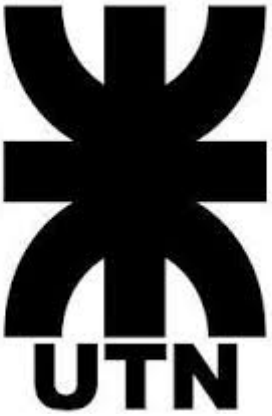
Lenguaje de Programación JAVA

Trabajo Práctico

"TeamUp"

Documentación



Ingeniería en Sistemas de Información

Comisión: 3EK02

Integrantes:

- Nombre: Berli, Nahuel - Legajo: 50310
- Nombre: Giampietro, Gustavo - Legajo: 50671
- Nombre: Jaca, Juan Pablo - Legajo: 50311

Profesores:

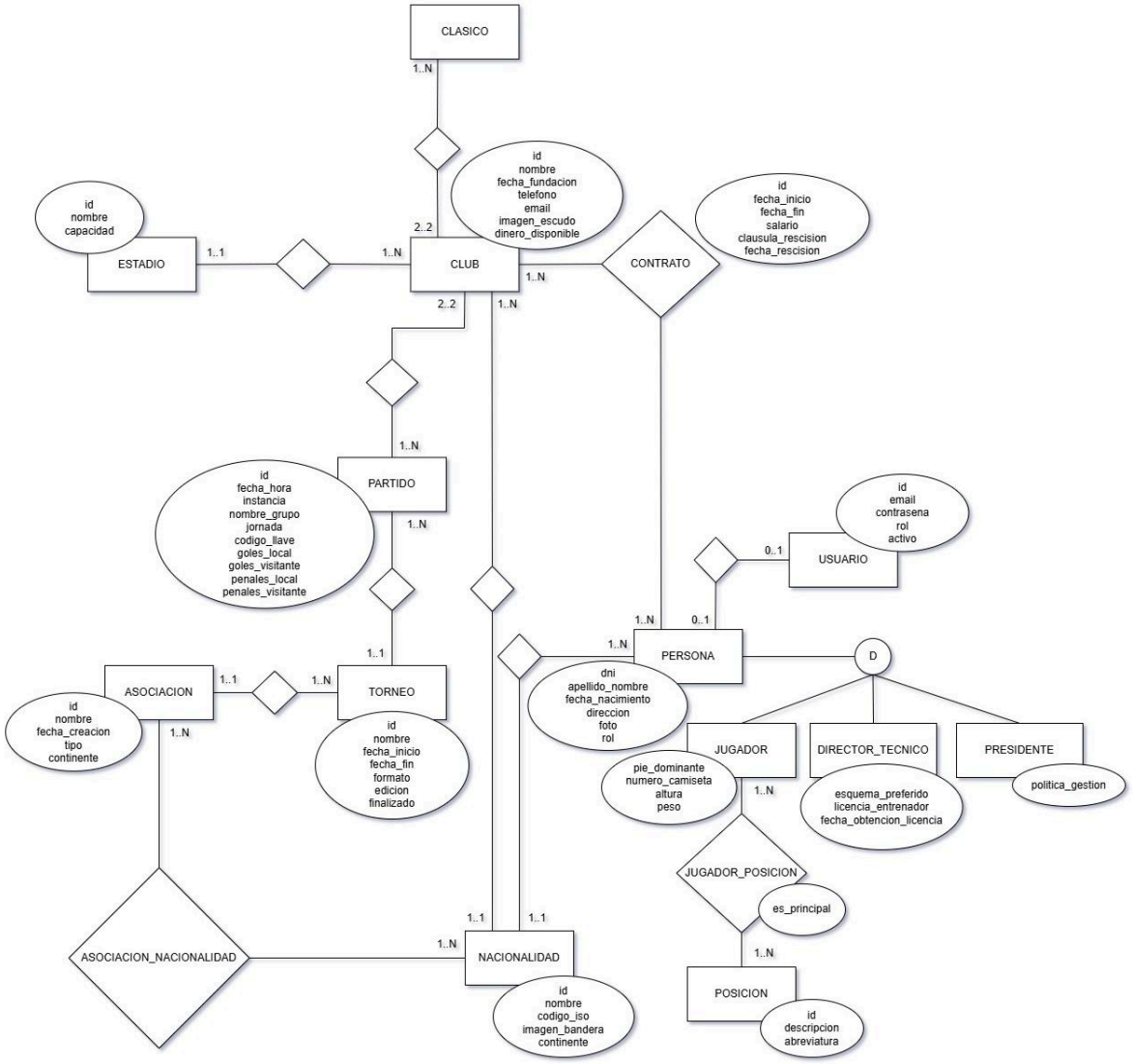
- Meca, Adrián
- Tabacman, Ricardo

Enunciado General

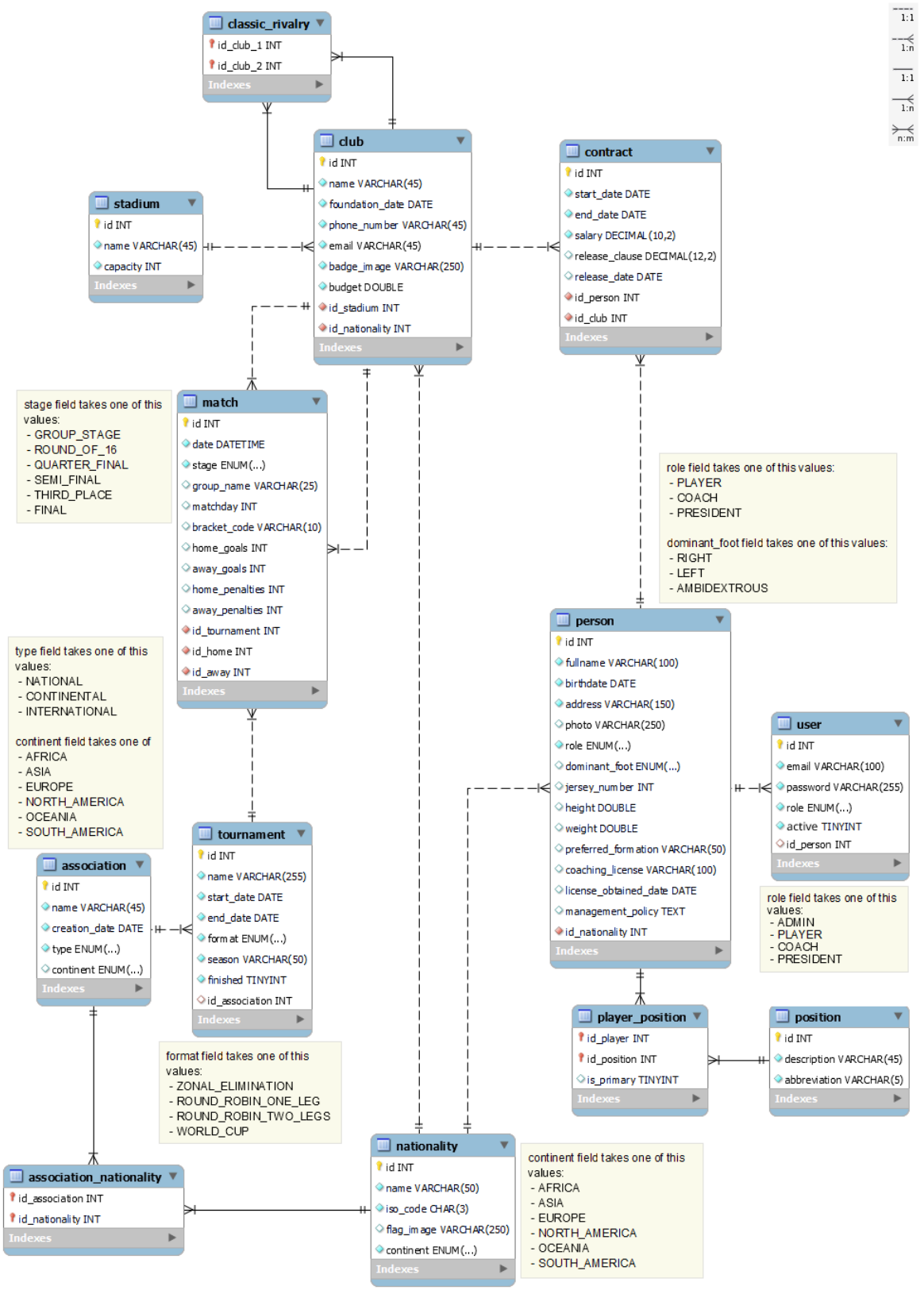
El sistema a desarrollar consiste en una plataforma para la gestión de clubes de fútbol, enfocada en el seguimiento de torneos, jugadores y directores técnicos. Esta aplicación permitirá administrar la información de entidades deportivas, incluyendo la gestión de clubes con sus respectivos estadios, el registro de personas vinculadas a dichos clubes (jugadores, directores técnicos y presidentes), así como los contratos que los vinculan a las instituciones y las participaciones en torneos organizados por las distintas asociaciones.

La plataforma facilitará el seguimiento completo de la actividad deportiva, permitiendo registrar partidos con sus correspondientes resultados, gestionar jornadas de torneos, y mantener estadísticas actualizadas de cada club y sus participantes. El sistema implementará distintos niveles de acceso según el rol del usuario (administrador, presidente, jugador, director técnico o invitado), garantizando así la seguridad y privacidad de la información, además de incluir funcionalidades adicionales como el envío de correos electrónicos para notificaciones importantes.

Modelo de Dominio



Modelo de Datos



Alcance funcional

Regularidad

Req	Detalle
CRUD Simples	1. Persona 2. Estadio 3. Asociación
CRUD Dependientes	1. Club 2. Torneo
Listados	1. Listado de Jugadores con sus respectivos Clubes actuales (permitiendo filtrar por Club)
CUU/Epic	1. Contrato de Jugadores y Director Técnico por parte de un Club 2. Rescisión de Contrato por parte del Jugador o Director Técnico

Adicionales para Aprobación

Req	Detalle
CRUD	1. Persona 2. Estadio 3. Asociación 4. Club 5. Torneo 6. Nacionalidad 7. Posición 8. Posiciones de Jugadores
Listados	1. Listado de clubes por los que pasó un jugador en su carrera deportiva (histórico), permitiendo filtrar por un rango de fechas 2. Listado de partidos disputados por un club junto a sus resultados, permitiendo filtrar por torneo
CUU/Epic	1. Armado de jornadas de torneos, junto a los partidos que se van a disputar con su fecha y hora, y la posterior asignación de los equipos que se enfrentarán en cada uno de ellos 2. Gestión completa de partidos de un torneo, incluyendo registro de resultados, puntos, y actualización automática de estadísticas de participación (partidos ganados, empatados, perdidos, goles a favor y en contra)
Niveles de Acceso	1. Administrador 2. Presidente 3. Jugador 4. Director Técnico 5. Invitado (no necesita loguearse)
Requerimientos Extra	1. Manejo de archivos (imágenes) 2. Envío de emails

Casos de Uso

CURS_1 - Gestión del Ciclo de Vida de Torneos

Dimensiones de clasificación:

Nivel	Estructura	Alcance	Caja	Instanciación	Interacción
Resumen	Re-Estructurado	Sistema	Negra	Real	Semántica

Meta del CASO DE USO:
Gestionar un torneo desde su planificación hasta el registro del resultado de sus partidos y la visualización de su tabla de posiciones

Actor Primario: Administrador (ADM) **Otros:** Usuario/Invitado (U/I)

PRECONDICIONES (de sistema):
- ADM está logueado en el sistema.
- (Usuario está logueado en el sistema).
- Asociaciones junto a sus clubes ya registradas.

DISPARADOR:
El Administrador debe organizar una nueva competencia deportiva y requiere realizar el seguimiento integral de la misma, desde su planificación inicial hasta la determinación de un ganador y el cierre oficial del evento.

CAMINO BÁSICO:

1. ADM genera el fixture del torneo invocando al **CUU_1.1 Generar Fixture de Torneos.**
2. ADM ingresa resultado de un partido invocando al **CUU_1.2 Registrar Resultado de Partido.**
   Se repite el paso 2 las veces necesarias.
3. En algún momento posterior, U/I requiere ver la tabla de posiciones del torneo invocando al **CUU_1.3 Consultar Tabla de Posiciones.**
4. ADM genera la fase de eliminación invocando al **CUU_1.4 Generar Fase de Eliminación.**
5. ADM ingresa resultado de un partido de fase de eliminación invocando al **CUU_1.2 Registrar Resultado de Partido.**
   Se repite el paso 5 las veces necesarias.

Se repiten los pasos 4 y 5 por cada fase eliminatoria del torneo.

6. ADM finaliza el torneo invocando al **CUU_1.5 Finalizar Torneo.**

CAMINOS ALTERNATIVOS:

2.a. ADM elimina el torneo invocando al **CUU_1.6 Eliminar Torneo.** Fin CU.

4.a. El formato del torneo no tiene fase de eliminación. Continúa en el paso 6.

POSTCONDICIONES (de sistema):

- Éxito: El torneo queda registrado como finalizado.
- Éxito Alternativo: —
- Fracaso: El torneo es eliminado.

POSTCONDICIONES (de negocio):

- Éxito: El torneo ha cumplido todo su ciclo de vida, y se cuenta con un resultado oficial y definitivo del mismo.
- Éxito Alternativo: —
- Fracaso: El torneo ha sido eliminado, anulándose de esta forma la competencia.

CUU_1.1 Generar Fixture de Torneos

Dimensiones de clasificación:

Nivel	Estructura	Alcance	Caja	Instanciación	Interacción
Usuario	Sin estructurar	Sistema	Negra	Real	Semántica

Meta del CASO DE USO:
Crear un nuevo torneo y generar automáticamente el calendario de partidos, incluyendo la fecha, hora y equipos que se enfrentan en cada uno

Actor Primario: Administrador (ADM) **Otros:** —

PRECONDICIONES (de sistema):
- ADM está logueado en el sistema.
- La asociación organizadora está registrada.
- Los clubes que participarán se encuentran registrados.

