



InfoNBA

Álvaro Navajas Camacho

2º DAM A

Contenido

1.- Introducción	3
2.- Abstract.....	4
3.- Requisitos funcionales.....	5
4.- Diseños y análisis	7
4.1.- Diagrama de la arquitectura de la aplicación	7
4.2.- Diagramas de casos de uso	8
4.2.1.- Diagrama de casos de uso de la aplicación para usuarios.....	8
4.2.3.- Diagrama casos de uso de la aplicación para administrador	9
4.3.- Diseño de datos	10
5.- Codificación	11
6.- Manual de usuario	26
6.1.- Aplicación para usuarios.....	26
6.2.- Aplicación para administradores.....	32
7.- Requisitos de instalación	40

1.- Introducción

InfoNBA es una aplicación pensada y desarrollada para todos los aficionados de la NBA. Es muy intuitiva y fácil de entender y usar, permitiendo que cualquier persona pueda adentrarse en el mundo de la NBA. Además, los aficionados actuales tendrán acceso a una gran cantidad de datos y estadísticas.

Esta aplicación surgió de la falta de herramientas similares para todos los aficionados, especialmente para los españoles. En España, la NBA no está tan difundida, y las aplicaciones existentes sobre noticias o estadísticas tienden a centrarse únicamente en los equipos más conocidos o en las noticias que generarán más clics, sin dar visibilidad a los 30 equipos de la NBA. InfoNBA busca cambiar esto, permitiendo que la gente conozca más sobre todos los equipos y jugadores, fomentando así una mayor afición por cualquier equipo y aumentando continuamente la comunidad de fans de la liga.

El proyecto está dividido en dos aplicaciones para escritorio. Una está pensada para los usuarios, en la que podrán ver listas de equipos, estadios, jugadores y partidos, además de la clasificación de la liga. La otra, destinada a los administradores, permitirá gestionar los datos de la aplicación, pudiendo crear, editar o eliminar cualquiera de los datos existentes en la base de datos que se muestran en la aplicación para los usuarios.

2.- Abstract

InfoNBA is an application designed to bring NBA fans closer to their passion for the sport. The application offers a comprehensive and accessible experience through two distinct desktop applications: one for users and one for administrators. InfoNBA allows users to check the updated league standings, explore the list of participating teams, and obtain detailed information about players, games, and stadiums.

The user application, developed in WPF, provides a rich and functional interface where users can:

- View league standings by conferences.
- Access a complete list of teams with detailed information of each one.
- Consult all league players.
- Review information about games.
- Get details on stadiums.

The administrator application, also developed in WPF, allows comprehensive information management. Administrators can:

- Add new teams, players, games, and stadiums.
- Update existing information in the lists.
- Remove obsolete or incorrect records, keeping the database accurate and up to date.

For data storage and management, InfoNBA uses a SQL database structured into the following main tables: Teams, Players, Stadiums, and Games. These tables are interconnected to ensure data integrity and consistency, facilitating data updates and retrieval through a REST API.

In summary, InfoNBA is a user-friendly tool for any basketball follower, providing precise and updated information at their fingertips while giving administrators the necessary tools to efficiently manage the database.

3.- Requisitos funcionales

Los requisitos funcionales de las aplicaciones, tanto la de usuarios como la de administradores, son los siguientes:

Aplicación para Usuarios:

1. Visualización de la Clasificación de la Liga:

- Mostrar la clasificación actual de la liga NBA, tanto a nivel general como separada por conferencias y divisiones.
- Actualización automática de la clasificación a medida que se registran nuevos resultados.

2. Listado de Equipos:

- Mostrar un listado completo de todos los equipos de la NBA.
- Permitir la visualización de información detallada de cada equipo.

3. Listado de Jugadores:

- Mostrar un listado de todos los jugadores de la liga.
- Proveer detalles individuales de cada jugador, como estadísticas personales y equipo al que pertenece.

4. Listado de Partidos:

- Mostrar los partidos ya jugados y la programación de partidos.

5. Información de Estadios:

- Proveer detalles de los estadios.
- Incluir imágenes de los estadios.

Aplicación para Administradores:

1. Gestión de Equipos:

- Permitir la creación de nuevos equipos con toda la información relevante.
- Permitir la edición de la información de equipos existentes.
- Permitir la eliminación de equipos.

2. Gestión de Jugadores:

- Permitir la creación de nuevos jugadores con toda la información relevante.
- Permitir la edición de la información de jugadores existentes.
- Permitir la eliminación de jugadores.

3. Gestión de Partidos:

- Permitir la creación de nuevos partidos con detalles como fecha, equipos participantes y resultados.
- Permitir la edición de la información de partidos existentes.
- Permitir la eliminación de partidos.

