



Ingeniería Informática

Obligatorio

Asignatura: Bases de Datos II

Docente: Federico Gómez

Juan Cosentino

Santiago Ferraro

Bianca Luzzatto

Índice

Introducción.....	3
Identificación de enunciados.....	4
Matriz de entidades y relaciones.....	5
Metodología definida.....	7
Construcción del MER.....	8
Identificación de requisitos.....	9
Diseño de la aplicación inicial.....	10
Login.....	10
Sign up.....	11
Fixture.....	11
Hacer predicción.....	12
Actualizar resultado.....	13
Leaderboard.....	13
Predicciones.....	14
Puntajes de alumnos.....	14
Aplicaciones similares.....	15
Montevideo Portal.....	15
Tienda Ingresa.....	16
Penka-Copa America & Euro 2024.....	17
Diseño de la aplicación final.....	17
Login.....	18
Sign up.....	18
Home.....	19
Fixture.....	20
Predicciones.....	21
Apostar.....	22
Leaderboard.....	23
Fixture del administrador.....	23
Herramientas.....	24
Angular.....	24
Bootstrap.....	25
Docker.....	25
MySQL con InnoDB.....	25
Postman.....	26
NetBeans.....	26
DataGrip.....	26
Java.....	27
Spring Boot.....	27
JDBC.....	27
Patrones de diseño.....	27
Patrón repository.....	27
Patrón DTO.....	28

Diagramas.....	28
Diagrama de clases.....	28
Diagrama de secuencia.....	29
Desarrollo del código.....	30
Frontend.....	30
Componentes.....	31
Diseño de componentes.....	32
Servicios.....	32
Módulos.....	33
Interfaces.....	34
Login.....	34
Sign up.....	35
Home.....	35
Fixture.....	35
Predicciones.....	36
Apostar.....	37
Leaderboard.....	37
Fixture del administrador.....	38
Backend.....	38
Borrado lógico.....	38
Clases DTO.....	39
Clases repositorios.....	41
Clases controladores.....	45
Conexiones a base de datos.....	47
Consultas SQL.....	48
Consultas SELECT.....	48
Consultas UPDATE.....	49
Consultas INSERT.....	51
Conclusiones.....	53
Bibliografía.....	53
Anexo.....	56
Enlace al proyecto.....	56

Introducción

El trabajo obligatorio consiste en realizar una aplicación de fixture para la Copa América 2024, la cual debe cumplir con ciertos requisitos y debe tener ciertas características. Dentro de estos requisitos, se pide crear una base de datos y administrarla según los contenidos del curso. Es por esto que se debe realizar una investigación acerca de las posibles tecnologías y herramientas, para encontrar las óptimas para los requerimientos de la letra.

Se planteó una idea inicial de aplicación con sus bocetos correspondientes, los cuales serán detallados a continuación, y luego se compararán con los resultados finales. Durante el proceso de desarrollo de la aplicación, inevitablemente aparecen imprevistos que deben ser resueltos. Es por esto que los documentos iniciales cambiaron, siendo los finales bastante más detallados y completos.

La aplicación debe permitir el ingreso de diferentes tipos de usuario, los cuales utilizarán de una forma u otra la aplicación, accediendo a funcionalidades específicas.

Identificación de enunciados

A partir de la letra, se identificaron los enunciados para modelar en el MER, buscando posibles atributos en las entidades. La identificación de enunciados inicial fue la siguiente:

Puntajes (**tipo**, puntaje)

Enunciado Puntajes

Para ser más justo se pensó en una manera de premiar a los que acierten con los resultados más exactos. ¿Cómo se otorgarán los puntos? Se otorgarán 4 puntos por resultado exacto de un partido, y 2 puntos por resultado correcto. También se otorgarán 10 puntos por campeón y 5 por subcampeón, ambos datos se deberán cargar al momento de crear el usuario y no podrán modificarlo luego, por lo cual tendrán que estar muy seguros de lo que ingresan.

Usuarios(**documento**, nombre, apellido, celular, contraseña, correo)

Alumnos(carrera)

Administradores()

Enunciado Usuarios

Se contará con un registro de usuarios, así como con un usuario administrador para cargar los resultados de los partidos jugados. Vale recordar que el usuario administrador no podrá ser un alumno, ya que el mismo no podría participar de la penca.

Partidos(id_partido, equipo1, equipo2, fecha, gol_equipo1, gol_equipo2, etapa)

Enunciado Partidos

Cada día, antes del primer partido de la etapa, se le enviará una notificación a los usuarios participantes que deben ingresar sus jugadas, si no lo han hecho aún. Podrán ingresar sus

predicciones una hora antes de que comience el partido. Como hay alumnos que son muy aplicados y ya tiene todos los posibles resultados en su mente, pueden ingresar los resultados en cualquier momento y modificarlo también, siempre que se cumpla que no falte una hora para el partido.

Se podrá visualizar la lista de participantes con sus puntajes para cotejo de todos, así como el fixture del campeonato, el cual se actualizará en cada etapa. ¿Cómo funcionaría?

Predicciones(id_usuario, id_partido, predicción_equipo1, predicción_equipo2,
puntaje_obtenido)

Enunciado Predicciones

Empecemos por registrarnos e ingresar predicciones para los partidos (obviamente no podré ingresar más de una predicción por partido). Deberá otorgar a cada usuario la posibilidad de ver sus predicciones ingresadas y los puntos obtenidos en cada una (si ya se ha jugado el partido). Esta aplicación debería poder ser usada inicialmente por todos los alumnos de la UCU que se hayan registrado en la misma.

El Sistema debe tener en cuenta que a futuro puede interesar sacar estadísticas de como son los porcentajes de acierto de los alumnos teniendo en cuenta la Carrera que cursan.

Luego de modelar estas entidades, se encontraron que faltaban algunas que eran necesarias modelar. También se encontró que habían atributos que no eran necesarios representar, por lo que se ajustó posteriormente. A pesar de estas diferencias, la identificación de estos enunciados iniciales sirvió de base para modelar la aplicación.

Matriz de entidades y relaciones

La matriz inicial con las relaciones se veía de esta manera, previo a la adición de nuevas entidades y relaciones:

	Usuarios	Alumnos	Administradores	Partidos	Prediccion es	Puntajes
Usuarios						
Alumnos					hace	
Administradores						

Partidos					genera: puntaje_o btenido	
Prediccion es		la hacen		tiene: puntaje_o btenido		Es definido por
Puntajes					define	

Una vez que se realizaron cambios, la matriz pasó a ser la siguiente:

	Usuarios	Administradores	Alumnos	Carreras	Predicciones	Puntajes	Partidos	Países	Estadios	Etapas
Usuarios										
Administradores										
Alumnos				Cursa (N, N)	Hace (1, N)			Campeón (N, 1); Subcampeón (N, 1)		
Carreras			Cursada por (N, N)							
Predicciones			Hechas por (N, 1)			Definida por (N, 1)	Generada por (N, 1)			
Puntajes					Define (1, N)					
Partidos					Genera (1, N)			Local (N, 1); Visitante (N, 1)	Se juega (N, 1)	Pertenecce (N, 1)
Países			Campeón(1, N);				Local (N, 1); Visitante			

			Subcategoría (1, N)				e (N, 1)			
Estadios							Se juega (1, N)			
Etapas							Contiene (1, N)			

Metodología definida

La justificación de la elección de metodología inicialmente fue la siguiente:

Para crear el Modelo Entidad-Relación se utilizó una metodología descendente, dado que se partió del concepto abstracto de una penca con determinadas funcionalidades que se fue refinando y concretando hasta llegar a las entidades. La primera entidad modelada fue la de “Usuarios”, dado que era una entidad compleja con diversos atributos, que a su vez tiene una generalización con las entidades hijas “Administradores” y “Alumnos”. Luego siguió “Partidos”, otra entidad central que tenía varios atributos. Finalmente, en los enunciados se mencionan las predicciones que pueden hacer los usuarios, lo cual es otra entidad con sus propios atributos. Además, incluimos la entidad “Puntajes”, que definirá cuántos puntos obtendrá cada usuario de acuerdo a su predicción en cada partido, se decidió crear una entidad para que ante cualquier cambio en el sistema de puntuación, esto sea fácilmente actualizable. Una vez que se tuvieron estas entidades, se planificó de qué forma se relacionarían entre ellas.

Inicialmente se comenzó con la relación de que los alumnos realizan predicciones, ya que uno de los requisitos de la aplicación era este. Para realizar las predicciones era importante saber sobre qué partido se estaba prediciendo, con lo cual fue natural relacionar esta entidad con la de “Partidos”. También se debía obtener un puntaje por la predicción, por lo que se formó una relación con la tabla de “Puntajes”, que indica cuántos puntos se obtiene por predicción. La misma es débil de Predicción, ya que sin las predicciones no existirían puntajes.

No existe relación directa entre ninguno de los usuarios con los partidos, dado que la única forma de interacción entre ellos es a través de las predicciones. Por otra parte, la relación

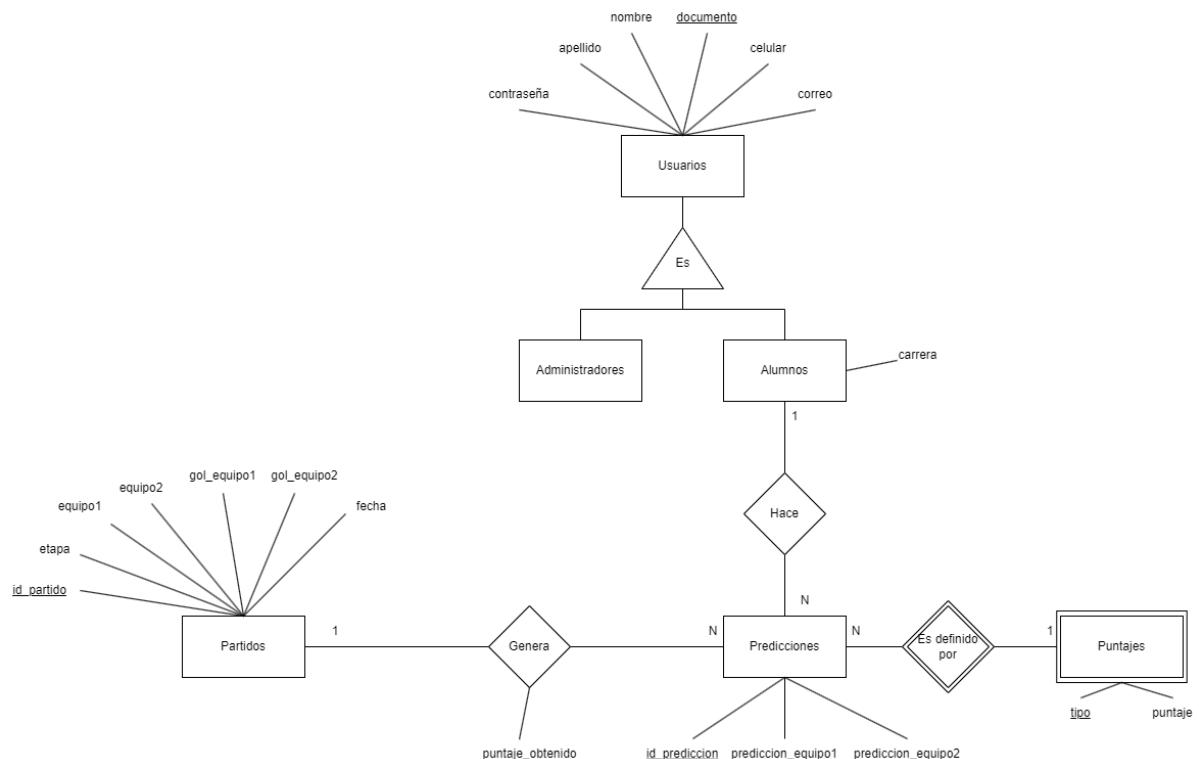
“Genera” cuenta con un atributo llamado “puntaje_obtenido”, donde se tienen los puntajes de cada predicción hecha. Este atributo está en la relación porque es propio de las predicciones para cada partido, además de que el valor de ese atributo se obtiene una vez que se finalizó el partido.

Luego de los refinamientos, es necesario aplicar algunos cambios a esta metodología. Inicialmente, la entidad “Puntajes” no es débil de “Predicciones”, dado que los puntajes van a existir independientemente de las predicciones, por ejemplo al definir el campeón y subcampeón al momento de registro, ya que al acertar en estos se obtiene un puntaje y no son predicciones. Luego, la relación “Genera” no debe contar con atributos de puntajes, dado que estos provienen de la relación entre “Predicciones” y “Puntajes”. En cuanto a entidades que faltan, es sabido que los partidos son jugados por países, dentro de un estadio, y en una fase de la Copa América. Estas entidades no están explícitamente indicadas en la letra, pero se podían deducir.