DISPARADOR:
ADM requiere iniciar una nueva edición de una competencia, a pedido de la asociación organizadora

CAMINO BÁSICO:

1. ADM ingresa los datos generales: Nombre, Fecha de Inicio aproximada, Formato, Edición y Asociación. Sistema muestra los clubes registrados para la asociación ingresada.
2. ADM selecciona los clubes participantes de la lista provista por el sistema. Sistema informa la cantidad de clubes seleccionados.
3. ADM confirma la operación presionando el botón “Agregar”. Sistema valida los datos, genera automáticamente el fixture con sus jornadas (partidos con fecha, hora y equipos que se enfrentan) y redirige al listado de torneos.

CAMINOS ALTERNATIVOS:

2.a. ADM marca la opción “Seleccionar todos”. Sistema selecciona a todos los clubes de la Asociación ingresada e informa la cantidad de clubes seleccionados.

3.a. ADM presiona el botón “Cancelar”. Fin CU.

POSTCONDICIONES (de sistema):

- Éxito: El torneo queda registrado y los partidos son visibles en la sección correspondiente.
- Éxito Alternativo: —
- Fracaso: —

POSTCONDICIONES (de negocio):

- Éxito: Se oficializó la competencia, garantizando la difusión del cronograma para la organización logística de los clubes y participantes.
- Éxito Alternativo: —
- Fracaso: —

CUU_1.2 Registrar Resultado de Partido

Dimensiones de clasificación:

Nivel	Estructura	Alcance	Caja	Instanciación	Interacción
Usuario	Sin estructurar	Sistema	Negra	Real	Semántica

Meta del CASO DE USO:
Registrar el resultado de un partido y actualizar la tabla de posiciones

Actor Primario: Administrador (ADM) **Otros:** —

PRECONDICIONES (de sistema):
- ADM está logueado en el sistema.
- El torneo, junto a sus jornadas de partidos, está registrado.

DISPARADOR:
Ha finalizado el partido y ADM requiere registrar su resultado

CAMINO BÁSICO:

1. ADM presiona el botón “Ver Fixture” del torneo en cuestión. Sistema emite un listado de todos los partidos del torneo.
2. ADM presiona el botón "Registrar Resultado" del partido cuyo resultado quiere cargar. Sistema habilita los campos para el ingreso de goles del equipo local y visitante.
3. ADM ingresa la cantidad de goles para el equipo local y el equipo visitante.
4. ADM presiona el botón “Confirmar”. Sistema registra el resultado del partido y vuelve a emitir el listado de los partidos del torneo, con el resultado actualizado. (Sistema actualiza la tabla de posiciones)

CAMINOS ALTERNATIVOS:

2.a. No hay ningún partido sin resultado cuya fecha sea anterior a la actual. Sistema deshabilita los botones “Registrar Resultado”. Fin CU.

3.a. El partido es de fase de eliminación y se definió por penales.
3.a.1 ADM ingresa la cantidad de goles para el equipo local y el equipo visitante. Sistema habilita los campos para el ingreso del resultado de la definición por penales.

3.a.2. ADM ingresa la cantidad de penales convertidos del equipo local y del equipo visitante. Continua con el paso 4.

4.a. ADM presiona el botón “Cancelar”. Fin CU.

POSTCONDICIONES (de sistema):

- Éxito: Se registró el resultado del partido.
- Éxito Alternativo: Se registró el resultado del partido y el de su definición por penales.
- Fracaso: —

POSTCONDICIONES (de negocio):

- Éxito: Se asentó el marcador para validar el desempeño de los equipos y permitir la actualización de la tabla de posiciones.
- Éxito Alternativo: Se asentó el marcador, junto a la definición por penales, para validar el desempeño de los equipos y permitir saber quien avanzó de fase.
- Fracaso: —

CUU_1.3 Consultar Tabla de Posiciones

Dimensiones de clasificación:

Nivel	Estructura	Alcance	Caja	Instanciación	Interacción
Usuario	Sin estructurar	Sistema	Negra	Real	Semántica

Meta del CASO DE USO:
Permitir que el actor obtenga una visión detallada y actualizada del rendimiento y la clasificación de los equipos en el torneo, facilitando la comparación estadística entre ellos.

Actor Primario: Usuario/Invitado (U/I) **Otros:** —

PRECONDICIONES (de sistema):
- (Usuario está logueado en el sistema).
- El torneo, junto a sus jornadas de partidos, está registrado.

DISPARADOR:
U/I quiere visualizar la tabla de posiciones del torneo.

CAMINO BÁSICO:

1. U/I presiona el botón “Ver Tabla de Posiciones” del torneo en cuestión. Sistema muestra la tabla de posiciones junto con los puntos, partidos jugados, partidos ganados, empatados y perdidos, goles a favor y en contra y la diferencia de goles de cada equipo ordenados de forma descendente por cantidad de puntos, diferencia de goles (en caso de que dos o más equipos tengan los mismos puntos) y goles a favor (en caso que dos o más equipos tengan los mismos puntos y misma diferencia de goles).

CAMINOS ALTERNATIVOS:

1.a.1. El formato del torneo tiene distintas zonas o grupos. Sistema muestra la tabla de posiciones de la primera zona o grupo.

1.a.2. U/I selecciona la opción para visualizar la tabla de posiciones de otra zona o grupo. Sistema muestra la tabla de posiciones de la zona o grupo seleccionado.

POSTCONDICIONES (de sistema):

- Éxito: El sistema ha recuperado y mostrado la información estadística de los equipos de acuerdo a los criterios de ordenamiento.

- Éxito Alternativo: El sistema ha recuperado y mostrado la información estadística de los equipos de distintas zonas o grupos de acuerdo a los criterios de ordenamiento.

- Fracaso: —

POSTCONDICIONES (de negocio):

- Éxito: Se ha garantizado la visibilidad del rendimiento deportivo y el orden de mérito de los equipos.

- Éxito Alternativo: Se ha garantizado la visibilidad del rendimiento deportivo y el orden de mérito de los equipos de la competencia en todas sus zonas o grupos.

- Fracaso: —

CUU_1.4 Generar Fase de Eliminación

Dimensiones de clasificación:

Nivel	Estructura	Alcance	Caja	Instanciación	Interacción
Usuario	Sin estructurar	Sistema	Negra	Real	Semántica

Meta del CASO DE USO:
Formalizar el cierre de la etapa clasificatoria y automatizar la creación del fixture de eliminación directa, asegurando que los cruces se correspondan con los resultados obtenidos previamente en la fase de grupos.

Actor Primario: Administrador (ADM) **Otros:** —

PRECONDICIONES (de sistema):
- ADM está logueado en el sistema.
- El torneo, junto a sus jornadas de partidos, está registrado.
- Todos los partidos de la fase de grupos tienen resultado.

DISPARADOR:
ADM requiere iniciar la fase de eliminación de la competencia tras la culminación del último partido programado de la etapa clasificatoria.

CAMINO BÁSICO:

1. ADM presiona el botón “Generar Fase de Eliminación” del torneo en cuestión. Sistema muestra mensaje indicando que se utilizará la tabla de posiciones actual para generar los partidos de la primera instancia eliminatoria.
2. ADM presiona el botón “Confirmar Generación”. Sistema genera automáticamente el fixture de la fase de eliminación (partidos con fecha, hora y equipos que se enfrentan) e inhabilita el botón “Registrar Resultado” para los partidos de las fases ya finalizadas.

CAMINOS ALTERNATIVOS:

1.a. El torneo ya ha finalizado. Sistema muestra el botón “Finalizar Torneo” inhabilitado en lugar del botón “Generar Fase de Eliminación”.

2.a. ADM presiona el botón “Cancelar”. Fin CU.

POSTCONDICIONES (de sistema):

- Éxito: Se registró una nueva fase de eliminación incluyendo la programación de sus partidos, junto con la fecha y hora de los mismos y los equipos que se enfrentan.
- Éxito Alternativo: —
- Fracaso: —

POSTCONDICIONES (de negocio):

- Éxito: Se ha habilitado la etapa definitoria del torneo y coordinado la logística de los cruces directos para garantizar la continuidad de la competencia.
- Éxito Alternativo: —
- Fracaso: —

CUU_1.5 Finalizar Torneo

Dimensiones de clasificación:

Nivel	Estructura	Alcance	Caja	Instanciación	Interacción
Usuario	Sin estructurar	Sistema	Negra	Real	Semántica

Meta del CASO DE USO:
Realizar el cierre administrativo y técnico de la competencia, consolidando los resultados finales en el historial y garantizando que la información no sea alterada posteriormente.

Actor Primario: Administrador (ADM) **Otros:** —

PRECONDICIONES (de sistema):
- ADM está logueado en el sistema.
- El torneo, junto a sus jornadas de partidos, está registrado.
- Todos los partidos, incluídos los de fase de eliminación si hubiera, tienen resultado.

DISPARADOR:
El Administrador decide finalizar el torneo tras la culminación del último partido programado.

CAMINO BÁSICO:

1. ADM presiona el botón “Finalizar Torneo” del torneo en cuestión. Sistema muestra mensaje indicando que el torneo pasará al historial de finalizados y se bloqueará la edición de resultados.
2. ADM presiona el botón “Finalizar Torneo”. Sistema registra el torneo como finalizado.

CAMINOS ALTERNATIVOS:

2.a. ADM presiona el botón “Cancelar”. Fin CU.

POSTCONDICIONES (de sistema):

- Éxito: El torneo queda registrado como finalizado.
- Éxito Alternativo: —
- Fracaso: —

POSTCONDICIONES (de negocio):

- Éxito: El torneo ha cumplido todo su ciclo de vida, y se cuenta con un resultado oficial y definitivo del mismo.
- Éxito Alternativo: —
- Fracaso: —

CUU_1.6 Eliminar Torneo

Dimensiones de clasificación:

Nivel	Estructura	Alcance	Caja	Instanciación	Interacción
Usuario	Sin estructurar	Sistema	Negra	Real	Semántica

Meta del CASO DE USO:
Eliminar de forma definitiva un torneo y toda su planificación asociada del sistema.

Actor Primario: Administrador (ADM) **Otros:** —

PRECONDICIONES (de sistema):
- ADM está logueado en el sistema.
- El torneo, junto a sus jornadas de partidos, está registrado.
- Ningún partido tiene resultado.

DISPARADOR:
El Administrador decide eliminar el torneo debido a un error en el armado del fixture o a pedido de la asociación organizadora.

CAMINO BÁSICO:

1. ADM presiona el botón “Eliminar Torneo” del torneo en cuestión. Sistema muestra mensaje indicando que la acción es irreversible y se borrarán permanentemente todos sus partidos programados.
2. ADM presiona el botón “Eliminar”. Sistema elimina el torneo junto con sus partidos programados.

CAMINOS ALTERNATIVOS:

2.a. ADM presiona el botón “Cancelar”. Fin CU.

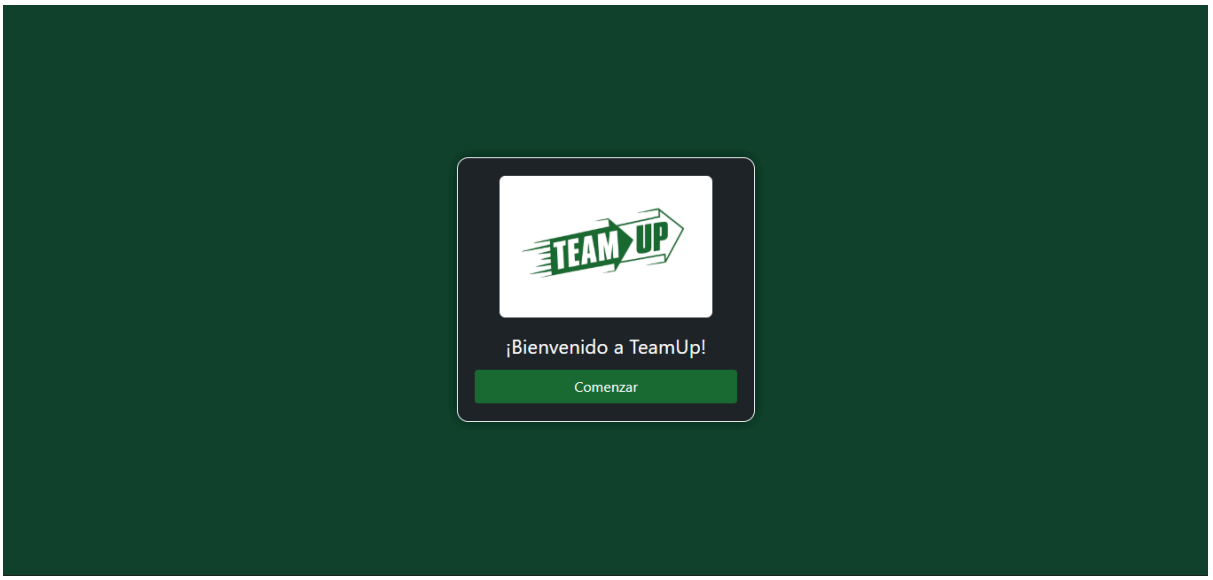
POSTCONDICIONES (de sistema):

- Éxito: El torneo es eliminado.
- Éxito Alternativo: —
- Fracaso: —

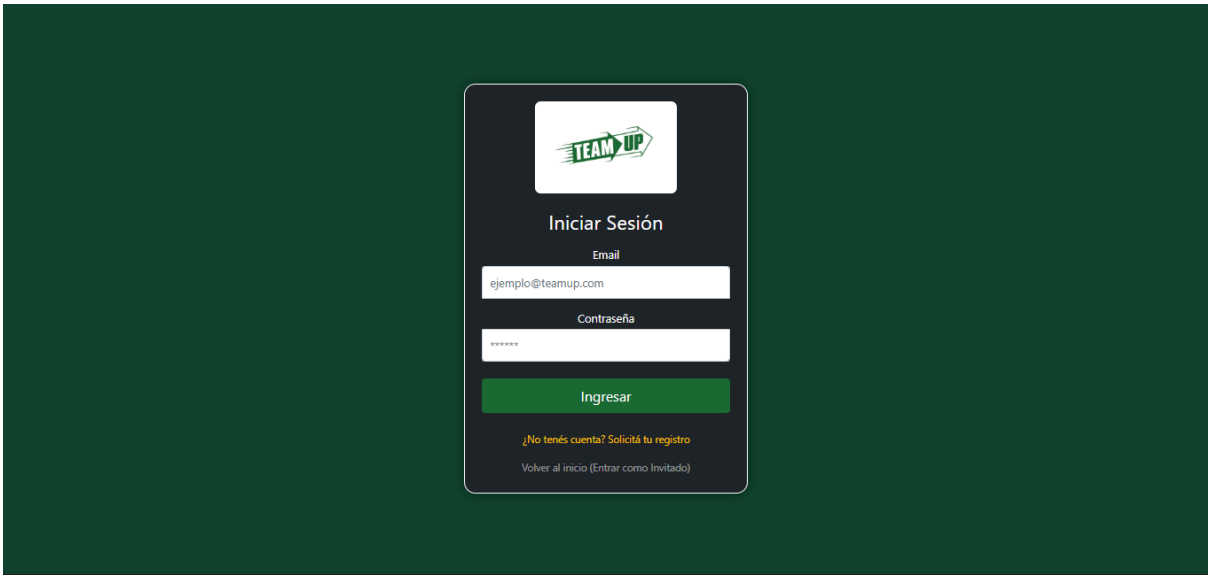
POSTCONDICIONES (de negocio):

