puede:
  - Crear `Booking`, `Team`, `OpenMatch`.
  - Unirse a `OpenMatch`.
  - Formar parte de un `Team`.
- **Admin (User)** puede:
  - Crear y administrar `Field`.
- **Booking** relaciona:
  - `User`, `Field` y un `Match` (relación uno a uno).
- **Match** puede ser:
  - `OpenMatch`: lista de `User`.
  - `CloseMatch`: dos `Team`.
- **Match** utiliza:
  - `Booking` (reserva del turno).
  - `TeamAssignmentStrategy` (para asignación en `OpenMatch`).
- **EmailService** es llamado desde:
  - `MatchService`, `UserService` o `TeamsService` según el caso.

4.3. Vista de Desarrollo

4.3.1. Diagrama de paquetes

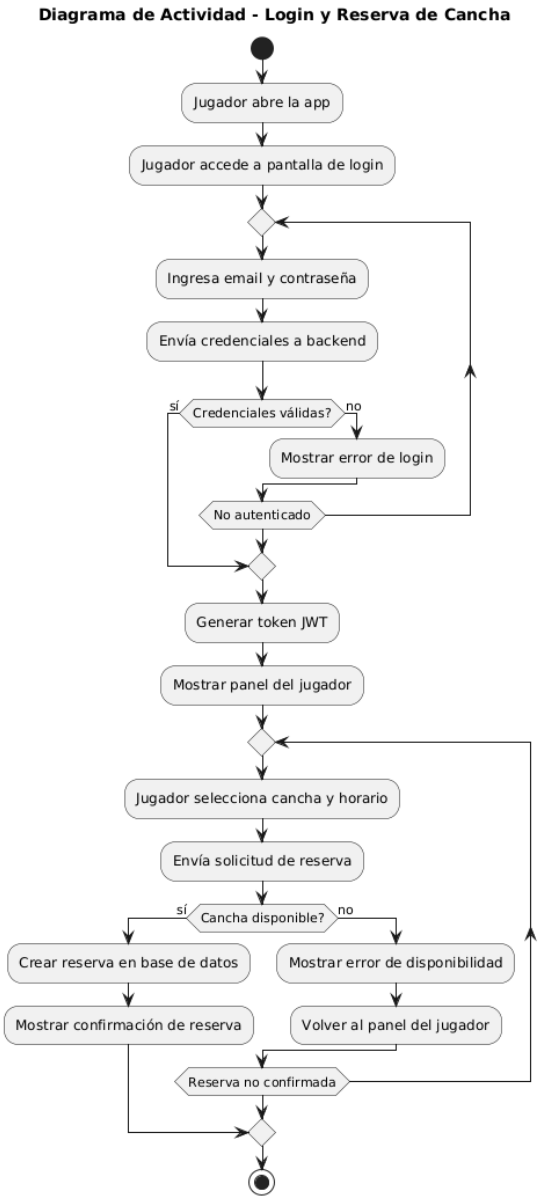


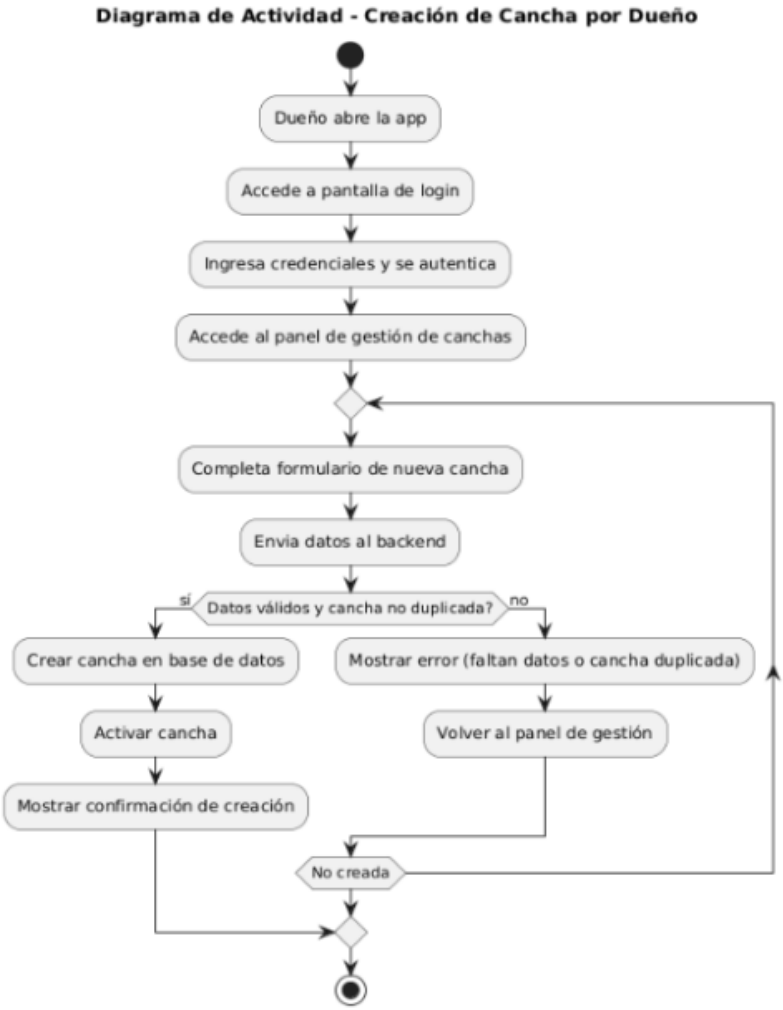
4.3.2. Estructura modular: frontend



4.4. Vista de Proceso

4.4.1. Diagrama de Actividad

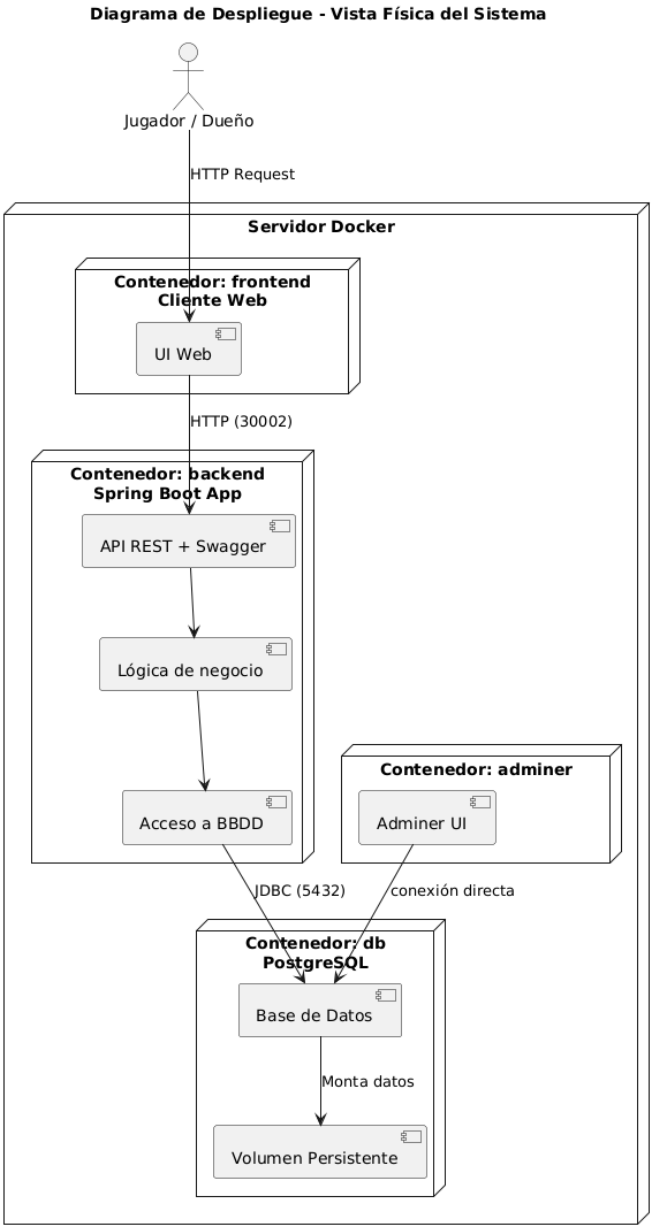




Nota aclaratoria sobre la vista de procesos: El diagrama presentado en este apartado no refleja el manejo detallado de la concurrencia, sino que representa la estructura general de un único proceso del sistema. La gestión de la concurrencia en el servidor no fue implementada manualmente, sino que está delegada al framework Spring Boot, el cual abstrae gran parte de esta complejidad. En términos generales, Spring Boot maneja la concurrencia utilizando un modelo basado en hilos (threads), apoyándose en componentes como el servidor embebido (por ejemplo, Tomcat), que crea un pool de hilos para atender múltiples solicitudes simultáneamente. Esto permite que el servidor procese múltiples peticiones en paralelo, garantizando la escalabilidad sin necesidad de que los desarrolladores implementen explícitamente la sincronización o el control de hilos en el nivel de la aplicación.