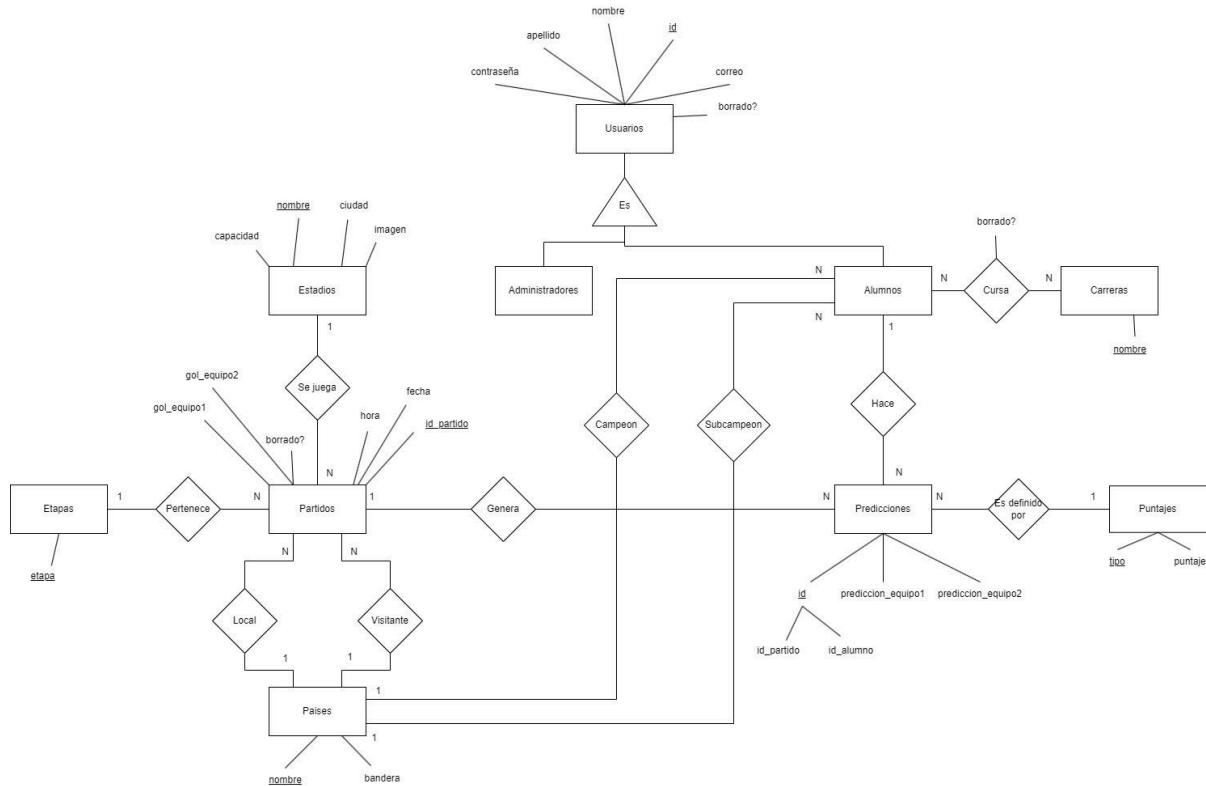
Construcción del MER

Para realizar la construcción del Modelo Entidad Relación, se deben tener en cuenta

El MER inicial se veía de la siguiente manera:



El MER luego de recibir ajustes:



Identificación de requisitos

- para los usuarios
 - poder registrarse
 - al momento de ingresar, se debe especificar el campeón y subcampeón y no se deben poder cambiar
 - hacer predicciones y modificarlas hasta una hora antes del partido
 - visualizar predicciones y puntajes
 - visualizar fixture
 - visualizar participantes y sus puntajes
- para los administradores
 - cargar los resultados de los partidos el día en que se juegan
- sistema
 - otorgar puntos por resultados de partidos
 - 4 por resultados exactos
 - 2 por resultados correctos
 - otorgar puntos por resultados de la Copa América

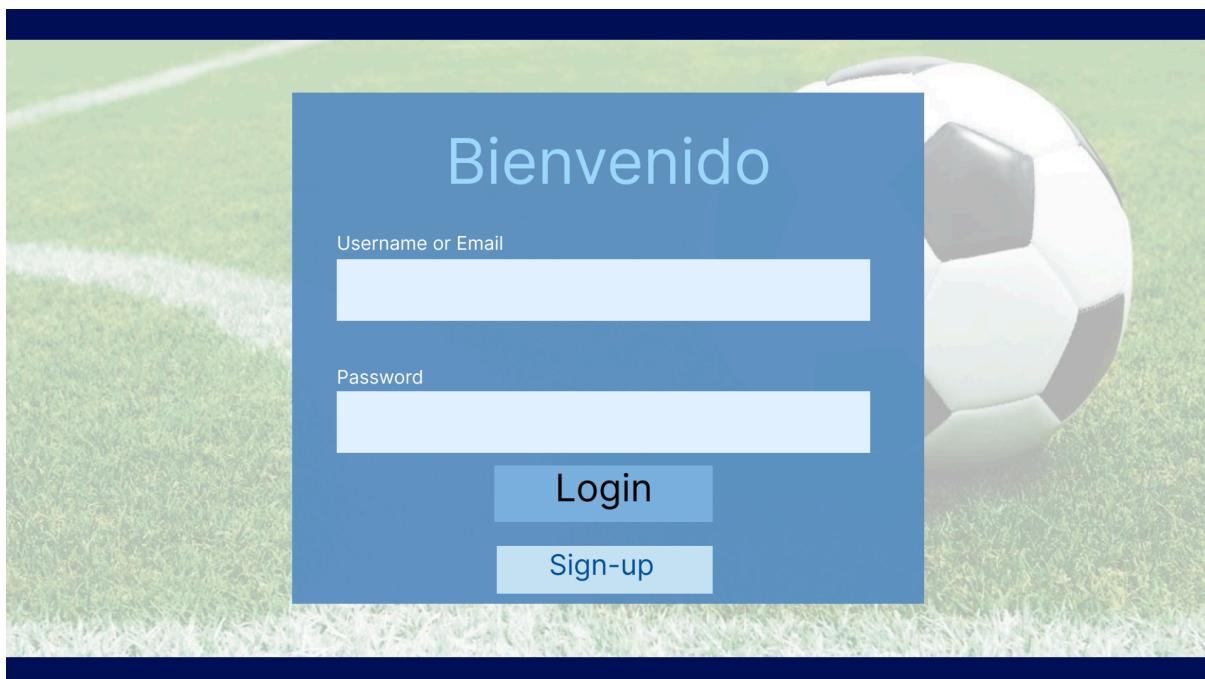
- 10 por campeón
- 5 por subcampeón
- enviar notificaciones de llenar predicción el día mismo del partido
- calcular porcentajes de acierto según las carreras de los alumnos

Diseño de la aplicación inicial

Para el diseño, creamos un prototipo en Figma. Se modelaron 8 pantallas de la aplicación que muestran la interfaz de algunas de las funcionalidades, las cuales sirvieron de base para modelar la aplicación y definir las funcionalidades dentro de cada pantalla de la misma.

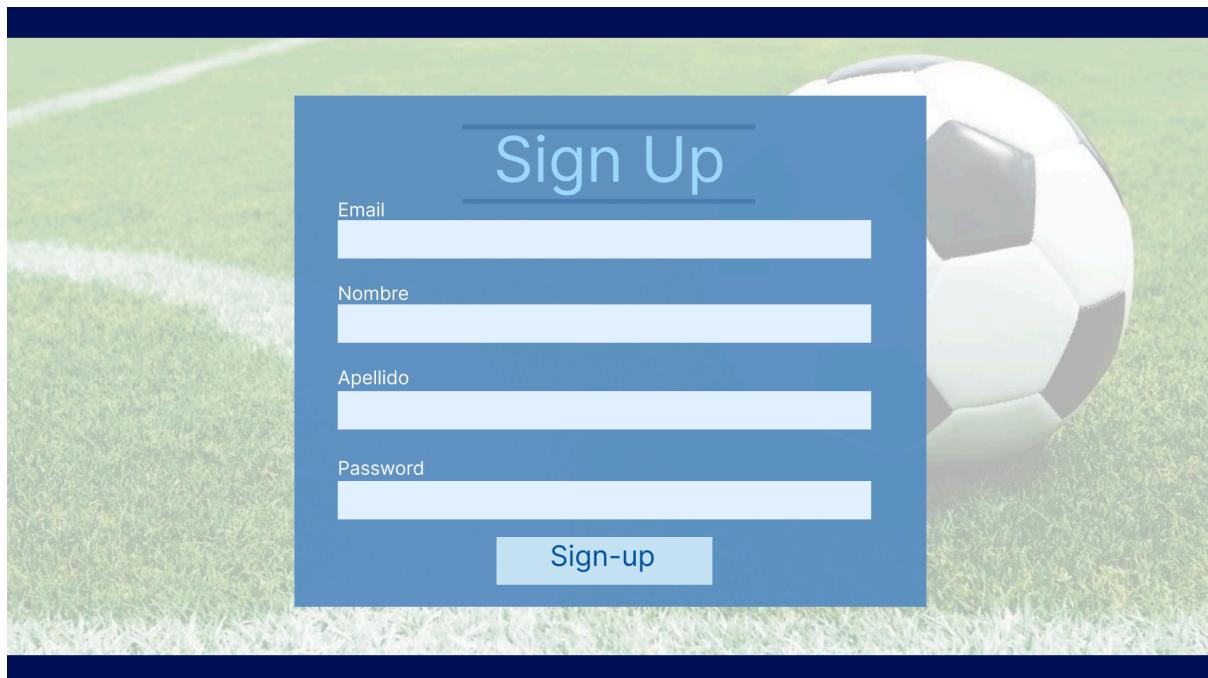
Login

El objetivo de esta pantalla es poder ingresar con un usuario al sistema. La aplicación cuenta con dos tipos de usuario: administradores y alumnos. Si bien comparten atributos, la principal diferencia radica en los usos que le puede dar cada tipo de usuario a la aplicación, ya que el administrador podrá actualizar la información de la aplicación, mientras que el alumno podrá realizar predicciones de los partidos y participar del torneo de la penca.



Sign up

La funcionalidad de registrarse permite a un usuario utilizar la aplicación si no tiene una cuenta. Esto estará disponible para los alumnos, ya que los administradores tendrán sus cuentas creadas y no será necesario que se registren. Se piden los datos iniciales de los alumnos, y luego se pide que ingresen una predicción acerca del país campeón y subcampeón, las cuales serán fijas.



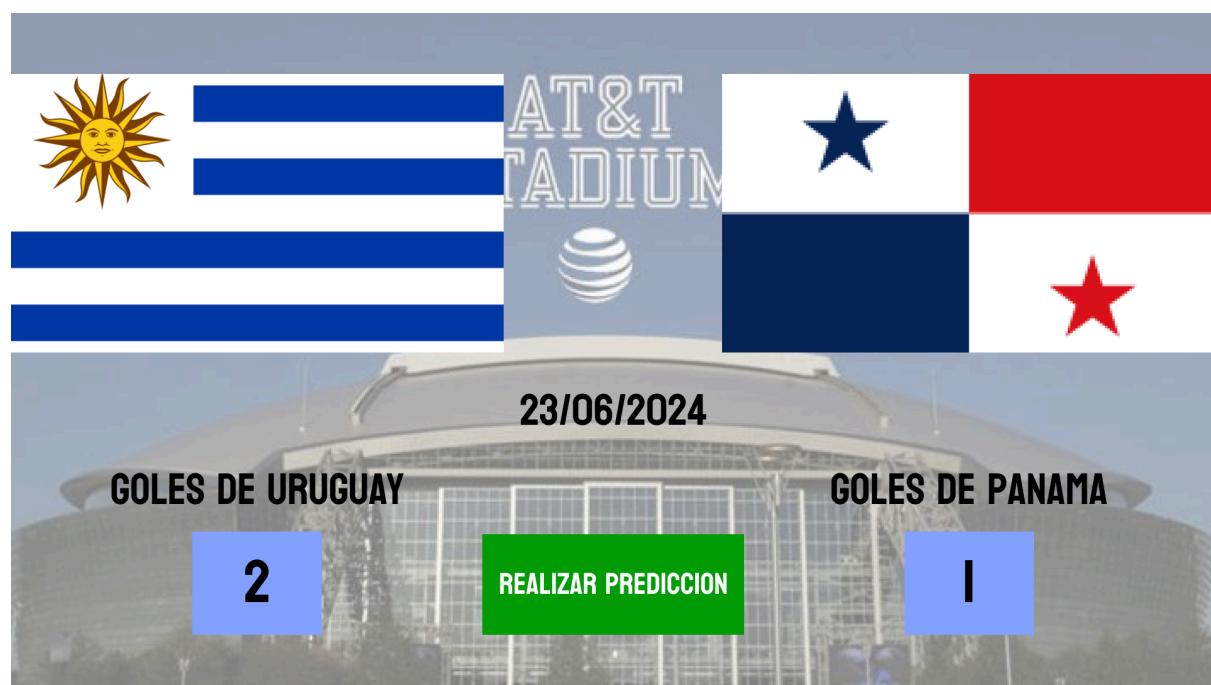
Fixture

En esta pantalla se podrán ver los partidos de la Copa América, así como también el avance de la misma. Aquí se diferencia entre alumno y administrador, dado que el alumno podrá ingresar predicciones de los partidos, mientras que el administrador podrá actualizar los resultados de los partidos.



Hacer predicción

Una vez que se elige un partido de la pantalla del fixture (al hacer click en alguno de los partidos) se muestra esta pantalla, donde se muestra qué países juegan, el estadio y la fecha. Aquí se puede ingresar el resultado que se predice, el cual podrá ser modificado hasta una hora antes de que comience el partido.



Actualizar resultado

Esta pantalla es para los administradores. Similar a la de realizar predicciones, aquí se puede actualizar el resultado de un partido, el cual será desplegado luego en las aplicaciones de los alumnos.

The image shows a match preview for a football game between Uruguay and Panama. The Uruguay flag is on the left, featuring a yellow sun with a face and twelve rays, set against five horizontal blue stripes. The Panama flag is on the right, featuring a white top half with a blue star, a red middle section, and a white bottom section with a red star. The date '23/06/2024' is centered between the flags. Below the flags, the text 'GOLES DE URUGUAY' is next to a blue box containing the number '2'. To the right is a green button labeled 'GUARDAR RESULTADO'. On the far right, the text 'GOLES DE PANAMA' is next to a blue box containing the number '1'.