- Éxito: El torneo ha sido eliminado, anulándose de esta forma la competencia.
- Éxito Alternativo: —
- Fracaso: —

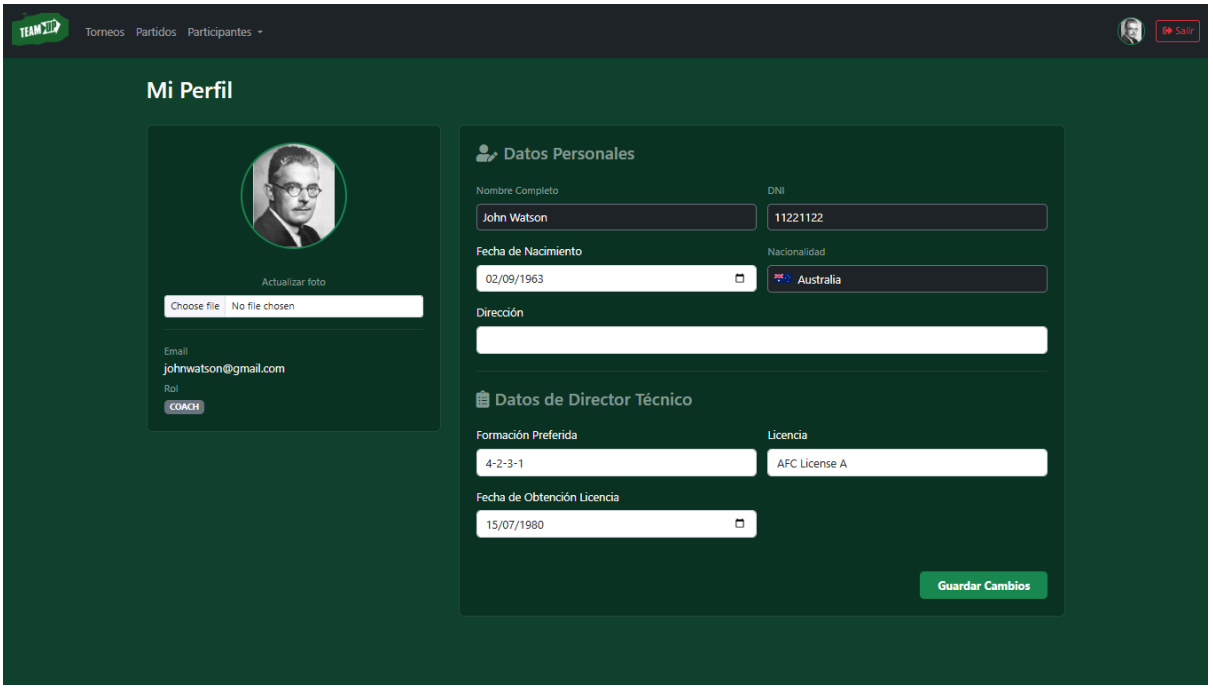
Capturas de pantalla del sistema



Login

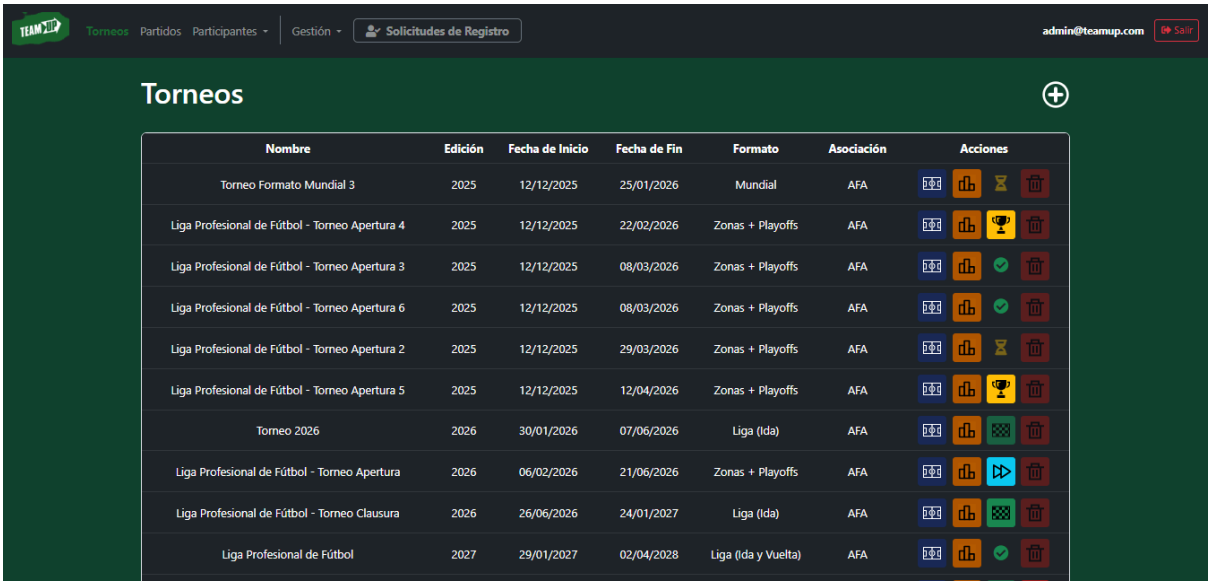


Perfil del Usuario Logueado

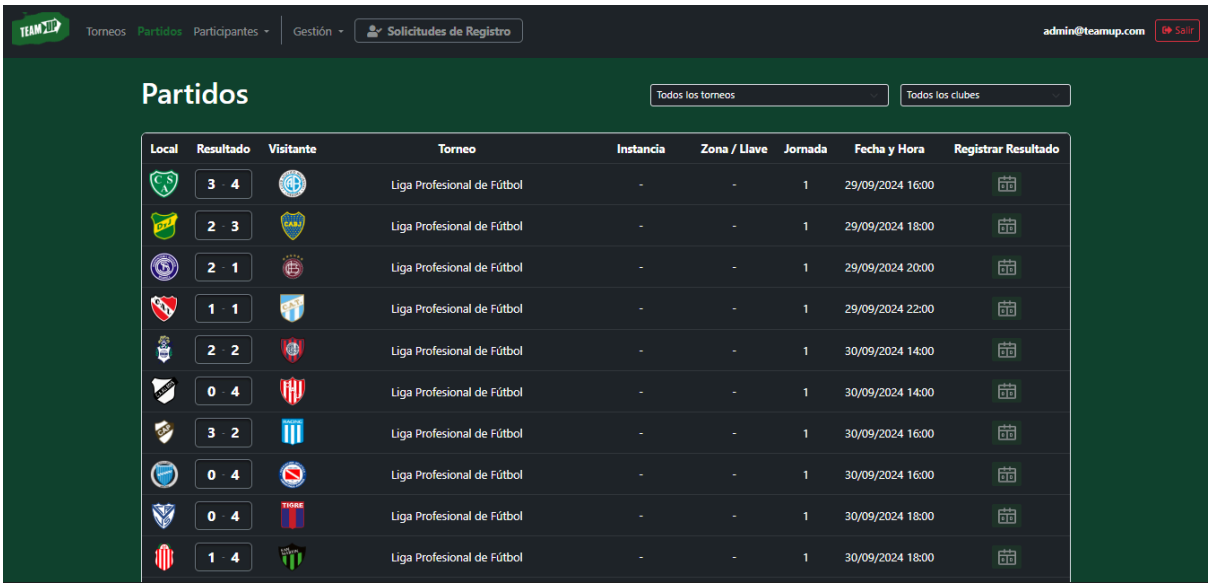


Home

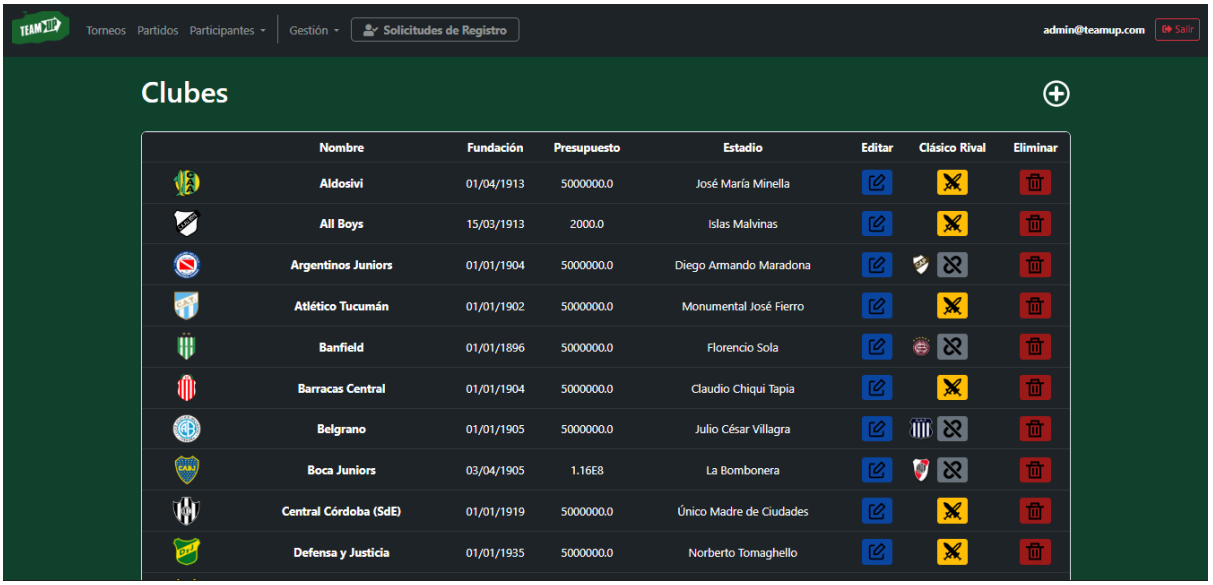
Listado de Torneos



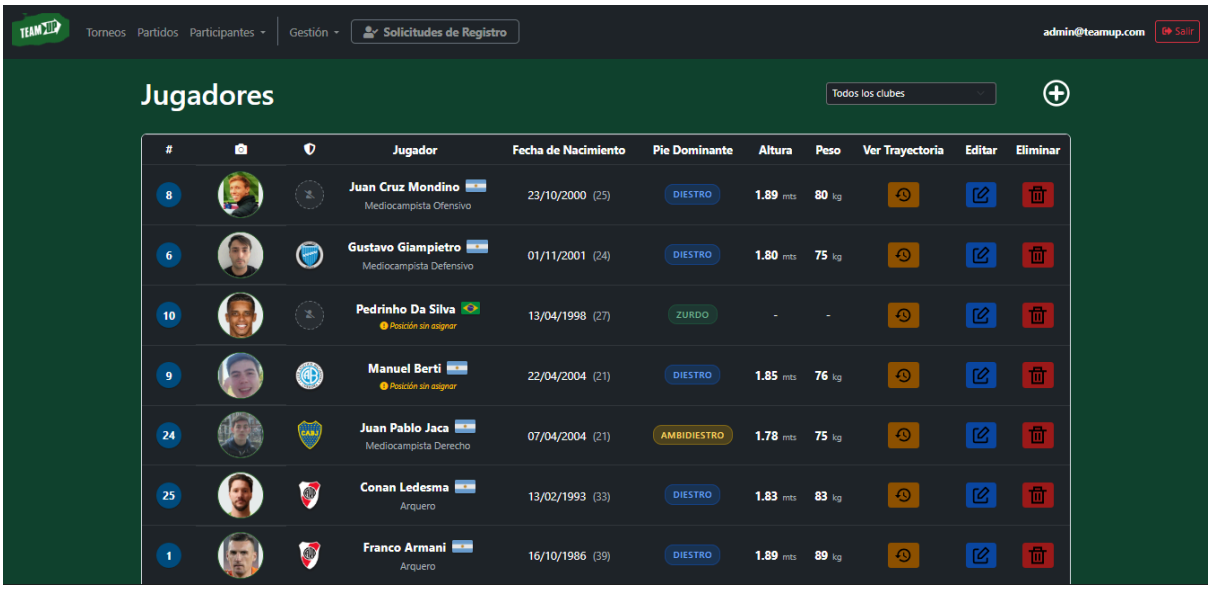
Listado de Partidos



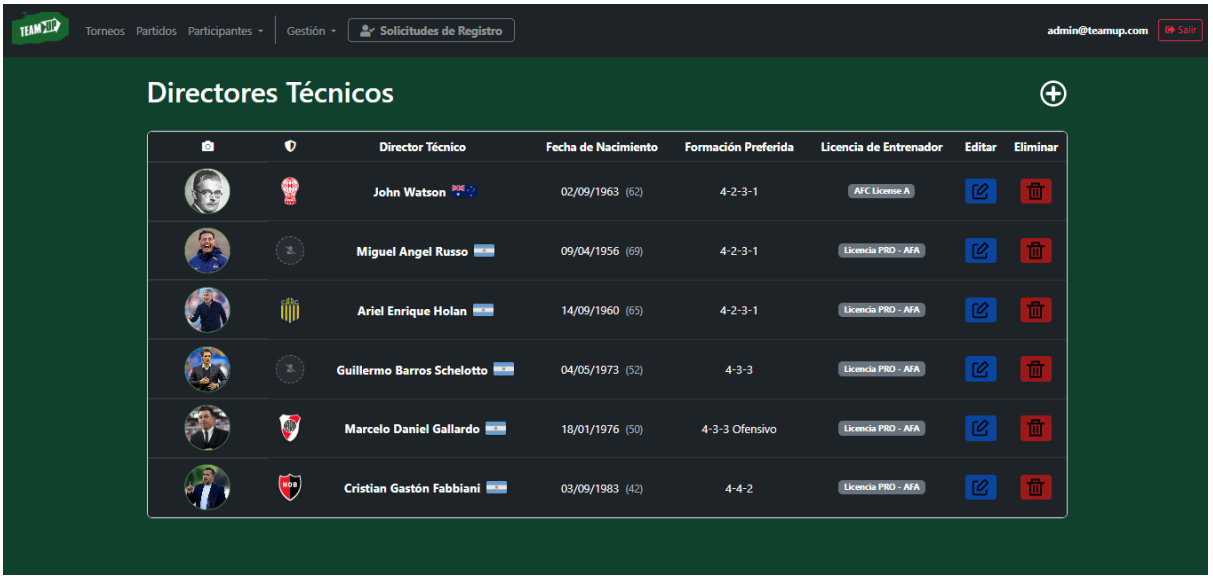
Listado de Clubes



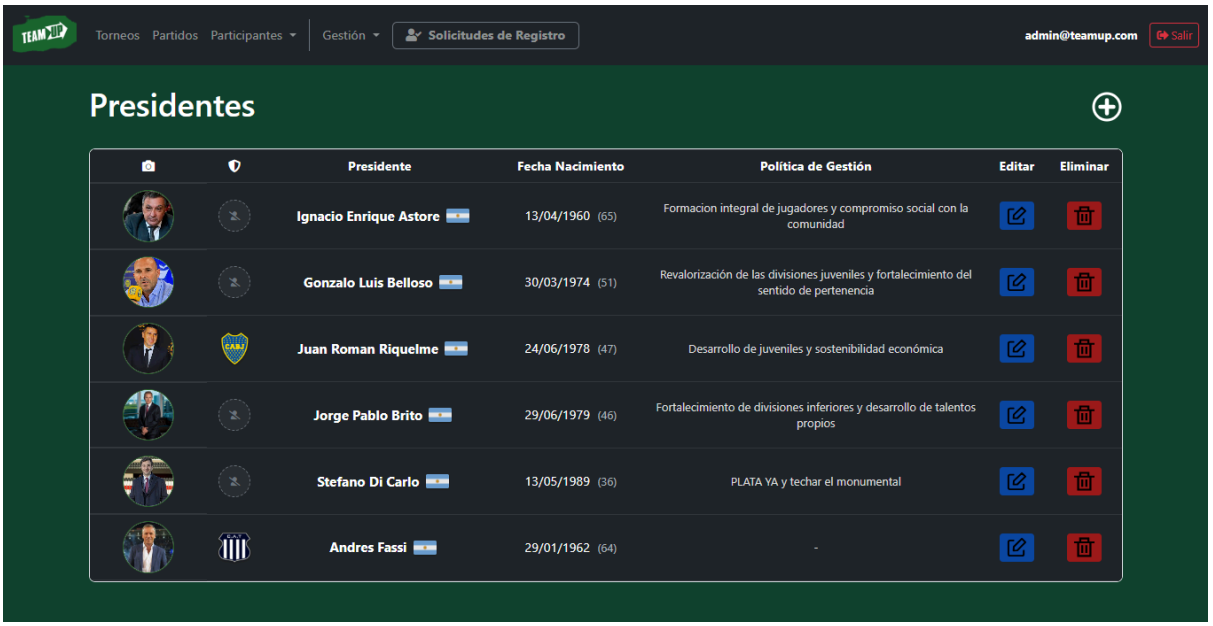
Listado de Jugadores



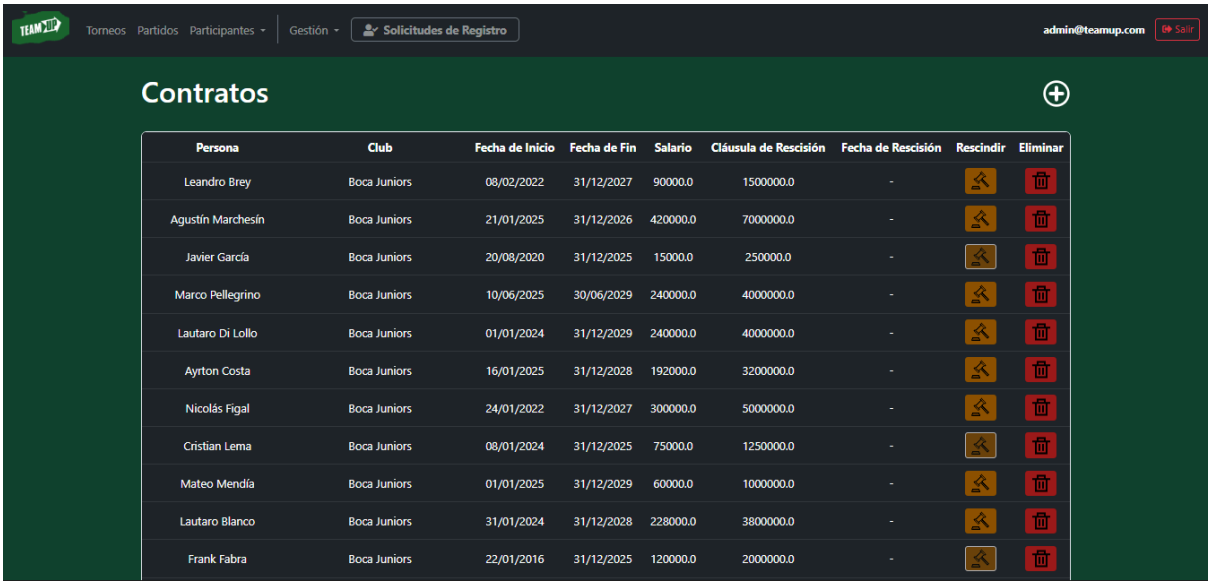
Listado de Directores Técnicos



Listado de Presidentes

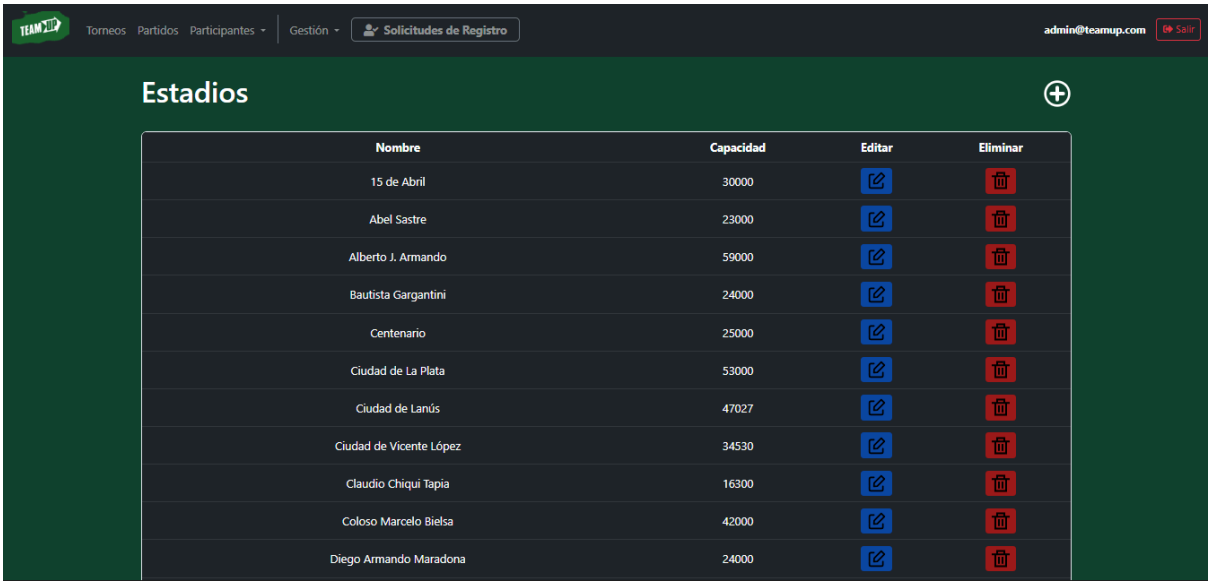


Listado de Contratos



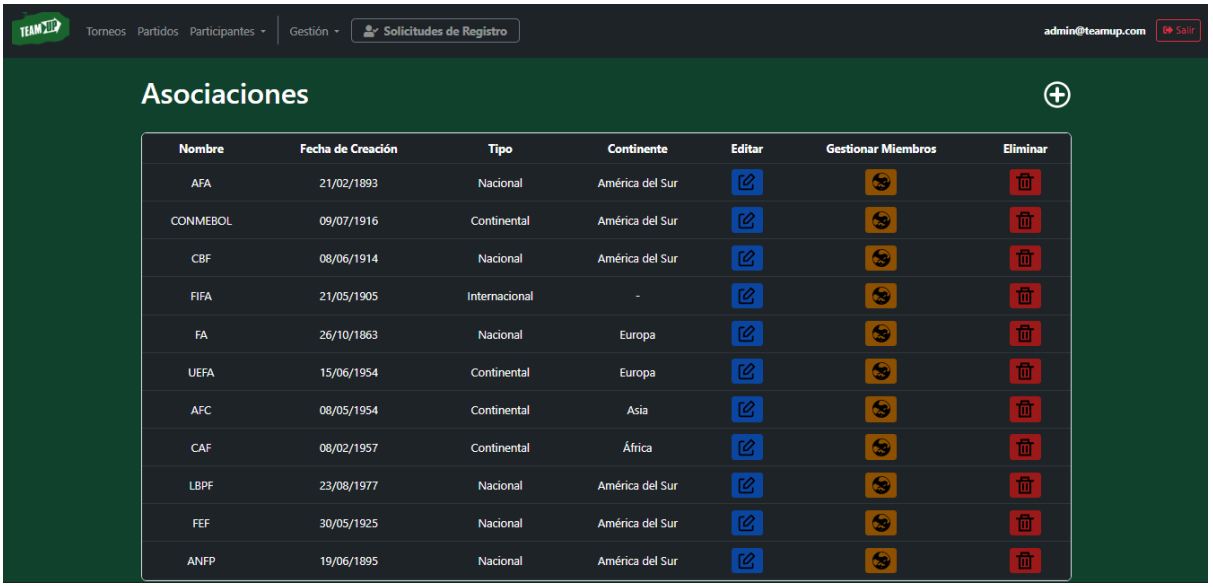
Persona	Club	Fecha de Inicio	Fecha de Fin	Salario	Cláusula de Rescisión	Fecha de Rescisión	Rescindir	Eliminar
Leandro Brey	Boca Juniors	08/02/2022	31/12/2027	90000.0	1500000.0	-		
Agustín Marchesín	Boca Juniors	21/01/2025	31/12/2026	420000.0	7000000.0	-		