4. Gestión de Estadios:

- Permitir la creación de nuevos estadios con detalles como nombre, capacidad, ubicación y equipo local.
- Permitir la edición de la información de estadios existentes.
- Permitir la eliminación de estadios.

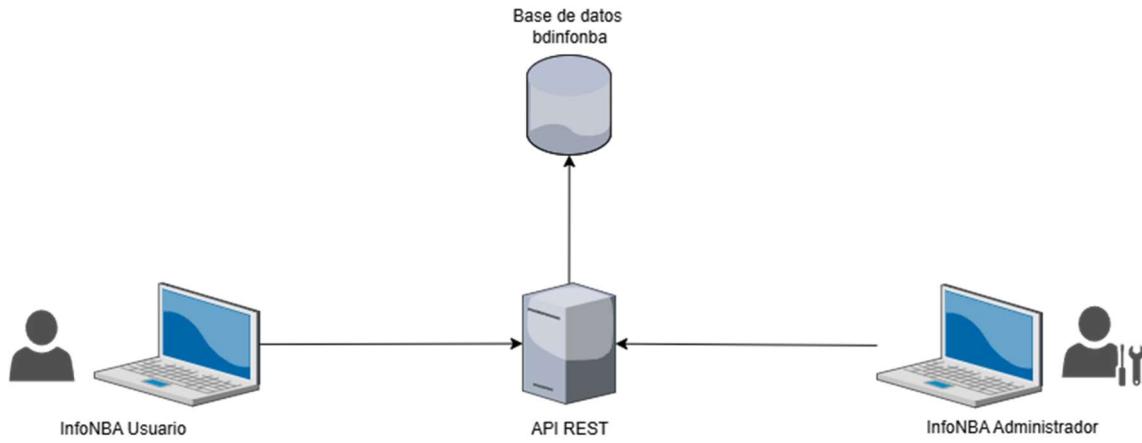
5. Actualización en Tiempo Real:

- Asegurar que todos los cambios realizados por los administradores se reflejen en tiempo real en la aplicación para usuarios.

Estos requisitos funcionales aseguran que InfoNBA proporcione una experiencia completa tanto para los usuarios que buscan información detallada sobre la NBA como para los administradores que necesitan gestionar eficientemente los datos de la aplicación.

4.- Diseños y análisis

4.1.- Diagrama de la arquitectura de la aplicación



El proyecto InfoNBA está desarrollado para uso local con un servidor Apache Tomcat. Tanto las dos aplicaciones de escritorio, como el API REST y la base de datos han sido desarrollados por mí. La base de datos almacena todos los datos en distintas tablas, que se explicarán detalladamente más adelante.

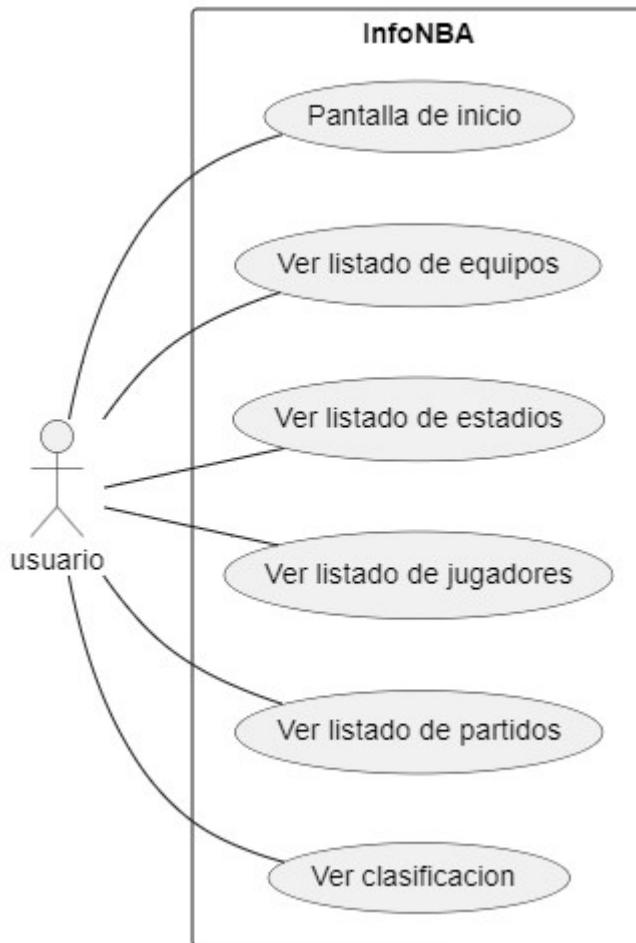
El API REST, desplegado en Tomcat, incluye peticiones GET, POST, PUT y DELETE. He desarrollado peticiones GET que devuelven listados completos, otras que devuelven un único objeto de la clase solicitada, y también peticiones que proporcionan la clasificación dividida por conferencias.