Leaderboard

La aplicación permite ver los puntajes obtenidos por los alumnos según sus predicciones, ordenados de mayor puntaje a menor, mostrando los datos con los que se registró.

Predicciones

Los alumnos podrán ver todas las predicciones que tienen hechas en la aplicación, se muestran todos los partidos en donde ingresaron algún puntaje.



Puntajes de alumnos

También pueden ver los puntos que obtuvieron según las predicciones. Por un resultado exacto se otorgan cuatro puntos, un resultado correcto (acertar en que es empate o quién gana pero no exactamente el resultado) otorga dos, mientras que en el resto de casos se otorgan 0 puntos.



Aplicaciones similares

El modelo inicial fue planteado por el equipo previo a investigar otras aplicaciones similares. Al momento de construir la aplicación, se investigaron características y funcionalidades de otras aplicaciones del mercado para identificar aquellos puntos a mejorar, así como también evitar crear una aplicación que fuera una copia de otra.

Montevideo Portal

La siguiente penca se encuentra en la página de Montevideo Portal:

Copa América 2024

Ver ranking general 0 pts.

Grupos - Fecha 1	Grupos - Fecha 2	Grupos - Fecha 3
20/06/24 21:00 Argentina - vs Canadá	Pronóstico: Sin pronóstico	
21/06/24		PREMIO PARA EL GANADOR DE LA PENCA Copa América 2024 - USA

Guardar pronóstico

Dentro de esta penca aparecen todos los partidos sobre los cuales se puede predecir, ordenados en una lista, con un botón al final para guardar los datos. Este formato resulta cómodo para observar todos los partidos que van a ocurrir ya que no es necesario entrar uno por uno para ver la información que ingresó el usuario.

Tienda Ingresa

Esta penca es de Tienda Ingresa:

The top screenshot displays a ranking page with the following data:

Ranking	Jugador	Puntos
1	VERONICA S.	28 PUNTOS
2	ROMINA C.	28 PUNTOS
3	MARCOS A.	28 PUNTOS
4	GABRIEL V.	28 PUNTOS

The bottom screenshot displays a page for a match between Colombia (COL) and Paraguay (PRY) on June 24, with the following details:

LUNES 24 DE JUNIO

FASE DE GRUPOS – GRUPO D
19:00, NRG STADIUM, HOUSTON

COL VS PRY

PRONOSTICAR

Similar a la anterior, aparecen todos los partidos y se puede ingresar una predicción de cómo terminarán. Cuenta con una pestaña de ranking donde se pueden ver los puntajes obtenidos por cada participante, ordenados de mayor a menor.

Penka-Copa America & Euro 2024

Esta aplicación se encuentra disponible en el Google Play Store:

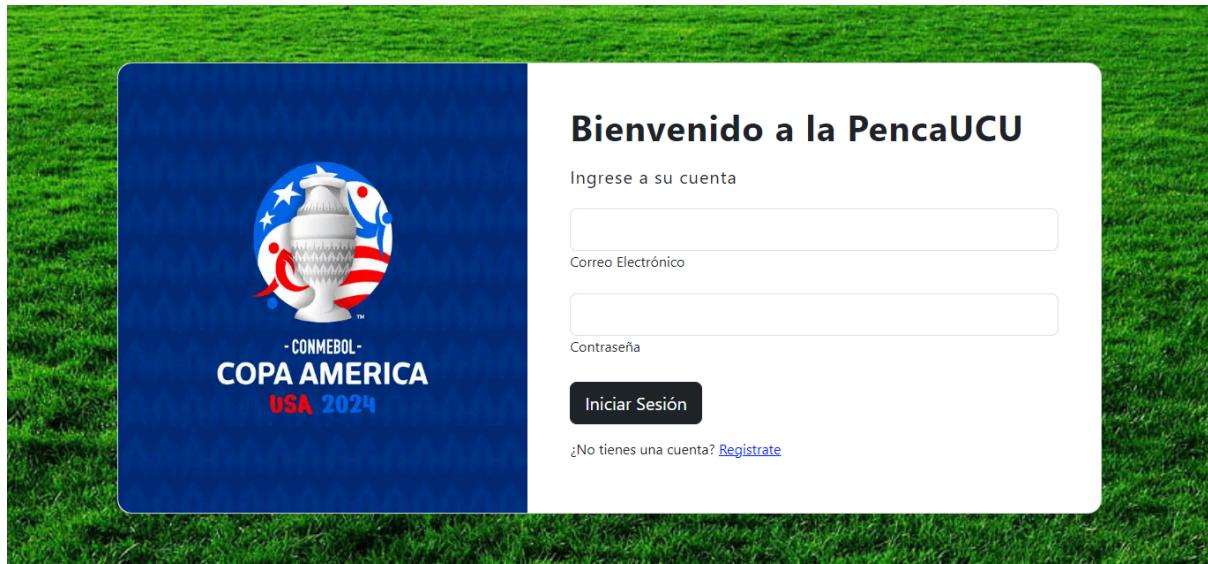


Se muestra una interfaz sencilla que permite ingresar las predicciones fácilmente y de forma rápida, y cuenta con características similares a las anteriores.

Diseño de la aplicación final

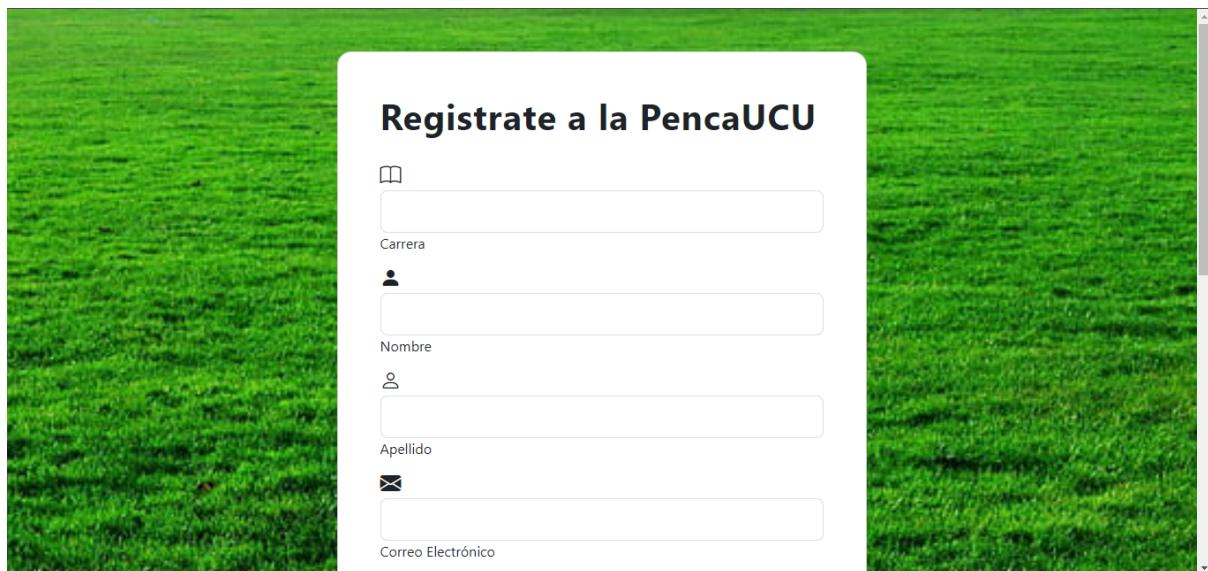
En base a los bocetos iniciales planteados y a las otras aplicaciones, se creó la aplicación final. Presenta algunos cambios en el diseño respecto a lo planteado inicialmente, enfocados a la facilidad de uso de la aplicación y también para mejorar la experiencia de usuario.

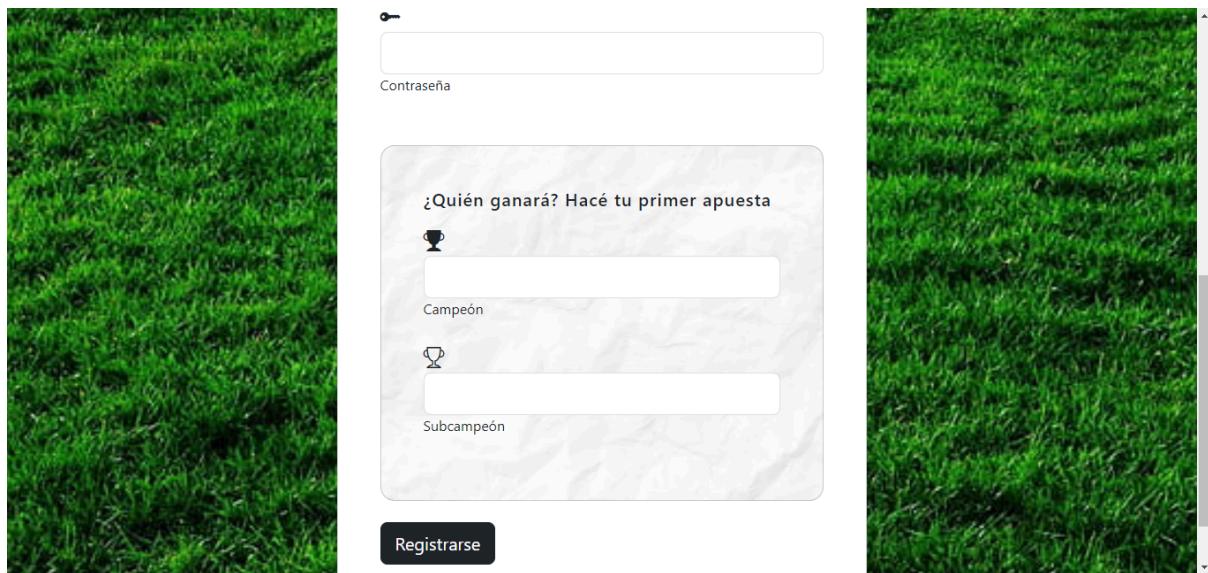
Login



Sign up

En el sign up se agregaron los campos de campeón y subcampeón, que eran requisito del problema y no estaban en los bocetos iniciales.



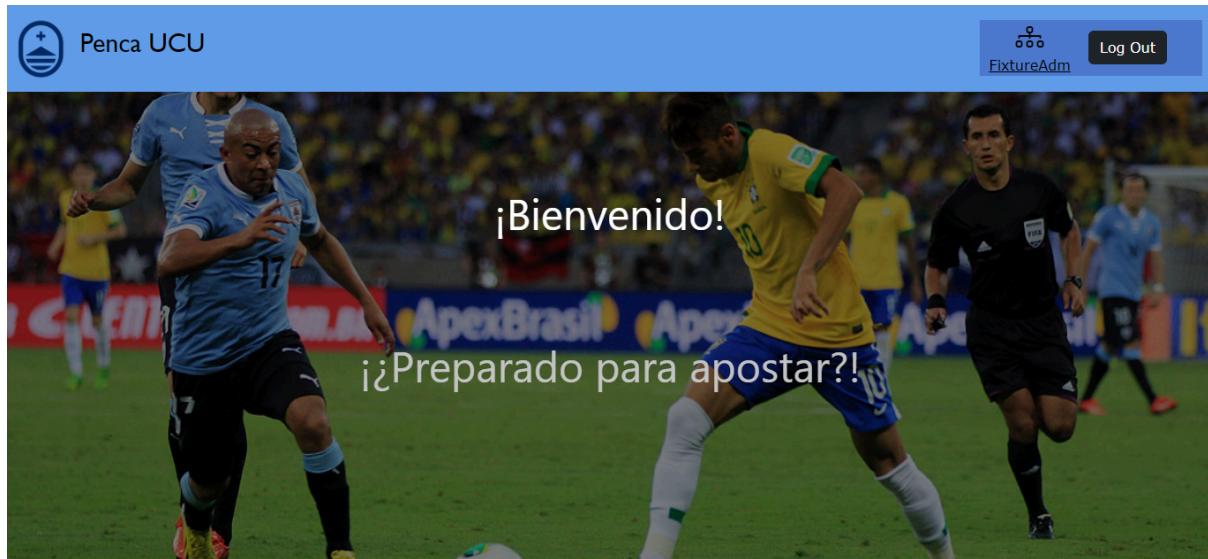


Home

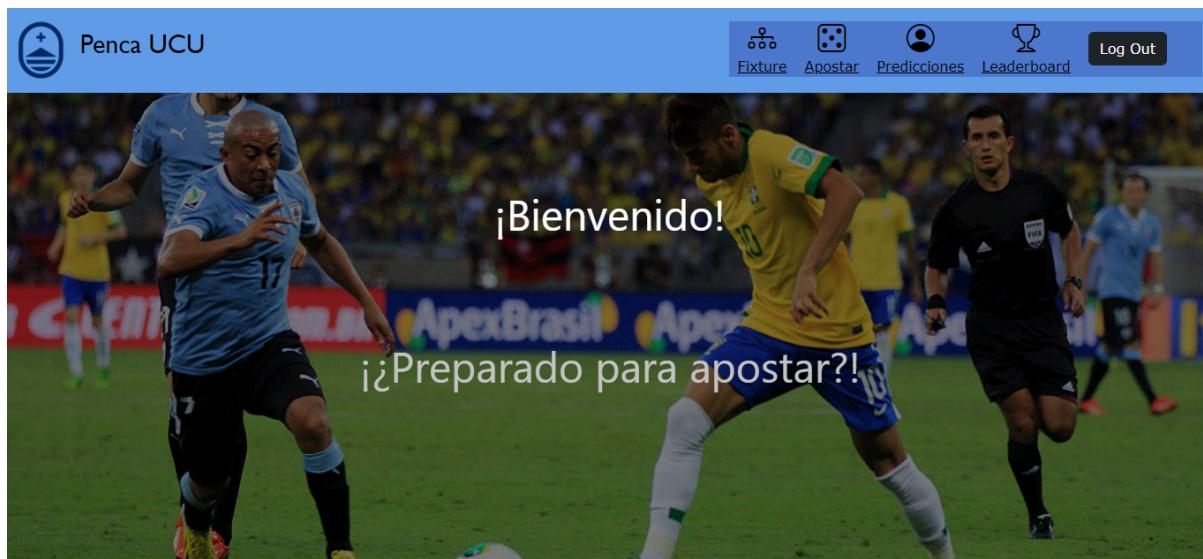
Al iniciar sesión, lo primero que aparece es el Home del sitio, el cual le da la bienvenida al usuario a la aplicación.