Javier García	Boca Juniors	20/08/2020	31/12/2025	15000.0	250000.0	-		
Marco Pellegrino	Boca Juniors	10/06/2025	30/06/2029	240000.0	4000000.0	-		
Lautaro Di Lollo	Boca Juniors	01/01/2024	31/12/2029	240000.0	4000000.0	-		
Ayrton Costa	Boca Juniors	16/01/2025	31/12/2028	192000.0	3200000.0	-		
Nicolás Figal	Boca Juniors	24/01/2022	31/12/2027	300000.0	5000000.0	-		
Cristian Lema	Boca Juniors	08/01/2024	31/12/2025	75000.0	1250000.0	-		
Mateo Mendía	Boca Juniors	01/01/2025	31/12/2029	60000.0	1000000.0	-		
Lautaro Blanco	Boca Juniors	31/01/2024	31/12/2028	228000.0	3800000.0	-		
Frank Fabra	Boca Juniors	22/01/2016	31/12/2025	120000.0	2000000.0	-		

Listado de Estadios



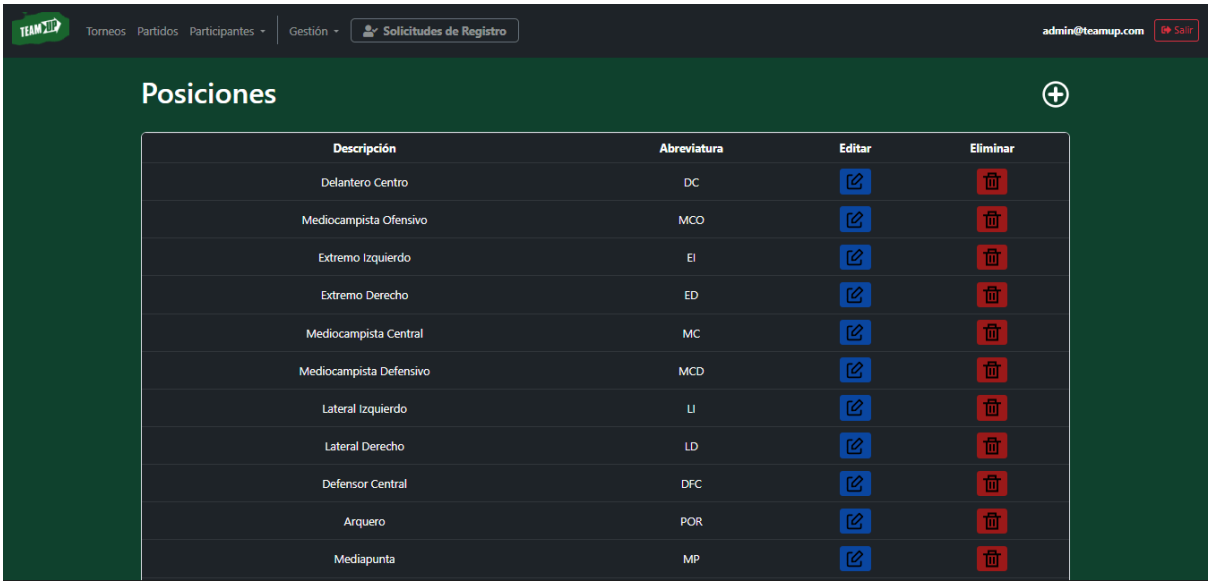
Nombre	Capacidad	Editar	Eliminar
15 de Abril	30000		
Abel Sastre	23000		
Alberto J. Armando	59000		
Bautista Gargantini	24000		
Centenario	25000		
Ciudad de La Plata	53000		
Ciudad de Lanús	47027		
Ciudad de Vicente López	34530		
Claudio Chiqui Tapia	16300		
Coloso Marcelo Bielsa	42000		
Diego Armando Maradona	24000		