La aplicación de usuario realiza peticiones GET al API REST para cargar los listados. Todos los datos están almacenados en una base de datos SQL llamada bdinfonba. Al cargar los datos de una petición, se obtienen los listados deseados, ya sean equipos, estadios, jugadores, partidos o la clasificación.

La aplicación de administrador puede realizar peticiones POST, PUT o DELETE, dependiendo de las acciones requeridas con los datos de la aplicación. Estas peticiones pueden gestionar todos los datos presentes en la base de datos, incluyendo equipos, estadios, jugadores y partidos.

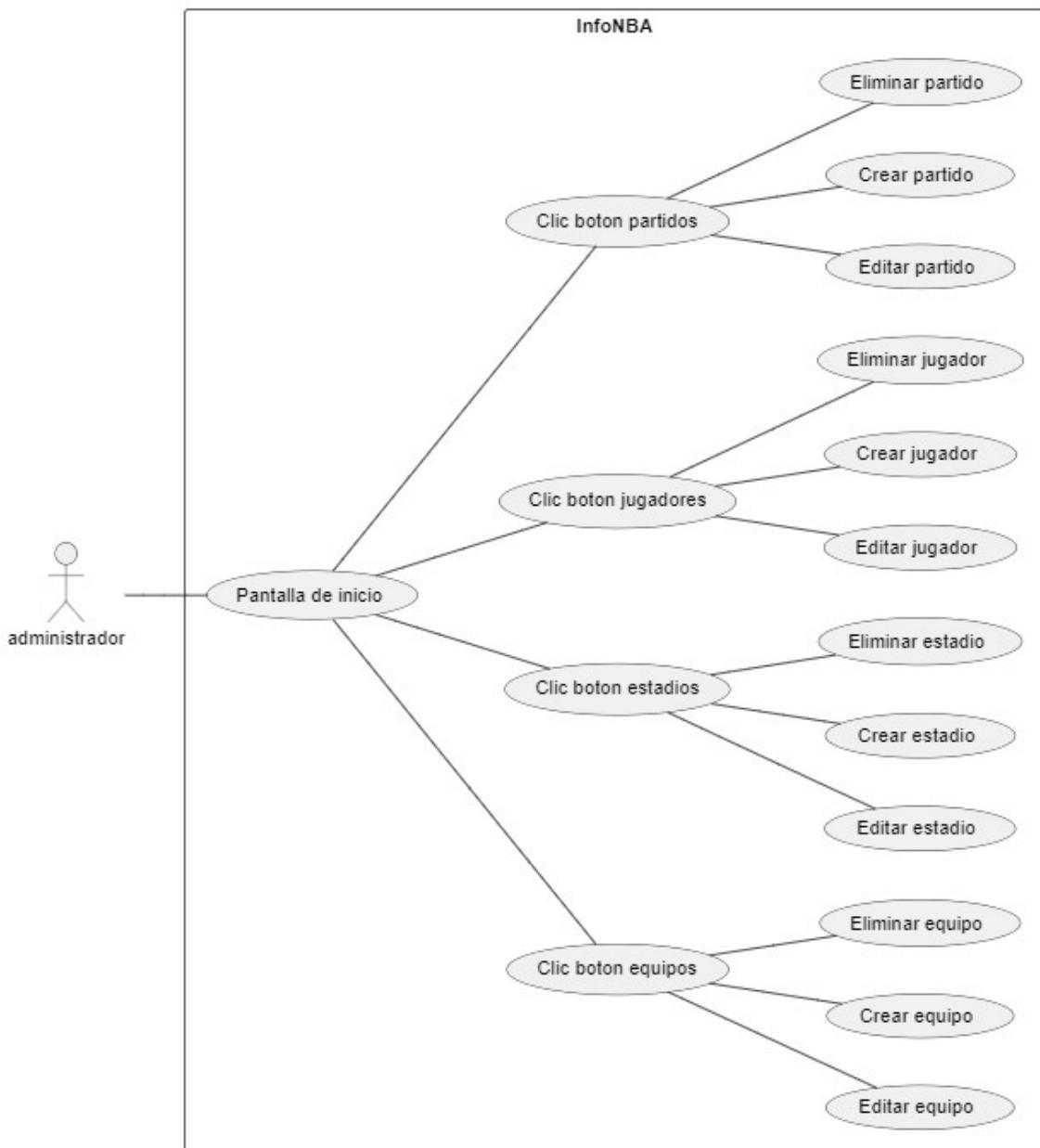
4.2.- Diagramas de casos de uso

4.2.1.- Diagrama de casos de uso de la aplicación para usuarios



Los usuarios tendrán seis casos de uso distintos, desde cualquiera de ellos podrían pasar de uno a otro sin necesidad de pasar por la pantalla de inicio, es decir, si se encuentran viendo el listado de equipos, podrían pasar directamente a ver la clasificación, por ejemplo. También desde cualquiera de los casos de uso en el que se encuentren pueden pasar a la pantalla de inicio.