Tanto para el alumno como administrador es el mismo Home, pero cambia la barra de navegación.

Para el administrador:



Para el alumno:



Fixture

En esta pantalla se muestra el fixture de partidos, que será el mismo que el administrador utilizará para actualizar la información de los partidos. Aquí aparecen los resultados de los partidos, y en caso de que no hayan ocurrido se mostrarán con 0 goles. En el caso de las fases posteriores a la fase de grupos, dado que todavía no está definido qué países competirán, aparece como vacío.

A screenshot of the fixture table. The top navigation bar includes the Penca UCU logo, "Fixture", "Apostar", "Predicciones", "Leaderboard", and "Log Out" buttons. Below this, the word "Fixture:" is displayed above a table. The table has four columns representing groups: "Grupo A", "Cuartos", "Semifinal", and "Final". It also has four rows representing knockout stages: "Cuartos", "Semifinal", "3er y 4to puesto", and "Cuartos". Each row contains two cells for home and away teams, both labeled "Vacío" and "0 - 0". The background features a large CONMEBOL trophy and the text "CONMEBOL-AMERICA". Buttons for "Grupo C" and "Grupo D" are located at the bottom left and right respectively.

Al hacer click en alguno de los botones de grupos de los extremos, se abre una ventana que muestra los partidos de ese grupo:

The screenshot shows the 'Fixture' section of the Penca UCU app. It displays four groups of matches:

- Group A:** Argentina vs Peru (0-0), Peru vs Chile (0-0), Chile vs Argentina (0-0).
- Group B:** Canada vs Chile (0-0), Chile vs Argentina (0-0).
- Group C:** Vacío vs Cuadros (0-0), Cuadros vs Vacío (0-0).
- Group D:** Vacío vs Artos (0-0), Artos vs Vacío (0-0).

Each match row includes the national flags of the teams and their names.

Estos partidos a su vez pueden ser modificados, tanto los equipos como los goles. No hay partidos de fases de grupo vacíos, ya que estos vienen definidos previo al comienzo de la penca.

Predicciones

En esta pantalla se pueden ver aquellos partidos a los cuales ya se les ingresó una predicción, cada una de ellas con sus respectivos goles y puntos obtenidos. También se muestra el puntaje total que tiene el usuario hasta el momento, y se le permite actualizar las predicciones existentes siguiendo la misma regla de que falte más de una hora para el partido.

The screenshot shows the 'Predicciones' section for the Argentina vs Canada match. The interface includes:

- Puntaje Total:** 0 (with five yellow stars)
- Goles Argentina:** 0 (input field)
- Goles Canada:** 1 (input field)
- Puntos obtenidos:** 0
- Actualizar:** A blue button.
- Flags:** Argentina flag (blue with white sun) and Canada flag (red with white maple leaf).
- Team Names:** Argentina and Canada.

Apostar

Aquí se despliegan los partidos y se pueden ingresar predicciones acerca de los goles de cada país. Solamente permite guardar la predicción si falta más de una hora para el partido, para que el usuario conozca cuánto tiempo restante queda antes del partido se le añade la fecha del mismo en cada elemento. En este ejemplo, los primeros partidos ya ocurrieron y no se puede clickear el botón de “Apostar”:

The screenshot shows a section titled "Partidos para apostar:" (Matches to bet on). It displays three rows of information for soccer games:

- Argentina vs Canada**: Argentina 1 - 1 Canada. The date is 20/06/2024 00:00. The "Apostar" button is disabled.
- Peru vs Chile**: Peru 1 - 1 Chile. The date is 21/06/2024 00:00. The "Apostar" button is disabled.
- Peru vs Argentina**: Argentina 0 - 0 Peru. The date is 29/06/2024 00:00. The "Apostar" button is enabled.

Each row includes input fields for "Goles Argentina" and "Goles Canada" (or similar for other teams), the national flags of the teams, the current score, and the match date.

En caso de que se pueda apostar, se ve así:

The screenshot shows a section titled "Partidos para apostar:" (Matches to bet on). It displays three rows of information for soccer games, all of which are currently possible to bet on:

- Peru vs Argentina**: Argentina 0 - 0 Peru. The date is 29/06/2024 00:00. The "Apostar" button is enabled.
- Canada vs Chile**: Canada 1 - 1 Chile. The date is 29/06/2024 00:00. The "Apostar" button is enabled.
- Ecuador vs Venezuela**: Ecuador 0 - 0 Venezuela. The date is 29/06/2024 00:00. The "Apostar" button is enabled.

Each row includes input fields for "Goles Peru" and "Goles Argentina" (or similar for other teams), the national flags of the teams, the current score, and the match date.

Leaderboard

Aquí se pueden ver los puntajes del top 15 usuarios con mayor puntajes, ordenados de mayor a menor. Se muestra el nombre y apellido de cada usuario, para identificarlo.

The screenshot shows a blue header bar with the logo "Penca UCU" and navigation links for "Fixture", "Apostar", "Predicciones", "Leaderboard", and "Log Out". Below the header is a banner with the word "Leaderboard". A table follows, with columns: Puesto (Position), Nombre (Name), Apellido (Last Name), and Puntaje (Score). One row is visible, showing position 1, name Juan, last name Perez, and score 0. In the background, there is a large, shiny silver trophy on a grassy field.

Puesto	Nombre	Apellido	Puntaje
1	Juan	Perez	0

Fixture del administrador

Cuando se ingresa como usuario administrador, se permite modificar los valores que se muestran en el fixture.

The screenshot shows a dark-themed interface for managing a fixture. At the top, it says "Fixture:" and "FixtureAdm". It features four groups labeled "Grupo A", "Grupo B", "Grupo C", and "Grupo D". Each group has sections for "Cuartos" (Quarterfinals) and "Semifinal". A central modal window is open over the fixture grid, showing a "Final" section with two empty boxes for scores (0 - 0) and a "3er y 4to puesto" (3rd and 4th place) section with two empty boxes for scores (0 - 0). Buttons at the bottom include "Confirmar cambios" (Confirm changes).

Al igual que en el fixture de los alumnos, el administrador puede ingresar a cada grupo y modificar los valores de los partidos de los grupos. Como los grupos están previamente creados, el administrador no es quien asigna quién juega con quién, sino simplemente los resultados.

Penca UCU

FixtureAdm

Log Out

Fixture:

Grupo A

Cuadros

Vacio

Vacio

Argentina

Peru

Peru

Chile

0 - 0

0 - 0

0 - 0

0 - 0

Canada

Chile

Canada

Argentina

Grupo B

artos

Vacio

Vacio

Grupo C

Grupo D

Para los partidos de fases posteriores, es el administrador quien elige quién juega contra quién de acuerdo al resultado de los partidos.

Penca UCU

FixtureAdm

Log Out

Fixture:

Grupo A

Cuadros

Vacio

Vacio

Argentina

Bolivia

Seleccionar

Seleccionar

Grupo B

artos

Vacio

Vacio

Grupo C

Grupo D

Herramientas

Angular

Angular es un framework para diseñar y desarrollar aplicaciones single-page. Es mantenido por un equipo de Google, el cual ofrece una gran variedad de herramientas, APIs y librerías para simplificar y facilitar el desarrollo. La principal razón de elección de Angular es que los integrantes de los equipos cuentan con experiencia trabajando con este framework, lo que

agilizaría el desarrollo de la aplicación y permitiría destinar más tiempo a otras tareas. El diseño de la aplicación se basa en diferentes páginas con funcionalidades específicas, por lo que Angular era ideal para este proyecto. Dado que permite la creación de componentes HTML que pueden ser utilizados como páginas independientes, es fácil segmentar la aplicación en secciones, así como también crear la lógica para cada componente y darle estilos con CSS. De esta manera fue posible dividir tareas dentro del frontend, permitiendo un desarrollo más efectivo por parte de los integrantes del equipo.

Bootstrap

Bootstrap es un framework de código abierto para el desarrollo de aplicaciones y sitios web. Es mantenido por Twitter y proporciona herramientas y componentes predefinidos. Incluye estilos CSS y componentes JavaScript para tipografía, formularios, botones, navegación y otros elementos de la interfaz. Bootstrap ofrece un diseño responsivo que se adapta a diferentes tamaños de pantalla, ofrece una gran variedad de componentes, es compatible con los navegadores modernos y es fácilmente personalizable. Al utilizar Bootstrap junto con Angular, permite mejorar el diseño y la experiencia de usuario de la aplicación, ya que ofrece facilidad de diseño y consistencia visual.

Docker

Docker es una plataforma que permite crear, probar e implementar aplicaciones a través de contenedores, los cuales son unidades aisladas que incluyen lo necesario para la ejecución de aplicaciones, como bibliotecas o herramientas del sistema. El punto fuerte de Docker es la capacidad de ejecutar código en cualquier entorno, con la seguridad de que funcionará correctamente.

Docker fue utilizado para crear una base de datos MySQL, a través de una imagen. Una imagen en Docker es una plantilla que define a su contenedor, contiene definiciones de bibliotecas y dependencias que el código necesite para funcionar. De esta manera, se podía generar una base de datos fácilmente que funcionara en todos los dispositivos sin complicaciones.

MySQL con InnoDB

MySQL es un sistema de gestión de bases de datos relacionales utilizado ampliamente en el mercado hoy, debido a su facilidad de uso, rendimiento, escalabilidad, seguridad y a que es de código abierto, entre otros motivos.

La elección de MySQL fue principalmente por el costo, ya que este sistema es gratis. Adicionalmente, al tratarse de una aplicación sencilla y con pocos datos, las capacidades de MySQL serían más que suficientes.

Postman

Postman es una plataforma para utilizar APIs, que simplifica la manera de utilizarlas. Con esta herramienta es posible utilizar los distintos métodos de las APIs de manera sencilla y rápida, permitiendo enviar mensajes con contenido personalizable tanto en headers como en el body.

A través de Postman se agiliza el proceso de testing de la API, verificando que tanto los puntos de conexión como el backend funcionaran correctamente.

NetBeans

NetBeans IDE es un entorno de desarrollo gratuito creado por Apache que permite desarrollar en varios sistemas operativos. Este IDE simplifica el desarrollo de aplicaciones que utilicen Java, HTML5, JavaScript, PHP, entre otros, al ofrecer plantillas para realizarlas. NetBeans se utilizó para desarrollar el backend y la API dentro de la aplicación. La elección se dio porque los integrantes del equipo tienen experiencia utilizándolo, lo cual facilita el desarrollo de código. Como NetBeans ofrece plantillas para crear aplicaciones, esto aceleró el proceso de creación.

DataGrip

DataGrip es una herramienta multiplataforma para bases de datos relacionales y NoSQL, que permite la conexión a las mismas, realizar consultas SQL, ver tablas, entre otras cosas. Es desarrollada por la compañía JetBrains, y ofrece la capacidad de conectarse a una gran variedad de bases de datos.

El equipo eligió esta herramienta porque contaban con experiencia utilizándola. Esto generó que crear la base de datos con sus respectivas tablas y datos fuera más sencillo y rápido. Otra característica útil de DataGrip es la capacidad de visualizar las tablas dentro de la misma aplicación, sin necesidad de realizar una consulta SELECT y ver el resultado en consola. Esto aceleró el proceso de testing de la API, para ver tanto si los datos se insertaban correctamente en las tablas como si eran consultados correctamente.

Java

Java es un lenguaje de programación creado por Sun Microsystems en 1995, que hoy en día se encuentra presente en una gran cantidad de dispositivos. Es un lenguaje potente que permite una gran versatilidad de desarrollo.

El equipo cuenta con experiencia en este lenguaje, dado que se utilizó junto con NetBeans. No fue necesario capacitarse con Java, con lo que fue posible pasar directamente al desarrollo del código. Dado que era necesario tener las consultas SQL reflejadas en el código, se optó por este lenguaje porque existen librerías para ejecutar consultas sin utilizar ORMs.

Spring Boot

Spring es un framework que sirve para crear aplicaciones en Java, mientras que Spring Boot es una extensión que minimiza las configuraciones necesarias para crear aplicaciones en Spring.

Se eligió Spring Boot por la facilidad que ofrece para crear aplicaciones web, dado que con poco trabajo se puede tener una aplicación funcionando. Se utilizó para crear la API a través de una plantilla, la cual resolvía todas las configuraciones necesarias para su funcionamiento.

JDBC

JDBC (Java Database Connectivity) es una API que permite la conexión a sistemas gestores de bases de datos en aplicaciones Java. Es utilizada para crear consultas utilizando SQL, permitiendo una gran variedad de bases de datos a las cuales es posible conectarse.