Listado de Asociaciones



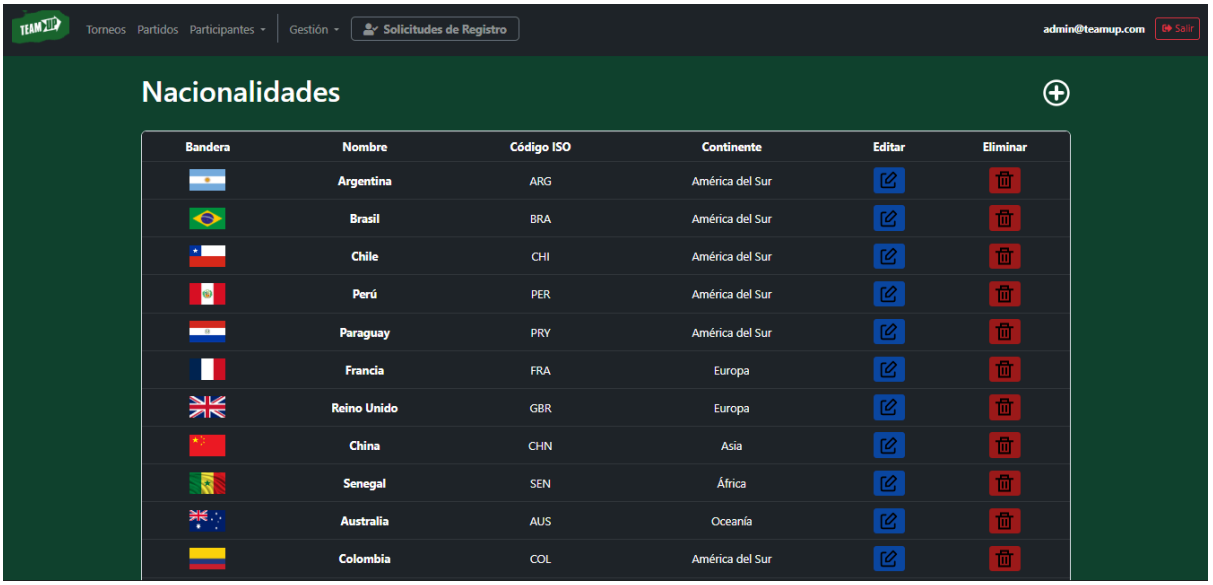
Nombre	Fecha de Creación	Tipo	Continente	Editar	Gestionar Miembros	Eliminar
AFA	21/02/1893	Nacional	América del Sur			
CONMEBOL	09/07/1916	Continental	América del Sur			
CBF	08/06/1914	Nacional	América del Sur			
FIFA	21/05/1905	Internacional	-			
FA	26/10/1863	Nacional	Europa			
UEFA	15/06/1954	Continental	Europa			
AFC	08/05/1954	Continental	Asia			
CAF	08/02/1957	Continental	África			
LBPF	23/08/1977	Nacional	América del Sur			
FEF	30/05/1925	Nacional	América del Sur			
ANFP	19/06/1895	Nacional	América del Sur			

Listado de Posiciones



Descripción	Abreviatura	Editar	Eliminar
Delantero Centro	DC		
Mediocampista Ofensivo	MCO		
Extremo Izquierdo	EI		
Extremo Derecho	ED		
Mediocampista Central	MC		
Mediocampista Defensivo	MCD		
Lateral Izquierdo	LI		
Lateral Derecho	LD		
Defensor Central	DFC		
Arquero	POR		
Mediapunta	MP		

Listado de Nacionalidades



Listado de Solicitudes de Registro

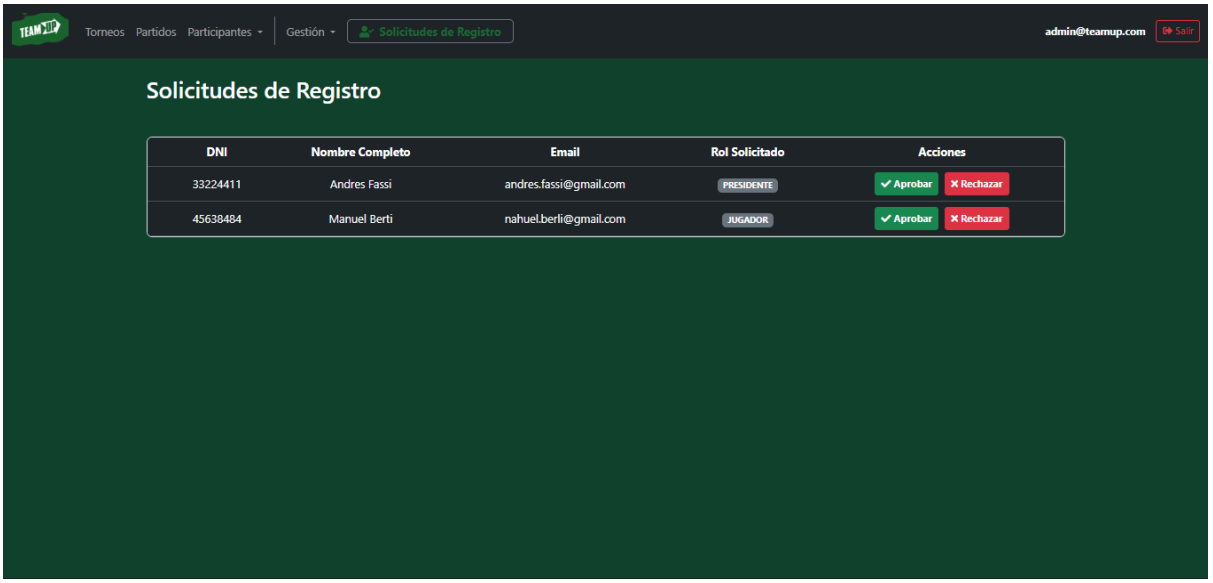
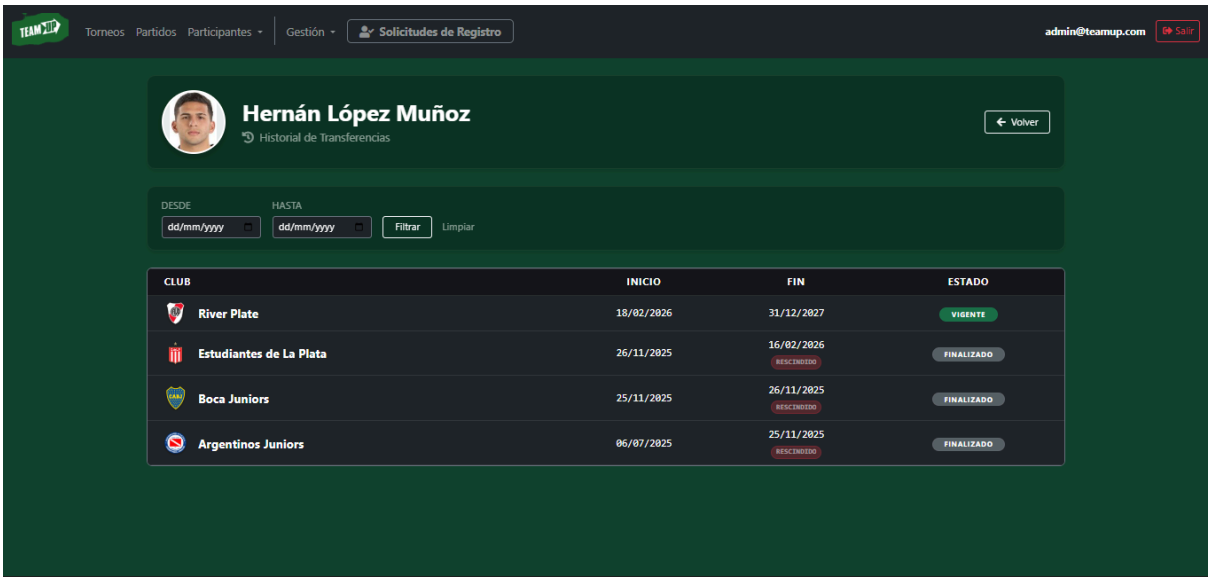


Tabla de Posiciones (Formato Mundial)

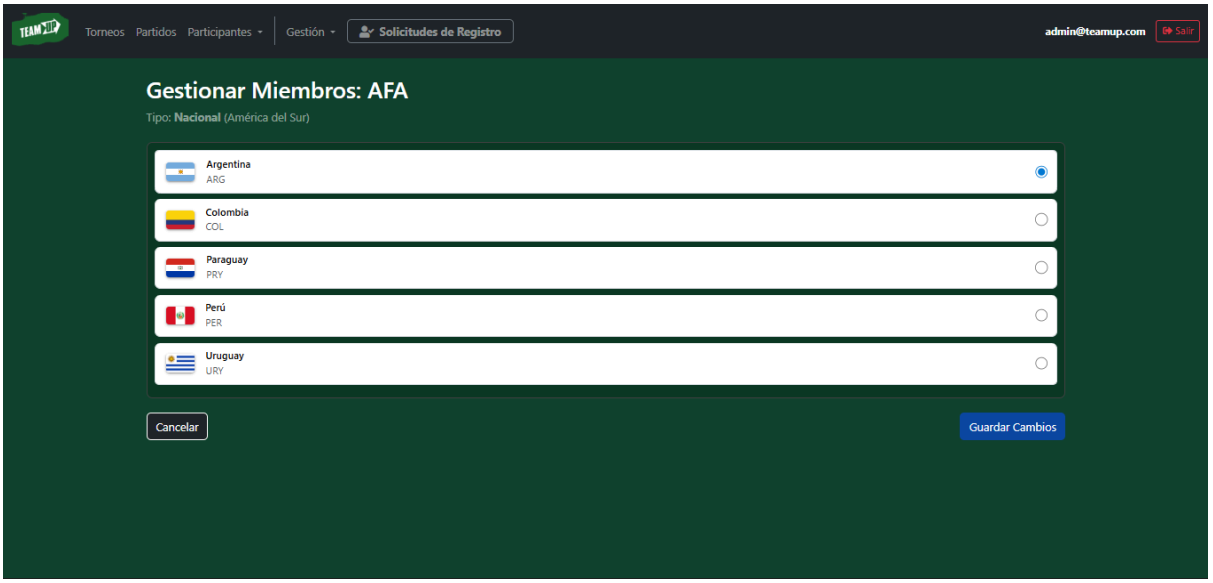
Tabla de Posiciones (Formato Dos Zonas + Eliminación)

Tabla de Posiciones (Formato Liga)

Historial de Clubes de un Jugador



Gestión de Miembros de una Asociación



Fragmentos de código para la defensa

A continuación, se muestra todo el código involucrado en el **CUU_1.1 Generar Fixture de Torneos**:

AddTournament.jsp

```
<%@ page import="java.util.LinkedList"%>
<%@ page import="java.util.HashMap" %>
<%@ page import="entities.*"%>
<%@ page import="enums.TournamentFormat"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%
   LinkedList<Association> associationsList = (LinkedList<Association>) request.getAttribute("associationsList");
        HashMap<Integer, LinkedList<Club>> clubsMap = (HashMap<Integer, LinkedList<Club>>) request.getAttribute("clubsMap");
%>

<!DOCTYPE html>
<html>
        <head>
           <meta charset="UTF-8">
           <title>Agregar Torneo</title>
           <link href="style/bootstrap.css" rel="stylesheet">
           <link rel="icon" type="image/x-icon" href="assets/favicon.png">
           <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css">

           <style>

              .button-container {
                 display: flex;
```

```
    justify-content: space-between;
    align-items: center;
    margin-top: 20px;
}

.club-static {
    opacity: 0.8;
    background-color: #e9ecef !important;
    border: 1px solid #adb5bd !important;
    cursor: default !important;
}

            #loadingOverlay {
                position: fixed;
                top: 0;
                left: 0;
                width: 100%;
                height: 100%;
                background-color: rgba(0, 0, 0, 0.85);
                z-index: 9999;
                display: none;
                flex-direction: column;
                justify-content: center;
                align-items: center;
                backdrop-filter: blur(5px);
            }

            @keyframes spin-ball {
                0% { transform: rotate(0deg); }
                100% { transform: rotate(360deg); }
            }
```

```
                .soccer-spinner {
                    font-size: 4rem;
                    color: #ffffff;
                    animation: spin-ball 1.5s linear infinite;
                    filter: drop-shadow(0 0 10px rgba(255, 255, 255, 0.5));
                }

            </style>
        </head>

        <body style="background-color: #10442E;">
            <jsp:include page="/WEB-INF/Navbar.jsp"></jsp:include>

            <div class="container" style="color: white;">
                <h2 class="mt-4">Agregar Torneo</h2>

                <form action="actiontournament" method="post" class="mt-4" onsubmit="return validateClubs()">
                    <input type="hidden" name="action" value="add" />
                    <div class="form-group">
                        <label for="name">Nombre:</label>
                        <input type="text" class="form-control" id="name" name="name" required />
                    </div>
                    <div class="form-group">
                                    <label for="startDate">Fecha de Inicio (Aproximada):</label>
                                    <input type="date" class="form-control" id="startDate" name="startDate" required />
                                    <small class="form-text text-white-50 d-flex justify-content-start gap-2 mt-1">
                                        <span><i class="fas fa-info-circle"></i></span>
                                        <span>Se ajustará automáticamente al <b>viernes</b> de la semana de la fecha seleccionada. Si ese viernes ya pasó,
se moverá al de la semana siguiente.</span>
                                    </small>
```

```
                    </div>
<div class="form-group">
    <label for="format">Formato:</label>
    <select name="format" id="format" class="form-control" required>
        <option value="">-- Seleccioná un formato --</option>
        <% for (TournamentFormat format : TournamentFormat.values()) { %>
                        <option value="<%= format.name() %>">
                            <%= format.getDescription() %>
                        </option>
                    <% } %>
    </select>
</div>
<div class="form-group">
    <label for="season">Edición:</label>
    <input type="text" class="form-control" id="season" name="season" required />
</div>
<div class="form-group">
            <label for="id_association">Asociación:</label>
            <select name="id_association" id="id_association" class="form-control" required onchange="showClubs(this.value)">
                <option value="">-- Seleccioná una asociación --</option>
                <% for (Association a : associationsList) { %>
                    <option value="<%= a.getId() %>"><%= a.getName() %></option>
                <% } %>
            </select>
        </div>

        <div class="mt-4">
                <% for (Association a : associationsList) {
                    LinkedList<Club> clubs = clubsMap.get(a.getId());
                    boolean hasClubs = (clubs != null && !clubs.isEmpty());
                    boolean isInsufficient = (hasClubs && clubs.size() < 2);
```

```
%>
   <div id="list_<%= a.getId() %>" class="clubs-list" style="display: none;">

      <% if (hasClubs) { %>

         <% if (!isInsufficient) { %>
               <label class="text-white fs-5">Seleccioná los clubes participantes:</label>

               <div class="my-2">
                     <button type="button"
                                                id="btnToggle_<%= a.getId() %>"
                                                class="btn btn-sm text-white"
                                                style="background-color:#1A6B32;"
                                                onclick="toggleAllClubs()">
                                             <i class="fas fa-check-square me-1"></i> Seleccionar todos
                                          </button>
                                    </div>
                              <% } else { %>
                                 <div class="alert alert-warning d-flex align-items-center mt-2 py-2" role="alert">
                                    <i class="fas fa-exclamation-circle me-2 flex-shrink-0"></i>
                                    <div>
                                       Esta asociación solo tiene <strong>1</strong> club registrado.
                                       Necesitás al menos <strong>2</strong> para crear un torneo.
                                    </div>
                                 </div>

                                 <label class="text-white-50 fs-6 mb-1 mt-2">Único integrante:</label>
                              <% } %>

                              <div class="row row-cols-2 row-cols-md-3 row-cols-lg-4 g-3 mt-1 mb-3">
                  <% for (Club c : clubs) { %>
```

```
                                        <div class="col">
                                          <% if (!isInsufficient) { %>
                                                <label class="p-2 rounded d-flex align-items-center gap-2 bg-white text-dark border border-success
w-100 h-100"
                                                       style="cursor: pointer;">
                                                    <input type="checkbox" name="selectedClubs" value="<%= c.getId() %>" class="club-checkbox"
onchange="updateCount()">
                                                    <img src="<%=request.getContextPath() + "/images?id=" + c.getBadgeImage()%>" width="30"
height="30" style="object-fit:contain;">
                                                    <small class="m-0 fw-bold"><%= c.getName() %></small>
                                                </label>
                                            <% } else { %>
                                                <div class="p-2 rounded d-flex align-items-center gap-2 club-static w-100 h-100 text-dark">
                                                    <img src="<%=request.getContextPath() + "/images?id=" + c.getBadgeImage()%>" width="30"
height="30" style="object-fit:contain; opacity: 0.75;">
                                                    <small class="m-0 fw-bold"><%= c.getName() %></small>
                                                </div>
                                            <% } %>
                                        </div>
                                    <% } %>
                                </div>

                                                <% if (!isInsufficient) { %>
                                                        <p class="mt-2 text-white">
                                    <strong>Clubes seleccionados:</strong> <span class="count-display">0</span>
                                    <span class="error-msg text-warning ms-3" style="display:none; font-size: 0.9em;">
                                       <i class="fas fa-exclamation-triangle me-1"></i> Debés seleccionar al menos 2 equipos.
                                    </span>
                                 </p>
                                <% } %>
```

```
                                <% } else { %>
                                 <p class="text-warning mt-3 p-3 border border-warning rounded" style="background-color: rgba(255, 193, 7, 0.1);">
                                      <i class="fas fa-exclamation-circle me-2"></i> No hay clubes registrados para <b><%= a.getName() %></b>.
                                   </p>
                                <% } %>

                              </div>
                            <% } %>
                          </div>

                          <div id="globalError" class="alert alert-warning align-items-center mt-3" style="display: none;" role="alert">
                            <i class="fas fa-exclamation-triangle me-2"></i>
                            <span id="globalErrorText"></span>
                          </div>

                    <div class="button-container my-4">
                       <button type="button" class="btn btn-dark border border-white" onclick="history.back()">Cancelar</button>
                       <button type="submit" class="btn text-white" style="background-color: #0D47A1">Agregar</button>
                    </div>

                  </form>
                </div>

                <div id="loadingOverlay">
                        <i class="fas fa-futbol soccer-spinner mb-4"></i>

                        <h3 class="text-white fw-bold">Creando Torneo...</h3>
                        <p class="text-white-50">Generando el fixture, esto puede demorar unos segundos</p>
                     </div>

            <script>
```

```
function checkValidity() {

        var associationId = document.getElementById("id_association").value;
    var format = document.getElementById("format").value;
    var btn = document.getElementById("btnSubmit");
    var isValid = false;
    var errorMessage = "";

    if (associationId && format) {

        var container = document.getElementById('list_' + associationId);

       if (container) {

           var total = container.querySelectorAll(".club-checkbox:checked").length;


           if (format === "ZONAL_ELIMINATION") {

                   if (total >= 16 && total <= 32 && total % 2 === 0) {

                           isValid = true;

                   }

           } else if (format === "ROUND_ROBIN_ONE_LEG" || format === "ROUND_ROBIN_TWO_LEGS") {

                   if (total >= 8 && total <= 32) {

                           isValid = true;
```

```
                }

            } else if (format === "WORLD_CUP") {

                if (total === 32) {

                    isValid = true;

                }

            }

        }

    }

    if (btn) {

        btn.disabled = !isValid;
        btn.style.opacity = isValid ? "1" : "0.5";
        btn.style.cursor = isValid ? "pointer" : "not-allowed";

    }

    return isValid;

}

function validateClubs() {
```

```
    var associationId = document.getElementById("id_association").value;
var format = document.getElementById("format").value;

var globalError = document.getElementById("globalError");
var globalErrorText = document.getElementById("globalErrorText");

if (globalError) globalError.style.display = "none";

if (!associationId) return false;

var container = document.getElementById('list_' + associationId);

if (!container || container.querySelectorAll(".club-checkbox").length === 0) {

    globalErrorText.textContent = "La asociación seleccionada no tiene suficientes clubes para crear un torneo.";
   globalError.style.display = "flex";
   globalError.scrollIntoView({ behavior: 'smooth', block: 'center' });
   return false;

}

const total = container.querySelectorAll(".club-checkbox:checked").length;
const errorSpan = container.querySelector(".error-msg");
let errorMessage = "";
let isValid = true;

if (total < 2) {

    errorMessage = "Debés seleccionar al menos 2 equipos.";
   isValid = false;
```

```
} else if (format === "ZONAL_ELIMINATION") {

    if (total < 16 || total > 32) {

            errorMessage = "Para Formato Zonal debés seleccionar entre 16 y 32 equipos (Seleccionados: " + total + ").";
      isValid = false;

    } else if (total % 2 !== 0) {

            errorMessage = "Para Formato Zonal debés seleccionar una cantidad PAR de equipos.";
      isValid = false;

    }

} else if ((format === "ROUND_ROBIN_ONE_LEG" || format === "ROUND_ROBIN_TWO_LEGS") && (total < 8 || total > 32)) {

     errorMessage = "Para Formato Todos contra Todos debés seleccionar entre 8 y 32 equipos.";
    isValid = false;

} else if (format === "WORLD_CUP" && total !== 32) {

     errorMessage = "Para Formato Mundial debés seleccionar exactamente 32 equipos.";
    isValid = false;

}

if (!isValid) {

     if (errorSpan) {

            errorSpan.innerHTML = '<i class="fas fa-exclamation-triangle me-1"></i> ' + errorMessage;
```

```
            errorSpan.style.display = "inline";
            errorSpan.scrollIntoView({ behavior: 'smooth', block: 'center' });

        }

        return false;

    }

    if (errorSpan) errorSpan.style.display = "none";
    document.getElementById("loadingOverlay").style.display = "flex";
    return true;

}

function updateCount() {

        var associationId = document.getElementById("id_association").value;
    if (!associationId) return;

    var container = document.getElementById('list_' + associationId);
    if (container) {

        var checkboxes = container.querySelectorAll(".club-checkbox");

        if (checkboxes.length > 0) {

                const total = container.querySelectorAll(".club-checkbox:checked").length;
            const totalCheckboxes = checkboxes.length;

            var counterSpan = container.querySelector(".count-display");
```

```
        var errorSpan = container.querySelector(".error-msg");
        if (counterSpan) counterSpan.textContent = total;

        if (total >= 2 && errorSpan) errorSpan.style.display = "none";

        var btn = document.getElementById('btnToggle_' + associationId);

        if (btn) {

            const allAreChecked = (total > 0 && total === totalCheckboxes);

            if (allAreChecked) {

                btn.innerHTML = '<i class="fas fa-minus-square me-1"></i> Quitar selección';
            btn.style.backgroundColor = "#c0392b";

            } else {

                btn.innerHTML = '<i class="fas fa-check-square me-1"></i> Seleccionar todos';
            btn.style.backgroundColor = "#1A6B32";

            }

        }

    }

}

checkValidity();
```

```
}

function toggleAllClubs() {

        var associationId = document.getElementById("id_association").value;

        if (!associationId) return;

        var container = document.getElementById('list_' + associationId);

        if (container) {

                const checkboxes = container.querySelectorAll(".club-checkbox");

                if(checkboxes.length === 0) return;

                const allChecked = Array.from(checkboxes).every(cb => cb.checked);
      checkboxes.forEach(cb => cb.checked = !allChecked);

      updateCount();

        }

}

function showClubs(associationId) {

        var globalError = document.getElementById("globalError");
   if (globalError) globalError.style.display = "none";

   var lists = document.querySelectorAll('.clubs-list');
```

```
lists.forEach(function(div) {

     div.style.display = 'none';

     var checkboxes = div.querySelectorAll('input[type="checkbox"]');
    checkboxes.forEach(function(cb) { cb.checked = false; });

    var counterSpan = div.querySelector(".count-display");
    if (counterSpan) counterSpan.textContent = "0";

    var errorSpan = div.querySelector(".error-msg");
    if (errorSpan) errorSpan.style.display = "none";

    var btn = div.querySelector("button[id^='btnToggle_']");

    if (btn) {

     btn.innerHTML = '<i class="fas fa-check-square me-1"></i> Seleccionar todos';
        btn.style.backgroundColor = "#1A6B32";

    }

});

if (associationId) {

     var chosenList = document.getElementById('list_' + associationId);
    if (chosenList) chosenList.style.display = 'block';

}
```

```
                checkValidity();

            }

            window.onload = function() {

                updateCount();
            checkValidity();

            };

        </script>

    </body>
</html>
```