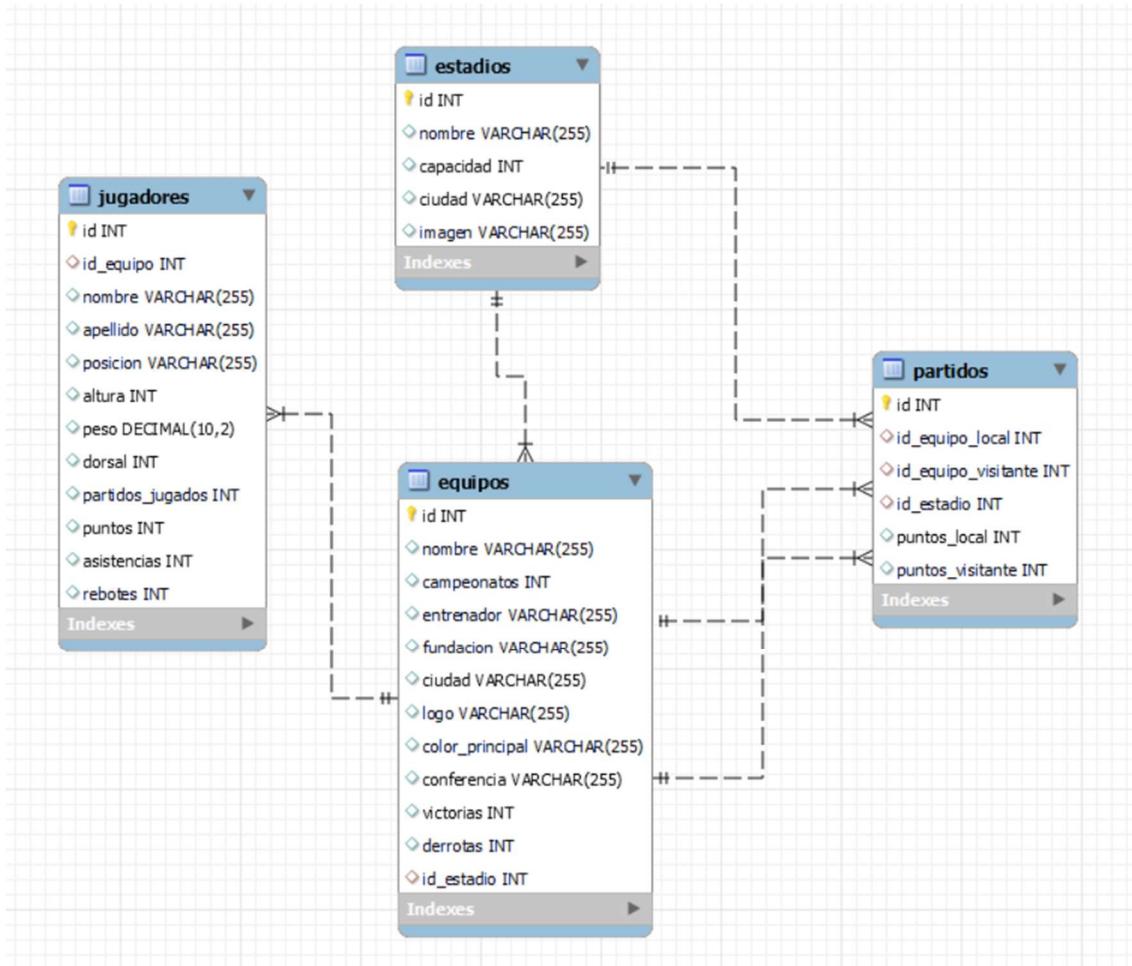
4.2.3.- Diagrama casos de uso de la aplicación para administrador



La aplicación de administrador se inicia mostrando cuatro botones, dependiendo de cual se clique se pasará a un caso de uso distinto. Entrando en cada uno se verán tres botones distintos para elegir que quiere hacer el administrador, ya sea editar, crear o eliminar alguna información.