Patrones de diseño

Patrón repository

La aplicación utiliza el patrón repository, el cual separa la capa de lógica de negocio de la capa de acceso a datos. En este patrón se modelan los controladores, que son los endpoints de la API y están encargados de comunicarse con el cliente y luego están los

repositorios que son los encargados de realizar consultas a la base de datos, tanto para insertar como para obtener datos.

Patrón DTO

También utiliza el patrón DTO (Data Transfer Object), el cual crea objetos solo de lectura (no contienen más métodos que get y set) para transferir datos entre procesos. En la aplicación los DTO se utilizan para obtener objetos de la base de datos y enviarlos a los clientes, así como también se utilizan cuando el cliente desea crear datos en la base de datos, ya que envía los datos necesarios para formar un DTO que luego se manipulará y se insertarán los datos en la base.

Diagramas

Diagrama de clases

El siguiente diagrama representa la interacción entre clases en la aplicación. Dado que se utilizó el patrón repository, el comportamiento es el mismo a lo largo de la aplicación, cambiando únicamente las clases que actúan.

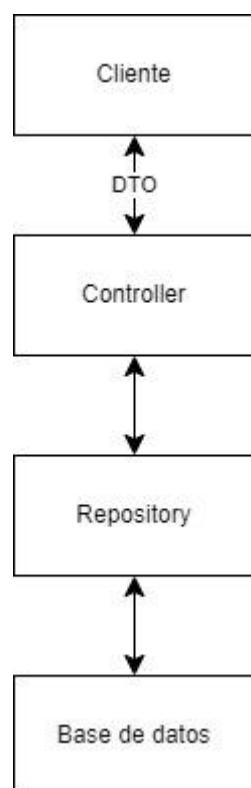
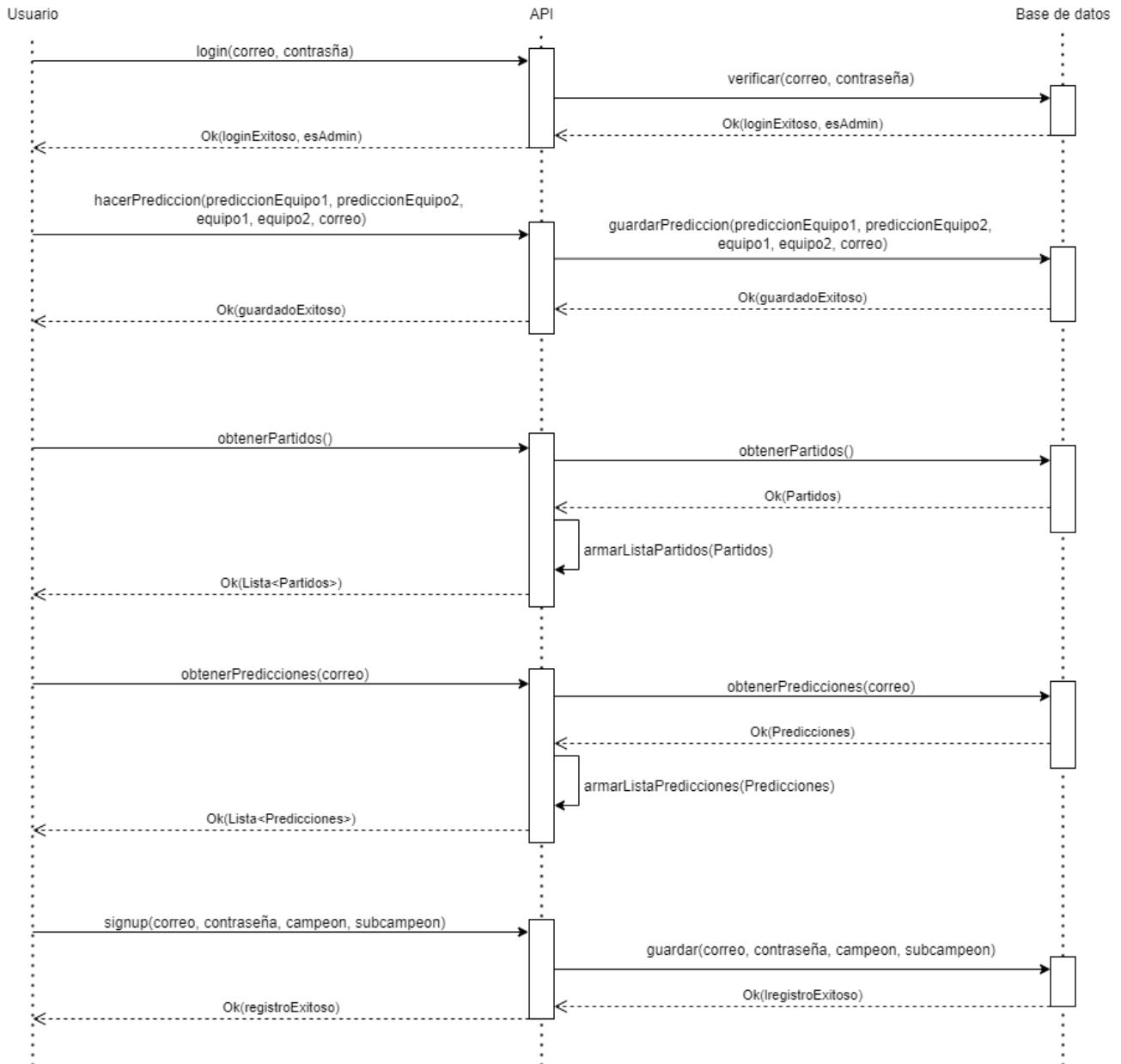


Diagrama de secuencia



Desarrollo del código

Frontend

Para el desarrollo del frontend se utilizó el framework Angular. Este utiliza TypeScript como su lenguaje principal, junto con HTML y CSS para el diseño. La estructura básica de una aplicación Angular consiste en una estructura de carpetas y archivos, los más importantes incluyen:

- ‘node_modules/’: Contiene todos los paquetes y dependencias que el proyecto necesita para funcionar. Esta carpeta se genera automáticamente al ejecutar el comando ‘npm install’, que instala todas las dependencias listadas en el archivo package.json. Esta carpeta no se debe incluir al implementar la aplicación en un repositorio git. Si se mueve el proyecto a una nueva ubicación, habrá que volver a ejecutar ‘npm install’ para regenerar esta carpeta.
- ‘src/’: Contiene el código fuente de la aplicación.
 - ‘app/’: Se encuentran los componentes, servicios, módulos, interfaces y archivos relacionados con la aplicación.
 - ‘assets/’: Contiene archivos estáticos como imágenes.
 - ‘environments/’: Archivos de configuración para diferentes entornos (desarrollo, producción).
- ‘angular.json’: Archivo de configuración principal del proyecto Angular.
- ‘package.json’: Lista de dependencias y scripts del proyecto.
- ‘tsconfig.json’ Configuración del compilador TypeScript.

Para crear un nuevo proyecto se utiliza el comando CLI ‘ng new’, en el caso del proyecto:

- ng new ObligatorioBD2-Angular

Cuando se ejecuta el comando solicitará información sobre las características que se desean incluir. Angular CLI instala los paquetes Angular necesarios y otras dependencias. Al crear el proyecto, el mismo contiene una aplicación de bienvenida simple, lo que permite ejecutarla desde el primer momento. Para iniciar la aplicación hay que ubicarse en el proyecto ‘ObligatorioBD2-Angular’ y ejecutar:

- ng serve --open

Este comando crea la aplicación e inicia el servidor de desarrollo. Cuando se realiza un cambio en un archivo, como se encuentra constantemente observando los archivos de

origen, se realizará una reconstrucción en el archivo modificado. La bandera ‘--open’ hará que la aplicación se abra en ‘<http://localhost:4200>’.

Componentes

Los componentes son la unidad básica de una aplicación Angular. Cada componente tiene los archivos:

- `.component.ts`: Contiene la lógica del componente
- `.component.html`: Plantilla HTML
- `.component.css`: Estilos CSS
- `.component.spec.ts`: Para realizar pruebas unitarias

Para la aplicación de la PencaUCU se crearon los componentes:

- `login`: corresponde al inicio de sesión.
- `signup`: para que los usuarios se registren.
- `navbar`: barra de navegación que está presente en las diferentes vistas dentro de la aplicación.
- `home`: es la vista que aparece al iniciar sesión.
- `fixture`: aquí se representa el fixture de la Copa América, el cual se irá actualizando a medida que un usuario administrador lo actualice. Este componente a su vez se compone de:
 - `partido-fixture`: cada partido desplegado corresponde a los partidos que se jugaron y jugarán en la Copa. Como incluye mucha información respecto a cada partido, es otro componente.
- `fixture-admin`: a esta vista sólo tendrán acceso los usuarios administradores. Es donde irán actualizando el fixture, ingresando los diferentes partidos y resultados.
 - `equipo`: cuando un administrador ingresa los partidos que se jugarán, utiliza una lista predefinida de equipos, para que pueda seleccionar los equipos que se enfrentarán, se creó este componente ‘equipo’.
- `apuestas`: esta vista muestra los partidos a los que los usuarios aún pueden apostar, para ello utiliza el componente ‘prediccion’, ya que no sólo se mostrarán las predicciones en esta vista de apuestas, sino que también los usuarios pueden ver sus predicciones pasadas en la vista predicciones.
- `prediccion`: el componente prediccion incluye las predicciones hechas por los usuarios, que a su vez se compone de los componentes:

- predicción-item: mientras ‘predicción’ es la vista a la que accede el usuario, ‘predicción-item’ es el componente encargado del despliegue de las distintas predicciones.
- predicción-usuario: como cada usuario tiene sus propias predicciones, se creó un componente para manejar el acceso a la base de datos correspondiente con el usuario que inicia sesión.
- partido: como las apuestas sólo se pueden realizar y modificar hasta 1 hora antes de que se juegue el partido, este componente es el encargado de controlarlo.
- leaderboard: despliega la lista de posiciones de los alumnos en orden decreciente de puntajes obtenidos hasta el momento.

Para crear cada componente hay que ejecutar el comando ‘ng generate componente’, por ejemplo:

- ng generate component apuestas

Diseño de componentes

Para el diseño se utilizó Bootstrap, que incluye una colección de herramientas y componentes predefinidos que permiten crear interfaces de usuario de manera rápida. Una ventaja de utilizar este framework es que incluye componentes predefinidos como botones, formularios, barras de navegación. Además, se pueden personalizar fácilmente utilizando CSS.

Bootstrap se instala a través de npm:

- npm install bootstrap

Después se incluye en el proyecto añadiendo las rutas a los archivos CSS y JS en el archivo ‘angular.json’.

Servicios

Los servicios son clases que contienen lógica compartida que puede ser utilizada por múltiples componentes. Los servicios se utilizan para manejar datos y lógica de negocio.

Para crear los servicios hay que correr el comando ‘ng generate service’, por ejemplo:

```
- ng generate service alumno
```

Esto genera los archivos:

- ‘.service.ts’: Contiene la lógica del servicio
- ‘service.spec.ts’: Pruebas unitarias

Los servicios son ideales para manejar datos, por lo que se utilizan para recuperar los datos de la API. Angular proporciona ‘HttpClient’, una API para manejar peticiones HTTP.

En el caso de la aplicación, se crearon los servicios:

- alumno: contiene los métodos ‘getLeaderboard’ para acceder a la lista de alumnos y sus puntajes en la penca, ‘getTop15’ para acceder solamente a los 15 alumnos con mayor puntaje y ‘getPuntaje’ donde indicando un usuario, devuelve el total de puntos que tiene.
- equipo: contiene el método ‘getEquipos’, con el cual se accede al nombre y bandera de los equipos.
- partido: contiene los métodos ‘getPartidos’ para acceder a los partidos pasados y futuros subidos a la base de datos por el administrador. Con ‘editarPartido’ el administrador podrá modificar los partidos para ingresar la cantidad de goles luego de jugado. Por último, ‘actualizarPartidos’ es un método que se llama cada vez que un administrador realiza cambios sobre el fixture, de esta manera se actualiza el fixture para todos los alumnos.
- predicción: con ‘getPredicciones’ se accede a las predicciones hechas por el usuario que se indique. También contiene el método ‘crearPrediccion’, para cuando un usuario ingresa una nueva predicción del resultado de un partido, y ‘actualizaPrediccion’, ya que el usuario podrá modificar su predicción hasta 1 hora antes de que se juegue el partido.
- usuario: tiene los métodos ‘login’, que devuelve si un usuario inicia sesión exitosamente y ‘signup’ que registra a un usuario a la penca si es que el mismo ingreso la información correspondiente.

Módulos

Los módulos son contenedores que agrupan componentes, directivas, pipes y servicios relacionados. El módulo principal es ‘AppModule’. Se pueden crear utilizando el comando: ‘ng generate module’.

Interfaces

Las interfaces en TypeScript se utilizan para definir la estructura de objetos. Son muy útiles para definir cómo deben ser los datos que se manejan en una aplicación, especialmente cuando se trabaja con APIs. Angular utiliza las interfaces para representar los datos de manera clara y segura.