4.5. Vista Física

4.5.1. Diagrama de Despliegue



5. Ejemplos de código

A continuación mostraremos algunos ejemplos en código de lo que nos parecieron buenas prácticas dentro de nuestro proyecto:

Patrón Strategy para asignación de partido

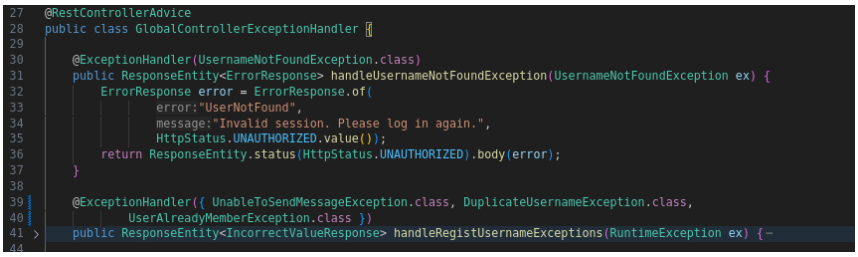
```java
package ar.uba.fi.ingsoft1.todo_template.match.strategy;

import ar.uba.fi.ingsoft1.todo_template.match.OpenMatch;

public interface TeamAssignmentStrategy {
    void assignTeams(OpenMatch match);
}
```

```java
public class AgeBasedAssignment implements TeamAssignmentStrategy {
    @Override
    public void assignTeams(OpenMatch match) {
        List<User> players = new ArrayList<>(match.getPlayers());
        match.getTeamOne().clearMembers();
        match.getTeamTwo().clearMembers();

        players.sort(Comparator.comparing(User::getBirthYear));

        for (int i = 0; i < players.size(); i++) {…
    }
}
```

```java
public class ManualAssignment implements TeamAssignmentStrategy {
    private final Map<Long, Integer> assignments; // userId -> teamNumber (1 or 2)

    public ManualAssignment(Map<Long, Integer> assignments) {
        this.assignments = assignments;
    }

    @Override
    public void assignTeams(OpenMatch match) {

        match.getTeamOne().clearMembers();
        match.getTeamTwo().clearMembers();
        for (User player : match.getPlayers()) {…
    }
}
```

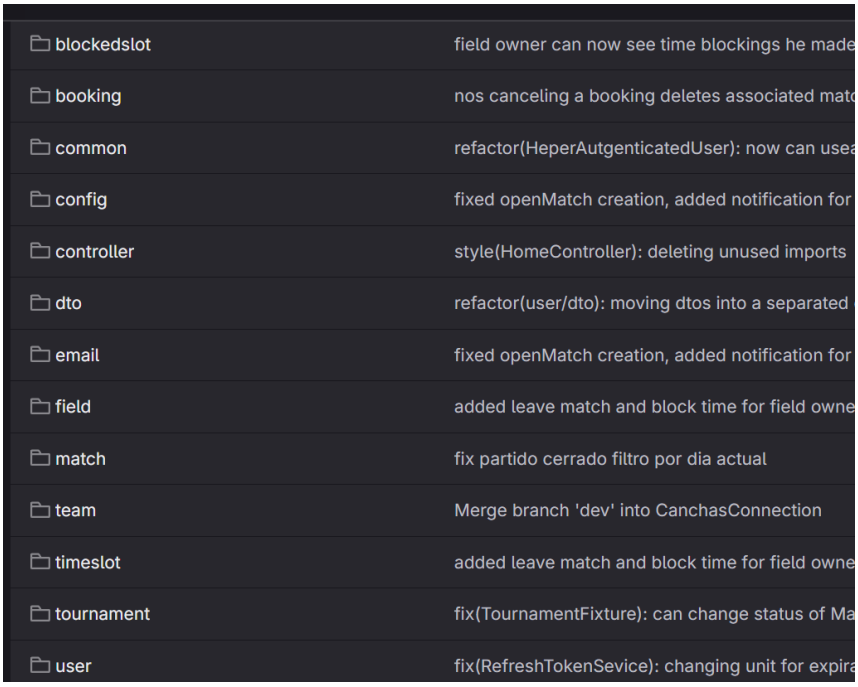
```java
public class RandomAssignment implements TeamAssignmentStrategy {
    private final Random random = new Random();

    @Override
    public void assignTeams(OpenMatch match) {
        List<User> players = new ArrayList<>(match.getPlayers());
        Collections.shuffle(players, random);
        match.getTeamOne().clearMembers();
        match.getTeamTwo().clearMembers();

        for (int i = 0; i < players.size(); i++) {…
    }
}
```