4.3.- Diseño de datos

Los datos del proyecto se almacenan en una base de datos SQL con la siguiente estructura:



El proyecto consta de cuatro tablas en la base de datos. Todas ellas guardan la información necesaria para luego mostrarla en la aplicación de usuario. Los datos almacenados en cada tabla han variado bastante desde la primera idea del proyecto. Al principio no tenía pensado tal cantidad de campos en cada tabla, pero con el desarrollo del proyecto vi que se quedaba muy escueto y no me satisfacía lo que se podría mostrar luego, al darme cuenta añadí campos nuevos y eliminé algunos que tenía anteriormente, hasta llegar a esta versión de la base de datos.

5.- Codificación

Todo el proyecto ha sido desarrollado con tecnologías, lenguajes y entornos de desarrollo que hemos estudiado y utilizado durante estos dos años de curso, ya que era la forma de tener las menos dudas posibles sobre como utilizar todas las tecnologías y en caso de surgir dudas, podría preguntar a los profesores, que me las resolverían sin problemas.

A continuación, se explica cómo se ha desarrollado cada parte del proyecto y todo lo que se va a entregar.

Base de datos

La base de datos es un base de datos SQL que se ha desarrollado en MySQL Workbench

En el script .sql que se ha entregado está todo lo necesario para crear la base de datos con todas las tablas y todos los datos iniciales.

Lo primero que encontramos en el script, es la creación de la base de datos para utilizarla posteriormente y crear ahí todas las tablas con sus datos.

```
create database if not exists bdinfonba;
use bdinfonba;
```

Después de crear la base de datos y usarla, se crean todas las tablas necesarias con sus campos y relaciones. Estas tablas con sus datos y relaciones se pueden ver en el apartado anterior en la imagen del apartado ‘4.3.- Diseño de datos’.

```
CREATE TABLE estadios (
    id integer PRIMARY KEY AUTO_INCREMENT,
    nombre varchar(255),
    capacidad integer,
    ciudad varchar(255),
    imagen varchar(255)
);

CREATE TABLE equipos (
    id integer PRIMARY KEY AUTO_INCREMENT,
    nombre varchar(255),
    campeonatos integer,
    entrenador varchar(255),
    fundacion varchar(255),
    ciudad varchar(255),
    logo varchar(255),
    color_principal varchar(255),
    conferencia varchar(255),
    victorias integer,
    derrotas integer,
    id_estadio integer,
    FOREIGN KEY (id_estadio) REFERENCES estadios(id)
);
```

```

    ) CREATE TABLE jugadores (
        id integer PRIMARY KEY AUTO_INCREMENT,
        id_equipo integer,
        nombre varchar(255),
        apellido varchar(255),
        posicion varchar(255),
        altura integer,
        peso decimal(10,2),
        dorsal integer,
        partidos_jugados integer,
        puntos integer,
        asistencias integer,
        rebotes integer,
        FOREIGN KEY (id_equipo) REFERENCES equipos(id)
    );

    ) CREATE TABLE partidos (
        id integer PRIMARY KEY AUTO_INCREMENT,
        id_equipo_local integer,
        id_equipo_visitante integer,
        id_estadio integer,
        puntos_local integer,
        puntos_visitante integer,
        FOREIGN KEY (id_equipo_local) REFERENCES equipos(id),
        FOREIGN KEY (id_equipo_visitante) REFERENCES equipos(id),
        FOREIGN KEY (id_estadio) REFERENCES estadios(id)
    );

```

En estas capturas de código se puede ver como se crea cada tabla de la base de datos y como se establecen las conexiones entre las tablas.

Tras crear las tablas, se realizan los inserts con todos los datos que he tenido que buscar en internet para tener toda la información detallada y sin errores en la aplicación.

```

INSERT INTO estadios VALUES
(0,'TD Garden', 18624,'Boston','https://upload.wikimedia.org/wikipedia/commons/thumb/6/6a/TD_Garden_%28crop%29.JPG/480px-TD_Garden_%28crop%29.JPG'),

```

```

INSERT INTO equipos VALUES
(0,'Atlanta Hawks',1,'Quin Snyder',1949,'Atlanta','https://content.sportslogos.net/logos/6/228/full/8190_atlanta_hawks-primary-2021.png','#E03A3E','Este',36,46,6),


insert into jugadores values
(0,1,'Trae','Young','Base',185,74,11,54,25,11,3),
(0,1,'Dejounte','Murray','Escolta',193,82,5,78,22,6,5),
(0,1,'Bogdan','Bogdanovic','Alero',196,102,13,79,17,3,3),
(0,1,'DeAndre','Hunter','Ala pivot',203,100,12,57,16,4,2),
(0,1,'Clint','Capela','Pivot',208,116,15,73,12,11,1),


insert into partidos values
(0,2,28,1,117,94),
(0,2,29,1,126,97),
(0,2,13,1,96,115),
(0,3,5,2,118,109),
(0,3,11,2,106,104),
(0,3,10,2,98,109),
(0,20,27,3,126,105),

```

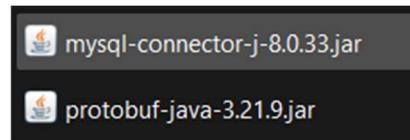
Debido a la gran cantidad de datos que hay, los inserts se hacen muy largos y ocupan muchas líneas de código, para verlos enteros hay que consultar el archivo .sql, ya que con capturas no se pueden apreciar bien todos los datos.