Para esta aplicación se crearon las siguientes interfaces, las cuales coinciden con el formato de datos json devuelto por la API:

- alumno: nombre, apellido, puntaje. Se utiliza para el leaderboard.
- equipo: nombre, bandera. Se utiliza para las apuestas, el fixture y las predicciones, ya que allí se despliega el nombre y bandera de los países.
- partido: id, golesEquipo1, golesEquipo2, imagenEquipo1, imagenEquipo2, posicionFormulario, equipo1, equipo2, fecha, hora, etapa. Se incluyen todos los datos útiles que respecta a un partido. Estos datos se utilizarán para el fixture.
- predicion: correoUsuario, predicionEquipo1, predicionEquipo2, equipo1, equipo2, puntaje, imagenEquipo1, imagenEquipo2, fecha, hora. Las predicciones, como su nombre indica, se utilizan para guardar y desplegar las predicciones hechas por cada usuario en específico en el mismo formato.
- registro: nombre, apellido, correo, contraseña, campeon, subcampeon, carrera. A la hora de registrarse con usuario, el mismo debe completar el formulario con estos datos.
- usuario: correo, contraseña. Para ingresar al sitio web, el usuario debe ingresar estos datos.

Login

Un usuario para acceder a la aplicación debe ingresar su correo y contraseña. Para el ingreso de datos en HTML se debe utilizar el tag `<input />` además de la directiva `[(ngModel)]`, lo que hará que el dato ingresado se actualice automáticamente en el TypeScript, donde están definidas las variables con `@Input()`. Login incluye un método 'login' que es llamado al presionar el botón "Iniciar sesión" con `(click)="login()"`. Este método utiliza la interfaz Usuario para mandar los datos a la API en un formato claro que pueda leer. Es en esta comunicación con la API que entra en juego el servicio de usuario, ya que allí se incluye el llamado al método POST a la url 'http://localhost:8080/login' que corresponde a la

API, donde se le manda el usuario (correo y contraseña), y devuelve si inicia sesión exitosamente y si es un usuario administrador o no. En la interacción con los servicios que realizan llamadas HTTP asincrónicas, es esencial incluir el método ‘.subscribe()’, el cual maneja el flujo de datos asincrónico devuelto, asegurando que la aplicación reaccione adecuadamente. En caso de haber ingresado credenciales válidas, además de autorizar el acceso a la aplicación, se guardará su correo y su rol, que viene en la respuesta como un booleano ‘esAdmin’, en el localStorage del navegador, ya que estos datos serán utilizados por otros componentes.

Sign up

Al igual que en el login, el usuario deberá ingresar diferentes datos en inputs, que se guardan en variables en el TypeScript que se actualizan dinámicamente. Esta información del usuario se estructura en un objeto de tipo Registro, y es enviada al servicio Usuario, el cual contiene un método ‘signup’, donde llamando a la API se validan los parámetros ingresados y en caso de ser coherentes, se crea el usuario. También utiliza el método ‘.subscribe()’ para esperar la respuesta de la API, en caso de ser un registro exitoso, guarda en el localStorage el correo y si es administrador. En caso de fallo, no permitirá avanzar e ingresar al sitio a menos que se ingresen datos coherentes. Esta comunicación se activa con el botón “Registrarse” que con ‘(click)=“signup()”’ llama al método ‘signup’, el cual se comunica con el servicio de usuario donde a través del método POST a la url <http://localhost:8080/usuario/crearlumno>, le envía a la API los datos de registro.

Home

El usuario al acceder a su cuenta le aparece el Home, que se compone principalmente de CSS para darle la bienvenida. Y es a través de la barra de navegación que el usuario recorrerá las distintas vistas del sitio.

La navbar muestra las vistas que corresponden al tipo de usuario (administrador o alumno) que ingresó accediendo a la variable ‘esAdmin’ almacenada por login o signup en el localStorage. Para que se muestre lo correspondiente a cada rol en el HTML se utiliza *ngIf=“esAdmin == ‘true’” (o false), donde dentro de cada condicional se definen las rutas con ‘routerLink= “/ruta”’ por las que podrá navegar ese usuario.

Fixture

El fixture muestra los partidos que se jugaron y jugarán. Como es bastante grande, la fase de grupos se encuentra almacenada en modales, los cuales son recuadros que aparecen sobre la página, que en esta aplicación se activan a través de botones. Es por ello que el

fixture tiene buttons con la función '(click)="openModal('grupoA')"', '(click)="openModal('grupoB')"', '(click)="openModal('grupoC')"' y '(click)="openModal('grupoD')"'. Este método 'openModal' accede al modal correspondiente a partir de su id (valor pasado por parámetro en el llamado). Como son cuadros emergentes, todos los modals cuentan con un método 'closeModal', que a partir del mismo id, señala que se debe cerrar el recuadro.

Para el despliegue del fixture fuera de la fase de grupos, se tienen creadas variables para almacenar los distintos partidos que se jugarán en cada fase, ya que a diferencia del despliegue de un listado cualquiera, al tratarse de un fixture, cada partido tendrá una posición predefinida en pantalla. Para desplegar los partidos en el sitio, se utiliza el método 'getPartidos' de servicio partido, donde se accede al método GET de la API con la url 'http://localhost:8080/partido/partidos', la cual devolverá una lista de tipo Partido, interfaz definida en el proyecto. Como el despliegue de los partidos respeta un orden determinado, se creó un método 'asignarPartidos' el cual se encarga de, a partir de la lista de tipo Partido retornada por la API, a ordenarlos en pantalla en las variables predefinidas en TypeScript. Para darle mayor atractivo visual al fixture, también se utiliza el método 'getEquipos' del servicio equipo, el cual retorna una lista de tipo Equipo, interfaz definida en el proyecto, la cual contiene el nombre y bandera de cada equipo. Con el método 'obtenerBanderaPorNombreEquipo' el cual recibe el nombre de un equipo por parámetro, se busca en esta lista de tipo Equipo la bandera que corresponde, para desplegarla en el fixture.

Para mostrar los partidos de cada fase, utiliza el componente partido-fixture, el cual se encarga de mostrar los partidos y sus datos de una forma visualmente clara y agradable.

Predicciones

Predicciones muestra el listado histórico de predicciones hechas por el usuario utilizando un bucle '*ngFor' en el HTML para mostrar cada partido, que se representa mediante el componente partido, que recibe el partido y un formulario para que los usuarios ingresen sus predicciones de goles para cada equipo. Como cada usuario tendrá sus propias predicciones, se creó el componente 'prediccion-usuario', el cual obtiene todos los equipos y los guarda en un arreglo, llama al servicio de alumno para acceder al puntaje total que tiene hasta el momento y llama al servicio de prediccion para obtener todas las predicciones hechas hasta el momento, el 'prediccion-usuario' a través de un for que recorre las predicciones, va creando las tarjetas de prediccion, las cuales son los componentes 'prediccion-item'. Cada prediccion-item tendrá la información del partido, y el usuario podrá editar desde ahí las predicciones ya hechas. Estas actualizaciones son posibles hasta 1

hora antes de comenzar el partido, por lo que en el TypeScript de partidos se realiza un condicional if, el cual controla que la diferencia entre el tiempo actual (now.getTime()) y la hora del partido (partidoFecha.getTime()) es mayor a 60 minutos. En caso afirmativo, el botón “Apostar” estará habilitado, pero en caso negativo, se deshabilitará, impidiendo que modificar su apuesta, si es que ya había apostado.

Apostar

Es muy similar a predicciones, solamente que en apostar se encuentran todos los partidos. Es en el componente partidos, el cual es llamado desde predicción, el cual se despliega en apuestas, que se encuentra el formulario para que los usuarios ingresen sus predicciones de goles para cada equipo. Cuando se presiona el botón ‘Apostar’ se llama al método ‘apostar’, donde se crea un objeto de tipo Prediccion que incluye los datos del usuario y las predicciones de goles, que se envía a la API mediante el servicio predicción a través del método ‘crearPrediccion’, el cual llama al método POST de la API a la url ‘<http://localhost:8080/prediccion/crear>’. Al igual que en predicciones, sólo le será posible apostar en partidos hasta 1 hora antes de que se jueguen.

Leaderboard

El componente Leaderboard muestra un listado de puntajes de los alumnos en orden decreciente. Este despliega el registro de puntos totales de cada alumno. En el HTML, utiliza el bucle *ngFor="let alumno of alumnos; let i = index" para recorrer la lista de alumnos almacenada en la variable alumnos de tipo Alumno en el TypeScript. Durante cada iteración del bucle, se accede a las propiedades del alumno (nombre, apellido y puntaje) mediante {{ alumno.nombre }}, {{ alumno.apellido }} y {{ alumno.puntaje }}, respectivamente. Esta forma de acceso es posible porque la lista de alumnos traída desde la base de datos contiene objetos de tipo Alumno, una interfaz definida en el proyecto.

Para el leaderboard, la aplicación utiliza el método del servicio de alumnos ‘getTop15’, que lo que hará, como su nombre indica, es devolver los 15 alumnos con mayor puntuación. En el servicio de alumnos se llama al método GET de la API a la url: ‘<http://localhost:8080/alumno/top15>’, la cual retorna una lista de tipo Alumno. Esta lista es recorrida con un bucle for en el TypeScript del componente Leaderboard, ya que los datos en la base de datos son almacenados en minúscula para asegurar la consistencia de los datos. Por lo que este bucle, accede a un Alumno en cada iteración y cambia la primera letra del nombre y apellido por una mayúscula.

Fixture del administrador

El fixture de administrador es muy similar al fixture para alumnos. La gran diferencia entre ambos, es que el administrador debe ingresar los resultados y partidos en el fixture, mientras el alumno sólo puede ver el fixture. Es por ello que, además de contener inputs en el HTML, utiliza los métodos ‘confirmar’ y ‘confirmarCambio’ los cuales son llamados al presionar botones. El método ‘confirmar’ actualiza muchos partidos y accede al método ‘actualizarPartidos’ del servicio Partido, donde se llama al método PUT de la API a través de la url ‘<http://localhost:8080/partido/editar>’ pasándole la lista de tipo Partido. Mientras ‘confirmarCambio’ envía un único partido a la base de datos, a través del método ‘editarPartido’ del servicio Partido y la url de la APU ‘<http://localhost:8080/partido/editar>’.

Backend

Como fue mencionado anteriormente, se utilizó Java para realizar el backend y la API, implementando Spring Boot y usando los patrones de diseño repository y DAO. En total se crearon 32 clases, las cuales incluyen el modelado de datos, interfaces, conexiones a bases de datos y controladores de la API. Como se mencionó en el diagrama de clases, las clases se dividen en tres categorías (DTOs, controladores y repositorios) que repiten el objetivo, cambiando los datos de la clase y la forma en que ejecuta los métodos

Borrado lógico

Algunos de los datos de la base de datos cuentan con un borrado lógico, cuyo fin es persistir datos que se desean borrar. Esto se logra colocando un atributo boolean en las entidades, donde si está con valor verdadero se considera borrado a efectos de realizar consultas, si está con valor falso entonces es un dato común y corriente. El borrado lógico sirve para recuperar datos que se quisieron borrar en el pasado.

Dentro del código, esto es reflejado con una clase abstracta que es extendida por aquellas clases que cuentan con el borrado lógico:

```
package com.bd2.api.dto;

public abstract class LogicalDelete {

    boolean borrado = false;
```

```

public boolean getBorrado() {
    return borrado;
}

public void borrar() {
    borrado = true;
}
}

```

Esta clase cuenta con el atributo booleano que indica si el dato está borrado, así como también con un método para obtener este valor y otro para simular el borrado del objeto. Fue modelado como una clase abstracta porque eran varias clases las que contaban con borrado lógico, de esta manera se tenía código reutilizable y sin repeticiones en distintas partes.

Clases DTO

Estas clases tienen como objetivo modelar los datos de la base de datos, también se utilizan para transferir los datos entre procesos y enviarlos desde o hacia al cliente. Lo característico de estas clases es que cuentan con atributos privados y métodos get y set, pero no cuentan con otras funciones. Algunos DTO extienden la clase LogicalDelete, que modela el borrado lógico.

Aquí se muestra la clase que modela un usuario, la cual cuenta con borrado lógico:

```

package com.bd2.api.dto;

public class UsuarioDTO extends LogicalDelete {

    private int id;
    private String nombre;
    private String apellido;
    private String correo;
    private String contrasenia;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}

```

```
public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellido() {
    return apellido;
}

public void setApellido(String apellido) {
    this.apellido = apellido;
}

public String getCorreo() {
    return correo;
}

public void setCorreo(String correo) {
    this.correo = correo;
}

public String getContrasenia() {
    return contrasenia;
}

public void setContrasenia(String contrasenia) {
    this.contrasenia = contrasenia;
}

public void borrarUsuario() {
    borrar();
}

public boolean getUsuarioBorrado() {
    return getBorrado();
}
}
```

Clases repositorios

El objetivo de estas es conectarse a la base de datos e interactuar con ella, a través de consultas SQL. Se utilizan tanto para insertar o actualizar datos como para obtenerlos.