ActionTournament.java (Servlet)

```
package servlet;

import java.io.IOException;
import java.sql.SQLException;
import java.time.LocalDate;
import java.time.DayOfWeek;
import java.util.*;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import entities.*;
import enums.*;
import logic.Logic;

@WebServlet("/actiontournament")
public class ActionTournament extends HttpServlet {

	private static final long serialVersionUID = 1L;

	private Tournament buildTournamentFromRequest(HttpServletRequest request, String action, Logic ctrl) throws SQLException {

		Tournament tournament = new Tournament();
		if ("edit".equals(action)) tournament.setId(Integer.parseInt(request.getParameter("id")));
		tournament.setName(request.getParameter("name"));
```

```
            tournament.setStartDate(LocalDate.parse(request.getParameter("startDate")));
            tournament.setFormat(TournamentFormat.valueOf(request.getParameter("format")));
            tournament.setSeason(request.getParameter("season"));
tournament.setAssociation(ctrl.getAssociationById(Integer.parseInt(request.getParameter("id_association"))));

return tournament;

    }

    private boolean checkStartDate(LocalDate startDate) {

             LocalDate today = LocalDate.now();

               if (startDate.isBefore(today)) {

                     return false;

               }

               return true;

    }

    private LocalDate adjustStartDateToFriday(LocalDate inputDate) {

       LocalDate fridayOfThatWeek = inputDate.with(DayOfWeek.FRIDAY);

       if (fridayOfThatWeek.isBefore(LocalDate.now())) {

            return fridayOfThatWeek.plusWeeks(1);
```

```
    }

    return fridayOfThatWeek;

}

private void redistributeMatchesByMatchday(LinkedList<Match> allMatches, LocalDate startDate, boolean isWorldCupFormat, Logic ctrl) {

    HashMap<Integer, LinkedList<Match>> matchesByDay = new HashMap<>();

    for (Match m : allMatches) {

            matchesByDay.computeIfAbsent(m.getMatchday(), k -> new LinkedList<>()).add(m);

    }

    for (Map.Entry<Integer, LinkedList<Match>> entry : matchesByDay.entrySet()) {

            int matchdayNum = entry.getKey();
        LinkedList<Match> matchesOfThisDate = entry.getValue();

        if (isWorldCupFormat) {

            matchesOfThisDate.sort(Comparator.comparing(Match::getGroupName));

        } else {

           Collections.shuffle(matchesOfThisDate);

        }
```

```
            LocalDate fridayOfMatchday = startDate.plusWeeks(matchdayNum - 1);

            ctrl.assignWeekendSchedule(matchesOfThisDate, fridayOfMatchday, isWorldCupFormat);

        }

    }

    private LinkedList<Match> generateInterzonalMatches(LinkedList<Club> zoneA, LinkedList<Club> zoneB, int matchday, HashMap<Integer, Club>
classicRivalries) {

        LinkedList<Match> interzonalMatches = new LinkedList<>();

        HashSet<Integer> assignedA = new HashSet<>();
        HashSet<Integer> assignedB = new HashSet<>();

        for (Club cA : zoneA) {

            if (classicRivalries.containsKey(cA.getId())) {

                int classicRivalId = classicRivalries.get(cA.getId()).getId();

                for (Club cB : zoneB) {

                    if (cB.getId() == classicRivalId && !assignedB.contains(cB.getId())) {

                        Match m = new Match();

                        if (Math.random() < 0.5) {

                            m.setHome(cA);
```

```
            m.setAway(cB);

                                    } else {

                                            m.setHome(cB);
            m.setAway(cA);

                                    }

                                    m.setMatchday(matchday);

                                    m.setStage(TournamentStage.GROUP_STAGE);
                                    m.setGroupName("Interzonal");

                                    interzonalMatches.add(m);

                                    assignedA.add(cA.getId());
                                    assignedB.add(cB.getId());

                                    break;

                            }

                    }

            }

    }

LinkedList<Club> remainingA = new LinkedList<>();
for (Club c : zoneA) if (!assignedA.contains(c.getId())) remainingA.add(c);
```

```
        LinkedList<Club> remainingB = new LinkedList<>();
        for (Club c : zoneB) if (!assignedB.contains(c.getId())) remainingB.add(c);

        Collections.shuffle(remainingA);
        Collections.shuffle(remainingB);

        int count = Math.min(remainingA.size(), remainingB.size());

        for (int i = 0; i < count; i++) {

                Match m = new Match();

                m.setHome(remainingA.get(i));
                m.setAway(remainingB.get(i));

                m.setMatchday(matchday);

                m.setStage(TournamentStage.GROUP_STAGE);
                m.setGroupName("Interzonal");

                interzonalMatches.add(m);

        }

        return interzonalMatches;

}

private LinkedList<Match> generateZonalPhase(LinkedList<Club> allClubs, LocalDate startDate, Logic ctrl) throws SQLException {
```

```
                if (allClubs.size() < 16 || allClubs.size() > 32 || allClubs.size() % 2 != 0) {

                        throw new SQLException("El formato ZONAL requiere una cantidad PAR de equipos, entre 16 y 32 en total. (Seleccionados: " +
allClubs.size() + ")");

                }


                LinkedList<Match> allMatches = new LinkedList<>();

            HashMap<Integer, Club> classicRivalries = ctrl.getClassicRivalsMap();

            LinkedList<Club> zoneA = new LinkedList<>();
            LinkedList<Club> zoneB = new LinkedList<>();

            HashMap<Integer, Club> clubsMap = new HashMap<>();
            for (Club c : allClubs) clubsMap.put(c.getId(), c);

            HashSet<Integer> assignedIds = new HashSet<>();

            for (Club c : allClubs) {

               if (assignedIds.contains(c.getId())) continue;

               if (classicRivalries.containsKey(c.getId())) {

                   Club classicRivalObj = classicRivalries.get(c.getId());
                  int classicRivalId = classicRivalObj.getId();

                  if (clubsMap.containsKey(classicRivalId)) {
```

```
        Club classicRival = clubsMap.get(classicRivalId);

          zoneA.add(c);
          zoneB.add(classicRival);

          assignedIds.add(c.getId());
          assignedIds.add(classicRivalId);

       }

   }

}

LinkedList<Club> remaining = new LinkedList<>();

for (Club c : allClubs) {

       if (!assignedIds.contains(c.getId())) {

                remaining.add(c);

       }

}

Collections.shuffle(remaining);

for (Club c : remaining) {

       if (zoneA.size() <= zoneB.size()) {
```

```
                zoneA.add(c);

        } else {

                zoneB.add(c);

        }

}

LinkedList<Match> matchesA = generateFirstLeg(zoneA, startDate);
LinkedList<Match> matchesB = generateFirstLeg(zoneB, startDate);

int totalZonalDates = matchesA.getLast().getMatchday();
int interzonalDay = (totalZonalDates / 2) + 1;

for (Match m : matchesA) {

        if (m.getMatchday() >= interzonalDay) {

                m.setMatchday(m.getMatchday() + 1);

        }

        m.setStage(TournamentStage.GROUP_STAGE);
    m.setGroupName("Zona A");

}

for (Match m : matchesB) {
```

```
            if (m.getMatchday() >= interzonalDay) {

                    m.setMatchday(m.getMatchday() + 1);

            }

            m.setStage(TournamentStage.GROUP_STAGE);
        m.setGroupName("Zona B");

    }

    LinkedList<Match> interzonalMatches = generateInterzonalMatches(zoneA, zoneB, interzonalDay, classicRivalries);

    allMatches.addAll(matchesA);
    allMatches.addAll(matchesB);

    allMatches.addAll(interzonalMatches);

    return allMatches;

}

private LinkedList<Match> generateFirstLeg(LinkedList<Club> clubs, LocalDate startDate) {

            LinkedList<Match> fixture = new LinkedList<>();

            LinkedList<Club> rotated = new LinkedList<>(clubs);

            int n = rotated.size();
```

```
        if (n % 2 != 0) {

                rotated.add(null);
            n++;

        }

        int totalMatchdays = n - 1;
        int matchesPerMatchday = n / 2;

for (int day = 1; day <= totalMatchdays; day++) {

        LocalDate baseDate = startDate.plusWeeks(day - 1);

            for (int i = 0; i < matchesPerMatchday; i++) {

                Club home = rotated.get(i);
                Club away = rotated.get(n - 1 - i);

                if (home == null || away == null) continue;

                if (day % 2 == 0) {

                    Club temp = home;
                   home = away;
                   away = temp;

                }

                Match match = new Match();
```

```
                match.setHome(home);
                match.setAway(away);

                match.setStage(TournamentStage.GROUP_STAGE);
                match.setMatchday(day);

                match.setDate(baseDate.atStartOfDay());

                fixture.add(match);

            }

            rotated.add(1, rotated.remove(rotated.size() - 1));

        }

        return fixture;

}

private LinkedList<Match> generateSecondLeg(LinkedList<Match> firstLegMatches, int numberOfClubs) {

        LinkedList<Match> secondLegMatches = new LinkedList<>();

        int totalMatchdaysFirstLeg = (numberOfClubs % 2 == 0) ? (numberOfClubs - 1) : numberOfClubs;

    for (Match m : firstLegMatches) {

        Match returnMatch = new Match();

        returnMatch.setHome(m.getAway());
```

```
                returnMatch.setAway(m.getHome());

                returnMatch.setStage(TournamentStage.GROUP_STAGE);
                returnMatch.setMatchday(m.getMatchday() + totalMatchdaysFirstLeg);

                returnMatch.setDate(m.getDate().plusWeeks(totalMatchdaysFirstLeg));

            secondLegMatches.add(returnMatch);

        }

        return secondLegMatches;

    }

    private LinkedList<Match> generateWorldCupFixture(LinkedList<Club> allClubs, LocalDate startDate, Logic ctrl) throws SQLException {

            if (allClubs.size() != 32) {

                    throw new SQLException("El formato MUNDIAL requiere exactamente 32 equipos seleccionados. (Seleccionados: " + allClubs.size()
+ ")");

            }

            LinkedList<Match> allMatches = new LinkedList<>();

        List<LinkedList<Club>> groups = new ArrayList<>();

        for (int i = 0; i < 8; i++) groups.add(new LinkedList<>());

        HashMap<Integer, Club> classicRivalries = ctrl.getClassicRivalsMap();
```

```
Collections.shuffle(allClubs);

for (Club c : allClubs) {

    boolean assigned = false;

    List<Integer> groupIndexes = new ArrayList<>();

    for (int i = 0; i < 8; i++) groupIndexes.add(i);

    groupIndexes.sort(Comparator.comparingInt(i -> groups.get(i).size()));

    for (int i : groupIndexes) {

        LinkedList<Club> currentGroup = groups.get(i);

       if (currentGroup.size() >= 4) continue;

       boolean conflict = false;

       if (classicRivalries.containsKey(c.getId())) {

        Club classicRivalObj = classicRivalries.get(c.getId());
          int classicRivalId = classicRivalObj.getId();

          for (Club member : currentGroup) {

                  if (member.getId() == classicRivalId) {

                          conflict = true;
```

```
                    break;

                     }

               }

           }

           if (!conflict) {

            currentGroup.add(c);
               assigned = true;
               break;

           }

       }

       if (!assigned) {

           groups.get(groupIndexes.get(0)).add(c);

       }

   }

   char groupChar = 'A';

   for (LinkedList<Club> group : groups) {

           if (group.size() < 2) {
```

```
                groupChar++;
                continue;

            }

        LinkedList<Match> groupMatches = generateFirstLeg(group, startDate);

        String groupName = "Grupo " + groupChar;

        for (Match m : groupMatches) {

            m.setStage(TournamentStage.GROUP_STAGE);
           m.setGroupName(groupName);

        }

        allMatches.addAll(groupMatches);
        groupChar++;

    }

    return allMatches;

}

private LinkedList<Match> drawTournamentMatchdays(LinkedList<Club> clubs, Tournament tournament, Logic ctrl) throws SQLException {

        LinkedList<Match> matches = new LinkedList<>();

        LinkedList<Club> shuffledClubs = new LinkedList<>(clubs);
```

```
            Collections.shuffle(shuffledClubs);

            boolean isWorldCup = false;

                switch (tournament.getFormat()) {

                    case ZONAL_ELIMINATION:

                        matches = generateZonalPhase(shuffledClubs, tournament.getStartDate(), ctrl);

                        break;

                    case ROUND_ROBIN_ONE_LEG:

                        if (shuffledClubs.size() < 8 || shuffledClubs.size() > 32) {

                            throw new SQLException("El formato TODOS CONTRA TODOS requiere entre 8 y 32 equipos seleccionados.
(Seleccionados: " + shuffledClubs.size() + ")");

                        }

                        matches = generateFirstLeg(shuffledClubs, tournament.getStartDate());

                        break;

                    case ROUND_ROBIN_TWO_LEGS:

                        if (shuffledClubs.size() < 8 || shuffledClubs.size() > 32) {

                            throw new SQLException("El formato TODOS CONTRA TODOS requiere entre 8 y 32 equipos seleccionados.
(Seleccionados: " + shuffledClubs.size() + ")");
```

```
                        }

        LinkedList<Match> firstLeg = generateFirstLeg(shuffledClubs, tournament.getStartDate());
        matches.addAll(firstLeg);

        LinkedList<Match> secondLeg = generateSecondLeg(firstLeg, shuffledClubs.size());
        matches.addAll(secondLeg);

        break;

                case WORLD_CUP:

                        matches = generateWorldCupFixture(shuffledClubs, tournament.getStartDate(), ctrl);

                        isWorldCup = true;

                        break;

        }

        redistributeMatchesByMatchday(matches, tournament.getStartDate(), isWorldCup, ctrl);

    matches.sort(Comparator.comparing(Match::getDate));

        return matches;

}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
```

```
        String action = request.getParameter("action");
    Logic ctrl = new Logic();

    try {

        if ("standings".equals(action)) {

            int id = Integer.parseInt(request.getParameter("id"));
            Tournament tournament = ctrl.getTournamentById(id);

            if (tournament != null) {

                TreeMap<String, LinkedList<TeamStats>> tables = ctrl.calculateTables(id);

                request.setAttribute("tournament", tournament);
                request.setAttribute("tablesMap", tables);
                request.getRequestDispatcher("WEB-INF/Management/Standings.jsp").forward(request, response);

            } else {

                request.setAttribute("errorMessage", "El torneo solicitado no existe");
                request.getRequestDispatcher("WEB-INF/ErrorMessage.jsp").forward(request, response);

              }

        } else if ("add".equals(action)) {

            LinkedList<Association> associations = ctrl.getAllAssociations();

if (associations.size() > 0) {
```

```
                associations.sort(Comparator.comparing(Association::getName));
                request.setAttribute("associationsList", associations);

                HashMap<Integer, LinkedList<Club>> clubsMap = new HashMap<>();

                for (Association a : associations) {

            LinkedList<Club> associationClubs = ctrl.getClubsByAssociationId(a.getId());
            clubsMap.put(a.getId(), associationClubs);

            }

                if (clubsMap.size() > 0) {

                    request.setAttribute("clubsMap", clubsMap);
                    request.getRequestDispatcher("WEB-INF/Add/AddTournament.jsp").forward(request, response);

                } else {

                    request.setAttribute("errorMessage", "Debe agregar clubes primero");
                    request.getRequestDispatcher("WEB-INF/ErrorMessage.jsp").forward(request, response);

                }

} else {

                request.setAttribute("errorMessage", "Debés agregar una asociación primero");
                request.getRequestDispatcher("WEB-INF/ErrorMessage.jsp").forward(request, response);

            }
```

```
        } else {

    LinkedList<Tournament> tournaments = ctrl.getAllTournaments();

    for (Tournament t : tournaments) {

            ctrl.analyzeTournamentStatus(t);

    }

    request.setAttribute("tournamentsList", tournaments);

    LinkedList<Association> associations = ctrl.getAllAssociations();
request.setAttribute("associationsList", associations);

request.getRequestDispatcher("/WEB-INF/Management/TournamentManagement.jsp").forward(request, response);

        }

     } catch (SQLException e) {

            request.setAttribute("errorMessage", "Error al conectarse a la base de datos");
        request.getRequestDispatcher("WEB-INF/ErrorMessage.jsp").forward(request, response);

     }

}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

            request.setCharacterEncoding("UTF-8");
```

```
        response.setCharacterEncoding("UTF-8");

            String action = request.getParameter("action");

Logic ctrl = new Logic();

try {

    if ("add".equals(action)) {

    Tournament tournament = buildTournamentFromRequest(request, action, ctrl);

    if (checkStartDate(tournament.getStartDate())) {

            LocalDate realStartDate = adjustStartDateToFriday(tournament.getStartDate());
            tournament.setStartDate(realStartDate);

                                    LinkedList<Match> fixture = new LinkedList<>();

                                    LinkedList<Club> clubs = new LinkedList<>();
                                    String[] selectedClubs = request.getParameterValues("selectedClubs");

                                    if (selectedClubs == null || selectedClubs.length < 2) {

                                            throw new SQLException("Todos los torneos requieren de al menos 2 equipos.");

                                    }

                                    if (selectedClubs != null) {

                                            for (String clubIdStr : selectedClubs) {
```

```
                    try {

                        int clubId = Integer.parseInt(clubIdStr);
                        Club club = ctrl.getClubById(clubId);
                        clubs.add(club);

                    } catch (NumberFormatException e) {

                        System.err.println("ID de posición inválido: " + clubIdStr);

                    }

                }

            }

            fixture = drawTournamentMatchdays(clubs, tournament, ctrl);

            TournamentFormat format = tournament.getFormat();

            LocalDate lastMatchDate = fixture.getLast().getDate().toLocalDate();
            LocalDate endDate = lastMatchDate;

            if (format == TournamentFormat.WORLD_CUP) {

                endDate = lastMatchDate.plusWeeks(4);

            } else if (format == TournamentFormat.ZONAL_ELIMINATION) {

                if (clubs.size() >= 24) {
```

```
                                        endDate = lastMatchDate.plusWeeks(4);
                                    } else {
                                endDate = lastMatchDate.plusWeeks(3);
                                    }
                                }
                                tournament.setEndDate(endDate);
                                ctrl.addTournament(tournament);
                                for (Match match : fixture) {
                                    match.setTournament(tournament);
                        ctrl.addMatch(match);
                                }
                } else {
                                request.setAttribute("errorMessage", "Error en las fechas introducidas (el torneo debe empezar a partir de hoy y
durar, al menos, 4 meses)");
                                request.getRequestDispatcher("WEB-INF/ErrorMessage.jsp").forward(request, response);
                                return;
                }
```

```
} else if ("generatePlayoffs".equals(action)) {

                int id = Integer.parseInt(request.getParameter("id"));

                ctrl.generatePlayoffs(id);

            response.sendRedirect("actionmatch?tournamentId=" + id);
            return;

} else if ("generateNextStage".equals(action)) {

    int id = Integer.parseInt(request.getParameter("id"));

    ctrl.generateNextStage(id);

    response.sendRedirect("actionmatch?tournamentId=" + id);
    return;

} else if ("finishTournament".equals(action)) {

    int id = Integer.parseInt(request.getParameter("id"));

    ctrl.finishTournament(id);

    response.sendRedirect("actionmatch?tournamentId=" + id);
    return;

} else if ("delete".equals(action)) {

 Integer id = Integer.parseInt(request.getParameter("id"));
```

```
            ctrl.deleteTournament(id);

           }

           response.sendRedirect("actiontournament");

        } catch(SQLException e) {

            request.setAttribute("errorMessage", "Error al conectarse a la base de datos");
                request.getRequestDispatcher("WEB-INF/ErrorMessage.jsp").forward(request, response);

        }

            }

}
```