Con todos estos pasos realizados, se ejecuta el archivo en Workbench y tenemos la base de datos creada y con los datos necesarios para nuestra aplicación.

API REST

El API REST es un servicio web programado en java en NetBeans, que se desplegará en local en TOMCAT.

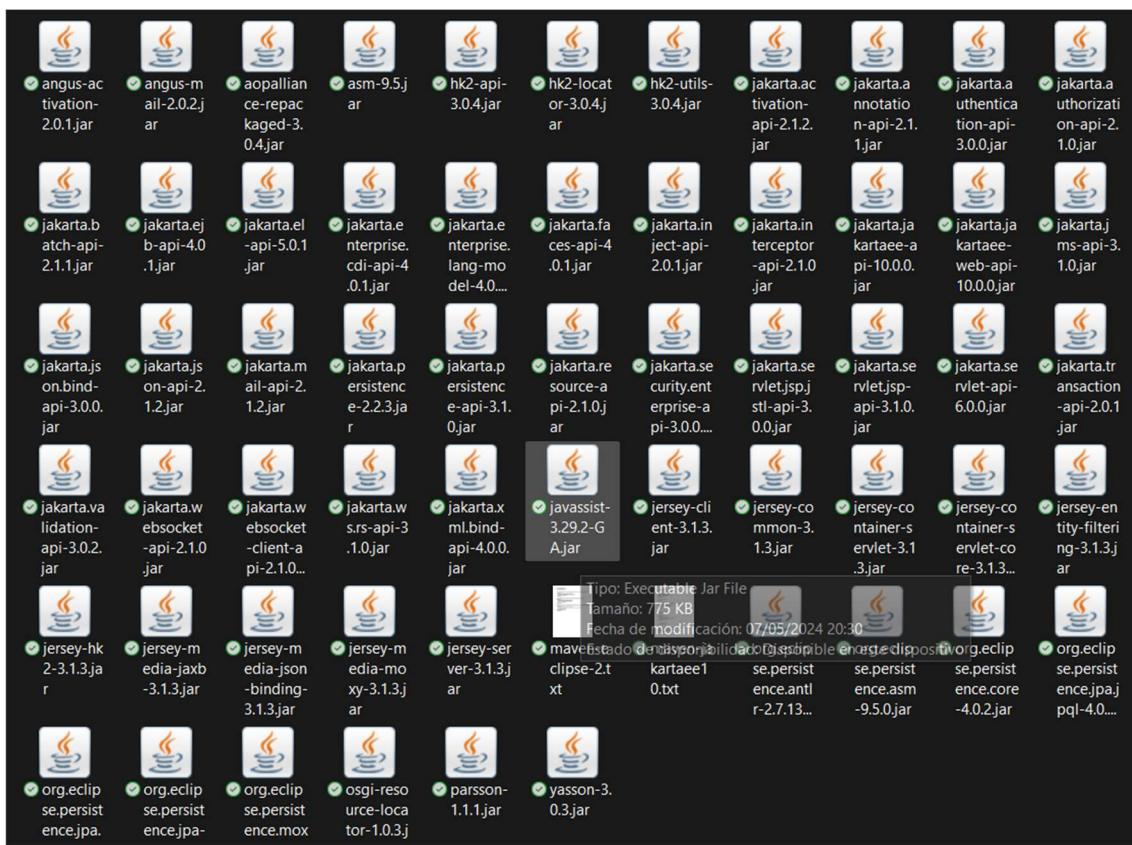
Al API REST se le han incorporado las librerías de jakartaee10, bd-mysql y json-org.



Archivos .jar de la librería jakartaee10



Archivo .jar de la librería json-org



Archivos .jar de la librería bd-mysql

Estas bibliotecas se han usado porque el API REST tiene que acceder a la base de datos creada anteriormente. Por lo que en NetBeans, antes de programar el API REST, debemos establecer una conexión a nuestra base de datos. Al crear el API REST y crear su persistence unit, estableceremos la conexión del API REST con la base de datos.

Los métodos GET, POST, PUT DELETE que se han utilizado en este API REST son muy parecidos a los que vimos en clase y siguen la misma estructura para controlar los posibles errores.