Se modelaron interfaces para cada repositorio, en vista de hacer la aplicación escalable en el futuro.

```
package com.bd2.api.repositories;

import com.bd2.api.dto.PartidoDTO;
import java.util.LinkedList;

public interface IPartidoRepository {

    public LinkedList<PartidoDTO> getPartidos();
    public PartidoDTO getPartido(int id);
    public boolean editarPartido(PartidoDTO[] partidos);
    public boolean borrarPartido(int id);
    public boolean crearPartido(PartidoDTO partido);
}
```

Dada la naturaleza de la aplicación, que no se accede a la base de datos continuamente, cada método crea una conexión con la base y la cierra luego de realizar la operación, para evitar dejar conexiones sin cerrar y optimizar el uso de recursos.

```
package com.bd2.api.repositories;

import com.bd2.api.dto.PartidoDTO;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.LinkedList;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

@Service // @Service hace que se inyecte directamente en el controller
public class PartidoRepository implements IPartidoRepository {

    @Override
    @Async
```

```

public LinkedList<PartidoDTO> getPartidos() {
    LinkedList<PartidoDTO> listaResultado = new LinkedList<>();
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/obligatorio", "user",
                "password");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM Partidos
WHERE NOT borrado ORDER BY posicion_formulario ASC");
        while (rs.next()) {
            PartidoDTO partido = new PartidoDTO();
            partido.setId(rs.getInt(1));
            partido.setEquipo1(rs.getString(2));
            partido.setEquipo2(rs.getString(3));
            partido.setGolesEquipo1(rs.getInt(4));
            partido.setGolesEquipo2(rs.getInt(5));
            partido.setEtapa(rs.getString(6));
            partido.setFecha(rs.getDate(7));
            partido.setHora(rs.getTime(8));
            partido.setEstadio(rs.getString(9));
            partido.setPosicionFormulario(rs.getInt(11));
            listaResultado.add(partido);
        }
        con.close();
    } catch (Exception e) {
        System.out.println(e);
    }
    return listaResultado;
}

@Override
@Async
public PartidoDTO getPartido(int id) {
    PartidoDTO partido = new PartidoDTO();
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/obligatorio", "user",
                "password");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM Partidos
WHERE id = " + id + " AND NOT borrado");

```

```

        rs.next();
        partido.setId(rs.getInt(1));
        partido.setEquipo1(rs.getString(2));
        partido.setEquipo2(rs.getString(3));
        partido.setGolesEquipo1(rs.getInt(4));
        partido.setGolesEquipo2(rs.getInt(5));
        partido.setEtapa(rs.getString(6));
        partido.setFecha(rs.getDate(7));
        partido.setHora(rs.getTime(8));
        partido.setEstadio(rs.getString(9));
        partido.setPosicionFormulario(rs.getInt(11));
        con.close();
    } catch (Exception e) {
        System.out.println(e);
    }
    return partido;
}

@Override
@Async
public boolean editarPartido(PartidoDTO[] partidos) {
    int resultado = 0;
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/obligatorio", "user",
            "password");

        for (PartidoDTO partido : partidos) {
            String query = "UPDATE Partidos SET equipo_1 = ?,"
equipo_2 = ?, goles_equipo_1 = ?, goles_equipo_2 = ?, etapa = ?, fecha
= ?, hora = ?, estadio = ?, posicion_formulario = ? WHERE id = ?";
            PreparedStatement pstmt = con.prepareStatement(query);

            pstmt.setString(1, partido.getEquipo1());
            pstmt.setString(2, partido.getEquipo2());
            pstmt.setInt(3, partido.getGolesEquipo1());
            pstmt.setInt(4, partido.getGolesEquipo2());
            pstmt.setString(5, partido.getEtapa());
            pstmt.setDate(6, partido.getFecha());
            pstmt.setTime(7, partido.getHora());
            pstmt.setString(8, partido.getEstadio());
            pstmt.setInt(9, partido.getPosicionFormulario());
        }
    }
}

```

```

        pstmt.setInt(10, partido.getId());

        resultado += pstmt.executeUpdate();
    }

    con.close();
} catch (Exception e) {
    System.out.println(e);
}
return resultado != 0;
}

@Override
@Async
public boolean borrarPartido(int id) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/obligatorio", "user",
            "password");
        String query = "UPDATE Partidos SET borrado = true WHERE id
= ?";
        PreparedStatement pstmt = con.prepareStatement(query);

        pstmt.setInt(1, id);

        int rs = pstmt.executeUpdate();
        con.close();
        return rs != 0;
    } catch (Exception e) {
        System.out.println(e);
    }
    return false;
}

@Override
@Async
public boolean crearPartido(PartidoDTO partido) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/obligatorio", "user",
            "password");

```

```

        String query = "INSERT INTO Partidos (equipo_1, equipo_2,
goles_equipo_1, goles_equipo_2, etapa, fecha, hora, estadio, borrado,
posicion_formulario) VALUES (
                + "?, ?, -1, -1, ?, ?, ?, ?, false, ?)";
        PreparedStatement pstmt = con.prepareStatement(query);

        pstmt.setString(1, partido.getEquipo1());
        pstmt.setString(2, partido.getEquipo2());
        pstmt.setString(3, partido.getEtapa());
        pstmt.setDate(4, partido.getFecha());
        pstmt.setTime(5, partido.getHora());
        pstmt.setString(6, partido.getEstadio());
        pstmt.setInt(7, partido.getPosicionFormulario());

        int rs = pstmt.executeUpdate();
        con.close();
        return rs != 0;
    } catch (Exception e) {
        System.out.println(e);
    }
    return false;
}
}

```

Cada método crea una conexión a la base de datos utilizando JDBC, luego se especifica la URL de la base de datos junto con el nombre de la misma, y se pone el usuario y la contraseña para acceder. Luego, se crea una consulta SQL que tiene placeholders, para parametrizarla y evitar inyecciones SQL en el programa. Finalmente, se ejecuta la consulta y se implementa alguna lógica para verificar que fue hecha correctamente, la cual puede ser la cantidad de líneas afectadas o la devolución de algún objeto en caso de una consulta SELECT. Cada función es marcada como asíncrona para que se haga en otro hilo y no tener al cliente esperando el resultado. En los casos donde se devuelve una lista de objetos, la consulta SQL obtiene todos los datos de la tabla y luego se itera sobre cada dato, creando un nuevo DTO e insertándolo en una lista, que será devuelta como resultado de la función.

Clases controladores

Los controladores se encargan de comunicar el frontend con el backend, son la API de la aplicación. Estos proveen métodos HTTP para conectarse al backend y permiten tanto

obtener datos del cliente como enviarlos. Se crearon rutas específicas para cada funcionalidad, así como también una ruta para el controlador en sí.

Los controladores utilizan los repositorios para el tránsito de datos con la base de datos. Los repositorios son almacenados en variables, que serán inyectadas en el constructor del controlador automáticamente al iniciar la aplicación. Esto se logra al agregar `@Autowired` en el constructor de la clase.

Para enviar y recibir datos, se utilizan variables en la URL y en el body del request. Un ejemplo de lo primero es al obtener un único partido (`getPartido`) que se pasa un id por la URL, el cual está reflejado en el parámetro de la función, que tiene `@PathVariable`. Un ejemplo de lo segundo es en la función `editarPartido`, donde se mandan los datos en el body del request, el cual es serializado automáticamente en un DTO al agregar `@RequestBody` en el parámetro de la función.

Los métodos de los controladores siempre devuelven algo para saber si fue exitoso el request o no. En caso de pedir datos, se envían los datos directamente, pero si no se pide ningún dato entonces se devuelve un booleano que indica si la operación fue exitosa o no.

```
package com.bd2.api.web;

import com.bd2.api.dto.PartidoDTO;
import com.bd2.api.repositories.IActualizarPuntajesRepository;
import com.bd2.api.repositories.IPartidoRepository;
import java.util.LinkedList;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/partido")
class PartidoController {

    private final IPartidoRepository partidoRepository;
    private final IActualizarPuntajesRepository
actualizarPuntajesRepository;

    @Autowired // @Autowired sirve para inyectar automáticamente
```

```

    public PartidoController(final IPartidoRepository
partidoRepository, final IActualizarPuntajesRepository
actualizarPuntajesRepository) {
        this.partidoRepository = partidoRepository;
        this.actualizarPuntajesRepository =
actualizarPuntajesRepository;
    }

    @GetMapping(path = "/partidos")
    public @ResponseBody LinkedList<PartidoDTO> getPartidos() {
//@ResponseBody serializa JSON en el body del response
        return partidoRepository.getPartidos();
    }

    @GetMapping(path = "/{id}")
    public @ResponseBody PartidoDTO getPartido(@PathVariable int id) {
//@PathVariable es para sacar el id de la url pasada como parametro
        return partidoRepository.getPartido(id);
    }

    @PutMapping(path = "/editar")
    public boolean editarPartido(@RequestBody PartidoDTO[] partidos) {
//@RequestBody arma un PartidoDTO con los datos del body
        return partidoRepository.editarPartido(partidos) &&
actualizarPuntajesRepository.actualizarPuntajes(partidos);
    }

    @PutMapping(path = "/eliminar/{id}")
    public boolean borrarPartido(@PathVariable int id) {
        return partidoRepository.borrarPartido(id);
    }

    @PostMapping(path = "/crear")
    public boolean crearPartido(@RequestBody PartidoDTO partido) {
        return partidoRepository.crearPartido(partido);
    }
}

```

Conexiones a base de datos

Como fue indicado previamente, se utiliza JDBC para realizar las conexiones a la base. A través de esta API se pueden ejecutar consultas SQL en la aplicación:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

```

Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/obligatorio", "user", "password");
String query = "INSERT INTO Partidos (equipo_1, equipo_2,
goles_equipo_1, goles_equipo_2, etapa, fecha, hora, estadio, borrado,
posicion_formulario) VALUES (
    ?, ?, -1, -1, ?, ?, ?, ?, false, ?)";
PreparedStatement pstmt = con.prepareStatement(query);

pstmt.setString(1, partido.getEquipo1());
pstmt.setString(2, partido.getEquipo2());
pstmt.setString(3, partido.getEtapa());
pstmt.setDate(4, partido.getFecha());
pstmt.setTime(5, partido.getHora());
pstmt.setString(6, partido.getEstadio());
pstmt.setInt(7, partido.getPosicionFormulario());

int rs = pstmt.executeUpdate();
con.close();

```

Dentro del string de conexión se indica que se utilizará un driver jdbc, se indica el sistema gestor de base de datos (en nuestro caso MySQL), se pone la URL de la base de datos junto con el puerto y el nombre de la base; luego se indican las credenciales para iniciar sesión (la base de datos no cuenta con restricciones de usuario, por lo que se usa el usuario por defecto) y se crea la conexión. Después se crea una consulta y se guarda en un string, en nuestro caso utilizamos consultas parametrizadas, que luego se insertan los valores de cada parámetro. Finalmente se ejecuta la consulta, la cual puede depender del tipo de consulta qué método se utiliza, siendo executeUpdate() utilizado en aquellas consultas que afectan a la base de datos (inserciones, actualizaciones, etc) y executeQuery() en las consultas que no la afectan (select). Una vez que se realizó la consulta, se cierra la conexión.

Consultas SQL

Las consultas que se realizaron a la base de datos fueron SELECT, UPDATE e INSERT. Todas las consultas son parametrizadas, para evitar inyecciones SQL.

Consultas SELECT

Las consultas SELECT se realizaron cada vez que era necesario obtener datos, por ejemplo al obtener la lista de partidos o usuarios. Siempre se filtra según si el dato está marcado como borrado.

```
SELECT * FROM Partidos WHERE NOT borrado ORDER BY posicion_formulario  
ASC
```

```
SELECT * FROM Usuarios WHERE NOT borrado
```

En algunos casos las consultas SELECT fueron más complejas, necesitando unir los datos de varias tablas a través de JOIN:

```
SELECT nombre, apellido, SUM(puntaje) AS puntaje  
FROM Predicciones  
INNER JOIN Usuarios on Predicciones.id_alumno = Usuarios.id  
INNER JOIN Puntajes on Predicciones.tipo_puntaje = Puntajes.tipo  
GROUP BY nombre, apellido  
ORDER BY puntaje DESC
```

Este ejemplo refiere a la consulta hecha para obtener el ranking de puntajes de los alumnos. Se devuelven todos los alumnos con su nombre, apellido y la suma de los puntajes, y para esto es necesario unir con la tabla de usuarios para obtener el nombre y apellido (la tabla de Predicciones cuenta con el id del alumno, pero sus datos están contenidos en la tabla Usuarios) y también unir con la tabla Puntajes (esta tabla contiene el valor asociado a cada tipo de puntaje). Luego, como cada usuario tiene muchas predicciones hechas, se realiza la suma de los valores de esta columna. Finalmente, se agrupan los datos por nombre y apellido para que quede en una única fila cada alumno, y se ordenan de forma descendente según el puntaje.

Consultas UPDATE

Las consultas UPDATE son complejas en algunos casos. Por ejemplo cuando se actualiza un partido se ejecutan una serie de sentencias para actualizar los puntajes de los competidores, comenzando por editar la información del partido para poder comparar los puntajes:

```
UPDATE Partidos SET equipo_1 = ?, equipo_2 = ?, goles_equipo_1 = ?,  
goles_equipo_2 = ?, etapa = ?, fecha = ?, hora = ?, estadio = ?,  
posicion_formulario = ? WHERE id = ?
```

Una vez que se ejecuta esta sentencia, se comienzan a actualizar los puntajes de las predicciones de cada participante que haya realizado una para ese partido. Primero se cambian los puntajes que fueron exactos:

```
UPDATE Predicciones  
SET tipo_puntaje = 'Exacto'  
WHERE id_partido = (SELECT id FROM  
Partidos WHERE equipo_1 LIKE ? AND equipo_2 LIKE ?)
```

```

                AND predicción_equipo_1 = (SELECT
goles_equipo_1 FROM Partidos WHERE equipo_1 LIKE ? AND equipo_2 LIKE ?)
                                AND predicción_equipo_2 = (SELECT
goles_equipo_2 FROM Partidos WHERE equipo_1 LIKE ? AND equipo_2 LIKE ?)

```

Esta sentencia primero filtra aquellas predicciones cuyos valores para los goles de cada equipo coinciden con los goles hechos en el partido, para luego cambiar el valor de la columna “tipo_puntaje” a “Exacto”. Como en la aplicación solamente se sabe cuáles equipos juegan, primero es necesario obtener el id del partido con esa información. La primera condición de la sentencia es que el id del partido coincide con el id de la subconsulta que busca aquel partido donde jugaron estos dos equipos. Luego, se agregan las condiciones de que las predicciones para ambos equipos deben coincidir con los goles hechos por cada equipo.