Controlador de excepciones



```
@RestControllerAdvice
public class GlobalControllerExceptionHandler {

    @ExceptionHandler(UsernameNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleUsernameNotFoundException(UsernameNotFoundException ex) {
        ErrorResponse error = ErrorResponse.of(
                error:"UserNotFound",
                message:"Invalid session. Please log in again.",
                HttpStatus.UNAUTHORIZED.value());
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(error);
    }

    @ExceptionHandler({ UnableToSendMessageException.class, DuplicateUsernameException.class,
            UserAlreadyMemberException.class })
    public ResponseEntity<IncorrectValueResponse> handleRegistUsernameExceptions(RuntimeException ex) {
```

Separación de incumbencias en módulos



blockedslot	field owner can now see time blockings he made
booking	nos canceling a booking deletes associated mat
common	refactor(HeperAutgenticatedUser): now can usea
config	fixed openMatch creation, added notification for
controller	style(HomeController): deleting unused imports
dto	refactor(user/dto): moving dtos into a separated
email	fixed openMatch creation, added notification for
field	added leave match and block time for field owne
match	fix partido cerrado filtro por dia actual
team	Merge branch 'dev' into CanchasConnection
timeslot	added leave match and block time for field owne
tournament	fix(TournamentFixture): can change status of Ma
user	fix(RefreshTokenSevice): changing unit for expira

6. Conclusiones

A partir de lo expuesto en este informe y lo realizado a lo largo de los sprints correspondientes, pudimos llegar a las siguientes conclusiones:

En primer lugar, creemos que el reparto de tareas entre *frontend* y *backend* permitió un desarrollo paralelo y eficiente de las tareas que requirieron mayor atención y organización por nuestra parte.

Desde el punto de vista técnico, el *backend* se implementó utilizando **Spring Boot**, con persistencia en **PostgreSQL** y despliegue en contenedores **Docker**, lo cual garantizó un entorno de desarrollo homogéneo y replicable. Se adoptaron buenas prácticas como:

- Autenticación y autorización mediante **JWT**.
- Validaciones de negocio sólidas (por ejemplo, en reservas y creación de canchas).
- Uso del patrón **Strategy** para asignación flexible de equipos en partidos abiertos.
- Manejo de errores de manera eficiente.
- Diseño limpio por capas (`Controller` → `Service` → `Repository` → `Entity`).
- Documentación automática con **Swagger** y pruebas unitarias con **JUnit/Mockito**.

Por su parte, el *frontend* integró las vistas necesarias para los dos tipos de usuarios del sistema: jugadores y dueños. Se logró una experiencia fluida que permite registrarse, loguearse, crear equipos, unirse a partidos o administrar canchas según el rol.

Además, se respetó una arquitectura lógica coherente y se modelaron claramente los diferentes aspectos del dominio: partidos abiertos y cerrados, reservas, canchas, equipos, torneos y slots horarios. El uso de los modelos en las vistas **4+1** aprendidos durante la cursada permitió visualizar y comunicar con claridad el funcionamiento del sistema.

En términos organizativos, el trabajo colaborativo permitió dividir tareas, hacer *code reviews*, integrar avances frecuentes y resolver conflictos a tiempo. Creemos que se cumplieron los objetivos funcionales propuestos y se logró un producto robusto, escalable y fácil de mantener.

7. Links útiles

Como apartadop final incluimos los enlaces a las herramientas auxiliares mencionadas y usadas a lo largo del desarrollo del proyecto.

- miro: pizarra para visualización en retros
- División de historias de usuario y endpoints en excel- Backend
- División de historias de usuario- Frontend
- jira- backlog
- Repositorio en GitLab