Para cada clase creada de la base de datos, he programado dos peticiones GET, una para devolver una lista completa con todas las instancias de esa clase en la base de datos, y otra que nos devuelve una única instancia de la clase a la que accedemos por el id de dicha clase. Además, únicamente para los equipos, he creado un nuevo GET, que utilizo para devolver la clasificación de cada conferencia. Lo que hago en este método en específico, es una consulta jpql con un select de todos los equipos de una conferencia que se le pasa por el path, ordenados por victorias descendenteamente.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response getAll() {
    EntityManagerFactory emf = null;
    HashMap<String, String> mensaje = new HashMap<>();
    Response response;
    Status statusResul;

    List<Equipos> lista;
    try {
        emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT);

        EquiposJpaController dao = new EquiposJpaController(emf);
        lista = dao.findEquiposEntities();
        if (lista == null) {
            statusResul = Response.Status.NO_CONTENT;
            response = Response.status(statusResul).build();
        } else {
            statusResul = Response.Status.OK;
            response = Response.status(statusResul).entity(lista).build();
        }
    } catch (Exception e) {
        statusResul = Response.Status.BAD_REQUEST;
        mensaje.put("mensaje", "Error al procesar la petición");
        response = Response.status(statusResul).entity(mensaje).build();
    } finally {
        if (emf != null) {
            emf.close();
        }
    }
    return response;
}
```

Ejemplo de GET que devuelve una lista

```

@GET
@Path("{id}")
@Produces(MediaType.APPLICATION_JSON)
public Response getOne(@PathParam("id") int id) {
    EntityManagerFactory emf = null;
    HashMap<String, String> mensaje = new HashMap<>();
    Response response;
    Status statusResul;
    Equipos equipo;
    try {
        emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT);
        EquiposJpaController dao = new EquiposJpaController(emf);
        equipo = dao.findEquipos(id);

        if (equipo == null) {
            statusResul = Response.Status.NOT_FOUND;
            mensaje.put("mensaje", "No existe departamento con ID " + id);
            response = Response
                .status(statusResul)
                .entity(mensaje)
                .build();
        } else {
            statusResul = Response.Status.OK;
            response = Response
                .status(statusResul)
                .entity(equipo)
                .build();
        }
    } catch (Exception ex) {
        statusResul = Response.Status.BAD_REQUEST;
        mensaje.put("mensaje", "Error al procesar la petición");
        response = Response
            .status(statusResul)
            .entity(mensaje)
            .build();
    } finally {
        if (emf != null) {
            emf.close();
        }
    }
    return response;
}

```

Ejemplo de GET que devuelve un único objeto

En estos dos ejemplos de GET, podemos observar la estructura que se sigue para todos los GET del API REST, cambian los valores que se devuelven dependiendo lo que queramos obtener.

```

    @GET
    @Path("/clasificacion/este")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getEste() {
        EntityManagerFactory emf = null;
        HashMap<String, String> mensaje = new HashMap<>();
        Response response;
        Status statusResul;

        String resultado = "{}";
        try {
            emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT);
            EquiposJpaController dao = new EquiposJpaController(emf);
            EntityManager em = dao.getEntityManager();
            Query query
                = em.createQuery("SELECT e FROM Equipos e WHERE e.conferencia = 'Este' ORDER BY e.victorias DESC");
            List<Equipos> lista = query.getResultList();

            if ((lista != null) && (!lista.isEmpty())) {
                JSONArray jsonArray = new JSONArray();
                for (Equipos equipo : lista) {
                    JSONObject json = new JSONObject();
                    json.put("campeonatos", equipo.getCampeonatos());
                    json.put("ciudad", equipo.getCiudad());
                    json.put("colorPrincipal", equipo.getColorPrincipal());
                    json.put("conferencia", equipo.getConferencia());
                    json.put("derrotas", equipo.getDerrotas());
                    json.put("entrenador", equipo.getEntrenador());
                    json.put("fundacion", equipo.getFundacion());
                    json.put("id", equipo.getId());

                    Estadios estadio = equipo.getIdEstadio();
                    JSONObject jsonEstadio = new JSONObject();
                    jsonEstadio.put("capacidad", estadio.getCapacidad());
                    jsonEstadio.put("ciudad", estadio.getCiudad());
                    jsonEstadio.put("id", estadio.getId());
                    jsonEstadio.put("imagen", estadio.getImagen());
                    jsonEstadio.put("nombre", estadio.getNombre());
                    json.put("idEstadio", jsonEstadio);

                    json.put("logo", equipo.getLogo());
                    json.put("nombre", equipo.getNombre());
                    json.put("victorias", equipo.getVictorias());

                    jsonArray.put(json);
                }
                resultado = jsonArray.toString();
            }