Logic.java

```
package logic;

import data.*;
import entities.*;
import enums.*;
import enums.PersonRole;
import utils.EmailSender;

import java.sql.SQLException;
import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.temporal.TemporalAdjusters;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;

public class Logic {

    private DataPerson dpe;
    private DataPlayer dpl;
    private DataCoach dc;
    private DataPresident dpr;
```

```
private DataStadium dst;
private DataClub dcl;
private DataAssociation das;
private DataTournament dto;
private DataContract dco;
private DataPosition dpo;
private DataPlayerPosition dpp;
private DataMatch dm;
private DataNationality dn;
private DataUser du;

public Logic() {

    dpe = new DataPerson();
    dpl = new DataPlayer();
    dc = new DataCoach();
    dpr = new DataPresident();
    dst = new DataStadium();
    dcl = new DataClub();
    das = new DataAssociation();
    dto = new DataTournament();
    dco = new DataContract();
    dpo = new DataPosition();
    dpp = new DataPlayerPosition();
    dm = new DataMatch();
    dn = new DataNationality();
    du = new DataUser();

}
```

```
public LinkedList<Tournament> getAllTournaments() throws SQLException {
        return dto.getAll();
    }
    public Tournament getTournamentById(int id) throws SQLException {
        return dto.getById(id);
    }
    public LinkedList<Tournament> getTournamentsByAssociationId(int id) throws SQLException {
        return dto.getByAssociationId(id);
    }
    public LinkedList<Tournament> getTournamentsByClubId(int id) throws SQLException {
        return dto.getByClubId(id);
    }
    public LinkedList<Tournament> getTournamentByName(String name) throws SQLException {
        return dto.getByName(name);
    }
    public void addTournament(Tournament t) throws SQLException {
```

```
        dto.add(t);
    }
    public void updateTournament(Tournament t) throws SQLException {
        dto.update(t);
    }
    public void deleteTournament(int id) throws SQLException {
        dto.delete(id);
    }
}
```

DataTournament.java

```
package data;

import entities.*;
import enums.TournamentFormat;

import java.sql.*;
import java.time.LocalDate;
import java.util.LinkedList;

public class DataTournament {

    private static final String BASE_QUERY =
            "SELECT "
            + "    t.id AS tournament_id, "
            + "    t.name AS tournament_name, "
            + "    t.start_date AS tournament_start_date, "
            + "    t.end_date AS tournament_end_date, "
            + "    t.format AS tournament_format, "
            + "    t.season AS tournament_season, "
            + "            t.finished AS tournament_finished, "
            + "    t.id_association, "
            + "    a.id AS association_id, "
            + "    a.name AS association_name, "
            + "    a.creation_date AS association_creation_date, "
            + "    SUM(CASE WHEN m.stage = 'GROUP_STAGE' AND (m.home_goals IS NULL OR m.away_goals IS NULL) THEN 1 ELSE 0 END) as
pending_group_matches, "
            + "    SUM(CASE WHEN m.stage = 'GROUP_STAGE' THEN 1 ELSE 0 END) as total_group_matches, "
            + "    SUM(CASE WHEN m.stage != 'GROUP_STAGE' THEN 1 ELSE 0 END) as playoff_matches_count, "
            + "    SUM(CASE WHEN m.home_goals IS NOT NULL THEN 1 ELSE 0 END) as matches_played_count "
```

```
        + "FROM tournament t "
        + "INNER JOIN association a ON t.id_association = a.id "
        + "LEFT JOIN `match` m ON t.id = m.id_tournament ";

private static final String GROUP_BY_CLAUSE =
        " GROUP BY t.id, t.name, t.start_date, t.end_date, t.format, t.season, "
        + "        t.finished, t.id_association, a.id, a.name, a.creation_date ";

public LinkedList<Tournament> getAll() throws SQLException {

    PreparedStatement stmt = null;
    ResultSet rs = null;
    LinkedList<Tournament> tournaments = new LinkedList<>();

    try {

        stmt = DbConnector.getInstance().getConn().prepareStatement(BASE_QUERY + GROUP_BY_CLAUSE + " ORDER BY t.start_date, t.end_date");
        rs = stmt.executeQuery();

        if (rs != null) {

            while (rs.next()) {

                Tournament tournament = mapFullTournament(rs);
                tournaments.add(tournament);

            }

        }

    } catch (SQLException e) {
```

```
            e.printStackTrace();
            throw new SQLException("No se pudo conectar a la base de datos.", e);

        } finally {

            closeResources(rs, stmt);

        }

        return tournaments;

    }

    public Tournament getById(int id) throws SQLException {

        Tournament tournament = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {

            stmt = DbConnector.getInstance().getConn().prepareStatement(
             BASE_QUERY + "WHERE t.id = ?" + GROUP_BY_CLAUSE
            );
            stmt.setInt(1, id);
            rs = stmt.executeQuery();

            if (rs != null && rs.next()) {

                tournament = mapFullTournament(rs);
```

```
            }

        } catch (SQLException e) {

            e.printStackTrace();
            throw new SQLException("No se pudo conectar a la base de datos.", e);

        } finally {

            closeResources(rs, stmt);

        }

        return tournament;

    }

    public LinkedList<Tournament> getByClubId(int id) throws SQLException {

        LinkedList<Tournament> tournaments = new LinkedList<>();
        PreparedStatement stmt = null;
        ResultSet rs = null;

        try {

            stmt = DbConnector.getInstance().getConn().prepareStatement(
             BASE_QUERY + "WHERE m.id_home = ? or m.id_away = ?" + GROUP_BY_CLAUSE
            );
            stmt.setInt(1, id);
            stmt.setInt(2, id);
```

```
            rs = stmt.executeQuery();

            if (rs != null) {

                while (rs.next()) {

                    Tournament tournament = mapFullTournament(rs);
                    tournaments.add(tournament);

                }
            }

        } catch (SQLException e) {

            e.printStackTrace();
            throw new SQLException("No se pudo conectar a la base de datos.", e);

        } finally {

            closeResources(rs, stmt);

        }

        return tournaments;

    }

    public LinkedList<Tournament> getByAssociationId(int id) throws SQLException {

        LinkedList<Tournament> tournaments = new LinkedList<>();
        PreparedStatement stmt = null;
```

```
ResultSet rs = null;

try {

    stmt = DbConnector.getInstance().getConn().prepareStatement(
     BASE_QUERY + "WHERE a.id = ?" + GROUP_BY_CLAUSE
    );
    stmt.setInt(1, id);
    rs = stmt.executeQuery();

    if (rs != null) {

        while (rs.next()) {

            Tournament tournament = mapFullTournament(rs);
            tournaments.add(tournament);

        }
    }

} catch (SQLException e) {

    e.printStackTrace();
    throw new SQLException("No se pudo conectar a la base de datos.", e);

} finally {

    closeResources(rs, stmt);

}
```

```
    return tournaments;

}

public LinkedList<Tournament> getByName(String name) throws SQLException {

    PreparedStatement stmt = null;
    ResultSet rs = null;
    LinkedList<Tournament> tournaments = new LinkedList<>();

    try {

        stmt = DbConnector.getInstance().getConn().prepareStatement(
         BASE_QUERY + " WHERE t.name = ?" + GROUP_BY_CLAUSE
        );
        stmt.setString(1, name);
        rs = stmt.executeQuery();

        if (rs != null) {

            while (rs.next()) {

                Tournament tournament = mapFullTournament(rs);
                tournaments.add(tournament);

            }
        }

    } catch (SQLException e) {

        e.printStackTrace();
```

```
            throw new SQLException("No se pudo conectar a la base de datos.", e);

        } finally {

            closeResources(rs, stmt);

        }

        return tournaments;

    }

    public void add(Tournament t) throws SQLException {

        PreparedStatement stmt = null;
        ResultSet keyResultSet = null;

        try {

            stmt = DbConnector.getInstance().getConn().prepareStatement(
                "INSERT INTO tournament (name, start_date, end_date, format, season, id_association) "
                + "VALUES (?, ?, ?, ?, ?, ?)",
                PreparedStatement.RETURN_GENERATED_KEYS
            );
            stmt.setString(1, t.getName());
            stmt.setObject(2, t.getStartDate());
            stmt.setObject(3, t.getEndDate());
            stmt.setString(4, t.getFormat().name());
            stmt.setString(5, t.getSeason());
            stmt.setInt(6, t.getAssociation().getId());
            stmt.executeUpdate();
```

```
            keyResultSet = stmt.getGeneratedKeys();
            if (keyResultSet != null && keyResultSet.next()) {

                t.setId(keyResultSet.getInt(1));

            }

        } catch (SQLException e) {

            e.printStackTrace();
            throw new SQLException("No se pudo conectar a la base de datos.", e);

        } finally {

            try {

                if (keyResultSet != null) keyResultSet.close();

            } catch (SQLException e) {

                e.printStackTrace();

            }

            closeResources(null, stmt);

        }

    }
```

```
public void update(Tournament t) throws SQLException {

    PreparedStatement stmt = null;

    try {

        stmt = DbConnector.getInstance().getConn().prepareStatement(
         "UPDATE tournament "
            + "SET name = ?, start_date = ?, end_date = ?, format = ?, season = ?, id_association = ? "
            + "WHERE id = ?"
        );
        stmt.setString(1, t.getName());
        stmt.setObject(2, t.getStartDate());
        stmt.setObject(3, t.getEndDate());
        stmt.setString(4, t.getFormat().name());
        stmt.setString(5, t.getSeason());
        stmt.setInt(6, t.getAssociation().getId());
        stmt.setInt(7, t.getId());
        stmt.executeUpdate();

    } catch (SQLException e) {

        e.printStackTrace();
        throw new SQLException("No se pudo conectar a la base de datos.", e);

    } finally {

        closeResources(null, stmt);

    }
```

```
}

public void finishTournament(int id) throws SQLException {

        PreparedStatement stmt = null;

        try {

                stmt = DbConnector.getInstance().getConn().prepareStatement(
          "UPDATE tournament SET finished = 1 WHERE id = ?"
        );
        stmt.setInt(1, id);

        stmt.executeUpdate();

        } catch (SQLException e) {

                e.printStackTrace();
        throw new SQLException("No se pudo conectar a la base de datos.", e);

    } finally {

        closeResources(null, stmt);

    }
}

public void delete(int id) throws SQLException {

    PreparedStatement stmt = null;
```

```
    try {

        stmt = DbConnector.getInstance().getConn().prepareStatement(
         "DELETE FROM tournament WHERE id = ?"
        );
        stmt.setInt(1, id);
        stmt.executeUpdate();

    } catch (SQLException e) {

        e.printStackTrace();
        throw new SQLException("No se pudo conectar a la base de datos.", e);

    } finally {

        closeResources(null, stmt);

    }

}

private Tournament mapFullTournament(ResultSet rs) throws SQLException {

    Tournament tournament = new Tournament();
    tournament.setId(rs.getInt("tournament_id"));
    tournament.setName(rs.getString("tournament_name"));
    tournament.setStartDate(rs.getObject("tournament_start_date", LocalDate.class));
    tournament.setEndDate(rs.getObject("tournament_end_date", LocalDate.class));
    tournament.setFormat(TournamentFormat.valueOf(rs.getString("tournament_format")));
    tournament.setSeason(rs.getString("tournament_season"));
    tournament.setFinished(rs.getBoolean("tournament_finished"));;
```

```
        int pending = rs.getInt("pending_group_matches");
        int totalGroup = rs.getInt("total_group_matches");
        int playoffCount = rs.getInt("playoff_matches_count");
        int playedCount = rs.getInt("matches_played_count");

        boolean allGroupMatchesPlayed = (totalGroup > 0) && (pending == 0);
        tournament.setAllGroupMatchesPlayed(allGroupMatchesPlayed);

        tournament.setPlayoffsAlreadyGenerated(playoffCount > 0);
        tournament.setHasMatchesPlayed(playedCount > 0);

        Association association = new Association();
        association.setId(rs.getInt("association_id"));
        association.setName(rs.getString("association_name"));
        association.setCreationDate(rs.getObject("association_creation_date", LocalDate.class));

        tournament.setAssociation(association);

        return tournament;

    }

    private void closeResources(ResultSet rs, PreparedStatement stmt) {

        try {

            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            DbConnector.getInstance().releaseConn();
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```