Luego de esta sentencia, se ejecuta una similar que actualiza el puntaje a “Correcto”, es decir, aquellos que acertaron qué equipo gana o empata pero no acertaron en la cantidad de goles. Para esta sentencia fue necesario ejecutar lógica adicional para saber qué país ganó, para luego filtrar según esa condición.

```

String operador = "";
        if (partido.getGolesEquipo1() >
partido.getGolesEquipo2()) {
            operador = ">";
        }
        else if (partido.getGolesEquipo1() <
partido.getGolesEquipo2()) {
            operador = "<";
        }
        else {
            operador = "=";
        }
query = """
        UPDATE Predicciones
        SET tipo_puntaje = 'Correcto'
        WHERE id_partido = (SELECT id FROM Partidos
WHERE equipo_1 LIKE ? AND equipo_2 LIKE ?)
                AND NOT (predicción_equipo_1 = (SELECT
goles_equipo_1 FROM Partidos WHERE equipo_1 LIKE ? AND equipo_2 LIKE ?)
                                AND predicción_equipo_2 = (SELECT
goles_equipo_2 FROM Partidos WHERE equipo_1 LIKE ? AND equipo_2 LIKE
?))

```

```
        AND predicción_equipo_1"" + operador +
"predicción_equipo_2";
```

La variable “operador” obtiene la relación entre goles, siempre manteniendo el orden de primero el equipo 1 y luego el 2. Una vez que se determina si los goles fueron mayores, menores o iguales, se ejecuta la predicción utilizando este operador. Al igual que en la anterior, primero se obtiene el id del partido con una subconsulta, y luego se concatenan condiciones para verificar el resultado. Dado que no pueden coincidir las predicciones con los resultados, se verifica que la predicción no sea igual a los goles efectivamente hechos, y luego sí se agrega que la predicción del equipo 1 tenga la relación determinada por la variable “operador” respecto a la predicción del equipo 2.

Finalmente, se actualizan los valores incorrectos. Esta operación es más sencilla porque implica actualizar aquellos que no son exactos ni correctos y colocarles el valor “Incorrecto”:

```
UPDATE Predicciones
    SET tipo_puntaje = 'Incorrecto'
    WHERE id_partido = (SELECT id FROM Partidos
WHERE equipo_1 LIKE ? AND equipo_2 LIKE ?)
        AND NOT tipo_puntaje = 'Correcto'
        AND NOT tipo_puntaje = 'Exacto'
```

Nuevamente se busca el id del partido con una subconsulta, luego los filtros es que el tipo de puntaje no sea correcto ni exacto.

Otro ejemplo de consulta compleja es al actualizar una predicción, donde se debe obtener el id del usuario a partir del correo electrónico (en la aplicación es el único dato para identificar al usuario) y donde se debe obtener el partido de forma similar a las consultas anteriores:

```
UPDATE Predicciones SET predicción_equipo_1 = ?, predicción_equipo_2 =
?
    WHERE id_alumno = (SELECT id FROM Usuarios
WHERE correo = ?) AND id_partido = (SELECT id FROM Partidos WHERE
equipo_1 like ? and equipo_2 like ?)
```

Consultas INSERT

Estas consultas fueron utilizadas cuando el usuario crea datos y debe persistirlos, por ejemplo cuando crea una predicción o al registrarse en la aplicación. En el caso de la predicción, se busca nuevamente el identificador del partido a partir de los equipos y también el identificador del usuario a partir de su correo:

```
INSERT INTO Predicciones (predicción_equipo_1, predicción_equipo_2,
id_partido, id_alumno, tipo_puntaje) VALUES (
```

```

        ?,
        ?,
        (SELECT id FROM Partidos WHERE
equipo_1 like ? AND equipo_2 like ?),
        (SELECT id FROM Usuarios WHERE
correo = ?),
        'No determinado')

```

Como se está creando la predicción, significa que el partido no ocurrió todavía, por lo que el puntaje será “No determinado” por defecto.

En el caso de registrar un usuario, se diferencia entre alumnos y administradores. Los alumnos deben ingresar un campeón, subcampeón y la carrera que cursan al momento de registrarse, mientras que los administradores solamente tienen los datos básicos. Además, dado que se cuenta con una tabla para alumnos y otra para administradores, al momento de insertar un nuevo usuario también se debe insertar en la tabla correspondiente al tipo de usuario. Si se quiere crear un administrador:

```

INSERT INTO Usuarios (nombre, apellido, correo, contrasenia, borrado)
VALUES (?, ?, ?, ?, false);

```

Primero se inserta en la tabla de usuarios, donde se le asignará un identificador automáticamente que será el que se inserte en la tabla de administradores. Para esto, es necesario realizar una subconsulta, buscando el último usuario de la tabla (el que se acaba de crear) y obteniendo su id:

```

INSERT INTO Administradores (id) SELECT id FROM Usuarios ORDER BY id
DESC LIMIT 1

```

En el caso de los alumnos, comienza insertándose de igual manera en la tabla de usuarios, luego se inserta en la tabla de alumnos realizando una subconsulta igual a la del administrador para obtener el id del registro recién creado:

```

INSERT INTO Alumnos (id, campeon, subcampeon) SELECT id, ?, ? FROM
Usuarios ORDER BY id DESC LIMIT 1

```

Por último, se debe registrar en la base de datos la carrera que cursa, lo cual se logra al insertar en una tabla que contiene todos los usuarios y sus carreras:

```

INSERT INTO Cursa (nombre_carrera, id_alumno, borrado) SELECT ?, id,
false FROM Alumnos ORDER BY id DESC LIMIT 1

```

Conclusiones

Para realizar la aplicación, fue necesario investigar cada parte por separado para que sea eficiente. El manejo de los datos fue fundamental, por lo que el diseño de la base de datos era imprescindible que fuera correcto. A través de varias iteraciones incrementales, la base de datos quedó construida de forma tal que en el peor caso, solamente se necesitó unir dos tablas para realizar consultas. También se trató de que el diseño fuera fácilmente escalable en caso de ser necesario agregar nuevas funcionalidades, al gestionar los datos en distintas tablas que luego podrían volver a utilizarse en diferentes partes de la aplicación. Un ejemplo de esto fue con la tabla “Países”, que se utiliza en los partidos y al momento de registrar el campeón y subcampeón ingresado por un usuario. La metodología descendente utilizada para construir el modelo fue acertada, dado que el Modelo Entidad Relación inicial sirvió como base para crear una base de datos que crecería junto con la aplicación, sin necesidad de realizar mayores cambios. El diseño lógico fue acorde a lo presentado en los materiales teóricos, siendo capaces de transformar el MER en tablas dentro de la base de datos sin ningún tipo de problemas.

Dentro de las herramientas utilizadas, MySQL fue una buena elección como base de datos. Originalmente se trató de utilizar SQL Server, pero hubo complicaciones a la hora de conectarse con la misma. En cambio, con MySQL, no hubo ningún problema con la conexión en ningún momento. Además, SQL Server se utilizó en un servicio de Azure, con lo cual tenía un costo mensual, mientras que para MySQL se utilizó una imagen en Docker completamente gratuita. La decisión de utilizar SQL Server fue para poder acceder a una misma base de datos que no fuera local en cada equipo, pero esto se resolvió con un script SQL que crea la base de datos e inserta datos. En caso de haber una modificación, se puede crear un nuevo contenedor en Docker a través del archivo docker-compose, el cual automáticamente corre el script SQL cuando termina de crear el contenedor.

Bibliografía

Introduction to the Angular docs. (s. f.). Angular. Recuperado el 15 de mayo de 2024

de: <https://v17.angular.io/docs>

What is Angular?. (s. f.). Angular. Recuperado el 15 de mayo de 2024 de:

<https://angular.dev/overview>

Docker: Accelerated Container Application Development. (2024, 20 mayo). Docker.

<https://www.docker.com/>

Contenedores de Docker | ¿Qué es Docker? | AWS. (s. f.). Amazon Web Services, Inc. Recuperado el 15 de mayo de 2024 de:

<https://aws.amazon.com/es/docker/>

¿Qué es Docker? (s. f.). Oracle Argentina. Recuperado el 15 de mayo de 2024 de:

<https://www.oracle.com/ar/cloud/cloud-native/container-registry/what-is-docker/>

¿Qué es MySQL? (s. f.). Oracle Colombia. Recuperado el 18 de mayo de 2024 de:

<https://www.oracle.com/co/mysql/what-is-mysql/>

MySQL. (s. f.). <https://www.mysql.com/> Recuperado el 18 de mayo de 2024 de:

MySQL | Google Cloud. (s. f.). Google Cloud. Recuperado el 18 de mayo de 2024 de: <https://cloud.google.com/mysql?hl=es>

GeeksforGeeks. (2023, 7 junio). *How to Work with Databases using Spring Boot?*

GeeksforGeeks. Recuperado el 28 de mayo de 2024 de:

<https://www.geeksforgeeks.org/how-to-work-with-databases-using-spring-boot/>

NetBeans IDE | Oracle Argentina. (s. f.). Recuperado el 27 de mayo de 2024 de:

<https://www.oracle.com/ar/tools/technologies/netbeans-ide.html>

NetBeans, A. (s. f.). *Welcome to Apache NetBeans.* Recuperado el 27 de mayo de 2024 de: <https://netbeans.apache.org/front/main/index.html>

DataGrip | JetBrains. (s.f.). Recuperado el 27 de mayo de 2024 de:

<https://www.jetbrains.com/es-es/datagrip/>

What is Java technology and why do I need it? (s.f.). Recuperado el 27 de mayo de 2024 de: https://www.java.com/en/download/help/whatis_java.html

Java Software. (s. f.). Oracle. Recuperado el 9 de junio de 2024 de:

<https://www.oracle.com/java/>

Spring boot. (s. f.). Spring Boot. Recuperado el 9 de junio de 2024 de:

<https://spring.io/projects/spring-boot>

P of EAA. (pdf). <https://martinfowler.com/books/eaa.html>

La penca de la tienda. (s. f.). Recuperado el 9 de junio de 2024 de:

<https://penca.tiendainglesa.com.uy/#!/login>

Penka-Copa America & Euro 2024 - Apps en Google Play. (s. f.). Recuperado el 9 de junio de 2024 de:

https://play.google.com/store/apps/details?id=com.penkapro.app&hl=es_419

Comm, M. (s. f.). Montevideo COMM. Copa América USA 2024. Recuperado el 9 de junio de 2024 de: <https://penca.montevideo.com.uy/>

Informix Servers 12.10. (s. f.). Recuperado el 9 de junio de 2024 de:

<https://www.ibm.com/docs/es/informix-servers/12.10?topic=started-what-is-jdbc>

Angular project structure tutorial. (s. f.). MDB - Material Design For Bootstrap.

Recuperado el 24 de junio de 2024 de:

<https://mdbootstrap.com/docs/angular/extended/angular-project-structure/>

Angular. (s. f.). Recuperado el 24 de junio de 2024 de:

<https://docs.angular.lat/tutorial/toh-pt0>

Angular. (s. f.). Recuperado el 24 de junio de 2024 de:

<https://docs.angular.lat/guide/setup-local>

Node.js — Run JavaScript everywhere. (s. f.). Recuperado el 24 de junio de 2024 de: <https://nodejs.org/en>

npm: node-modules. (s. f.). Npm. Recuperado el 26 de junio de 2024 de:

<https://www.npmjs.com/package/node-modules>

Docker Desktop: The #1 Containerization Tool for Developers | Docker. (2023, 21

diciembre). Docker. Recuperado el 26 de junio de 2024 de:

<https://www.docker.com/products/docker-desktop/>

DBMSs. 2024 [Diapositiva de PowerPoint]. Universidad Católica del Uruguay.

[https://webasignatura.ucu.edu.uy/mod/resource/view.php?id=682798.](https://webasignatura.ucu.edu.uy/mod/resource/view.php?id=682798)

MER. 2024 [Diapositiva de PowerPoint]. Universidad Católica del Uruguay.

[https://webasignatura.ucu.edu.uy/pluginfile.php/837482/mod_resource/content/4/BD2%2002.02%20MER.pdf.](https://webasignatura.ucu.edu.uy/pluginfile.php/837482/mod_resource/content/4/BD2%2002.02%20MER.pdf)

Modelado de Datos. 2024 [Diapositiva de PowerPoint]. Universidad Católica del Uruguay.

[https://webasignatura.ucu.edu.uy/pluginfile.php/715875/mod_resource/content/4/BDII%20MC%2001%20Modelado%20de%20Datos.pdf.](https://webasignatura.ucu.edu.uy/pluginfile.php/715875/mod_resource/content/4/BDII%20MC%2001%20Modelado%20de%20Datos.pdf)

Diseño Lógico-Físico. 2024 [Diapositiva de PowerPoint]. Universidad Católica del Uruguay.

[https://webasignatura.ucu.edu.uy/pluginfile.php/743712/mod_resource/content/1/BDII%20MC%2005%20Dise%C3%B1o%20Logico-Fisico.pdf.](https://webasignatura.ucu.edu.uy/pluginfile.php/743712/mod_resource/content/1/BDII%20MC%2005%20Dise%C3%B1o%20Logico-Fisico.pdf)

Anexo

Enlace al proyecto

La aplicación está alojada en el siguiente repositorio de GitHub:
<https://github.com/JP206/Obligatorio-BD2-2024>

Para obtener información acerca de cómo correr la aplicación, leer el README.md del repositorio.