            statusResul = Response.Status.OK;
            response = Response
                .status(statusResul)
                .entity(resultado)
                .build();
        } else {
            statusResul = Response.Status.NO_CONTENT;
            response = Response
                .status(statusResul)
                .build();
        }
    } catch (Exception e) {
        statusResul = Response.Status.BAD_REQUEST;
        mensaje.put("mensaje", "Error al procesar la petición");
        response = Response
            .status(statusResul)
            .entity(mensaje)
            .build();
    } finally {
        if (emf != null) {
            emf.close();
        }
    }
    return response;
}

```

Este es el código para el GET de la clasificación de los equipos. En esta imagen se ve para la conferencia este, para la oeste se cambiaría el path del método y la condición del where en la consulta.

A continuación, se van a mostrar ejemplos de los métodos POST, PUT y DELETE, que como en el caso del GET, es para mostrar la estructura que se sigue en todas las clases.

```
@PUT
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response put(Equipos equipo) {
    EntityManagerFactory emf = null;
    HashMap<String, String> mensaje = new HashMap<>();
    Response response;
    Status statusResul;
    try {
        emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT);

        EquiposJpaController dao = new EquiposJpaController(emf);
        Equipos equipoFound = dao.findEquipos(equipo.getId());
        if (equipoFound == null) {
            statusResul = Response.Status.NOT_FOUND;
            mensaje.put("mensaje", "No existe equipo con ID " + equipo.getId());
            response = Response
                .status(statusResul)
                .entity(mensaje)
                .build();
        } else {
            equipoFound.setId(equipo.getId());
            equipoFound.setNombre(equipo.getNombre());
            equipoFound.setCampeonatos(equipo.getCampeonatos());
            equipoFound.setEntrenador(equipo.getEntrenador());
            equipoFound.setFundacion(equipo.getFundacion());
            equipoFound.setCiudad(equipo.getCiudad());
            equipoFound.setColorPrincipal(equipo.getColorPrincipal());
            equipoFound.setConferencia(equipo.getConferencia());
            equipoFound.setVictorias(equipo.getVictorias());
            equipoFound.setDerrotas(equipo.getDerrotas());
            equipoFound.setIdEstadio(equipo.getIdEstadio());

            dao.edit(equipoFound);
            statusResul = Response.Status.OK;
            mensaje.put("mensaje", "Equipo con ID " + equipo.getId() + " actualizado");
            response = Response
                .status(statusResul)
                .entity(mensaje)
                .build();
        }
    } catch (Exception ex) {
        statusResul = Response.Status.BAD_REQUEST;
        mensaje.put("mensaje", "Error al procesar la petición");
        response = Response
            .status(statusResul)
            .entity(mensaje)
            .build();
    } finally {
        if (emf != null) {
            emf.close();
        }
    }
    return response;
}
```

Ejemplo de método PUT

```

@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response post(Equipos equipo) {
    EntityManagerFactory emf = null;
    HashMap<String, String> mensaje = new HashMap<>();
    Response response;
    Status statusResul;
    try {
        emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT);
        EquiposJpaController dao = new EquiposJpaController(emf);
        Equipos equipoFound = null;
        if ((equipo.getId() != 0) && (equipo.getId() != null)) {
            equipoFound = dao.findEquipos(equipo.getId());
        }
        if (equipoFound != null) {
            statusResul = Response.Status.FOUND;
            mensaje.put("mensaje", "Ya existe equipo con ID " + equipo.getId());
            response = Response
                .status(statusResul)
                .entity(mensaje)
                .build();
        } else {
            dao.create(equipo);
            statusResul = Response.Status.CREATED;
            mensaje.put("mensaje", "Equipo " + equipo.getNombre() + " grabado");
            response = Response
                .status(statusResul)
                .entity(mensaje)
                .build();
        }
    } catch (Exception ex) {
        statusResul = Response.Status.BAD_REQUEST;
        mensaje.put("mensaje", "Error al procesar la petición");
        response = Response
            .status(statusResul)
            .entity(mensaje)
            .build();
    } finally {
        if (emf != null) {
            emf.close();
        }
    }
    return response;
}

```

Ejemplo de método POST

```

@DELETE
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Response delete(@PathParam("id") int id) {
    EntityManagerFactory emf = null;
    HashMap<String, String> mensaje = new HashMap<>();
    Response response;
    Status statusResul;
    try {
        emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT);

        EquiposJpaController dao = new EquiposJpaController(emf);
        Equipos equipoFound = dao.findEquipos(id);
        if (equipoFound == null) {
            statusResul = Response.Status.NOT_FOUND;
            mensaje.put("mensaje", "No existe equipo con ID " + id);
            response = Response
                .status(statusResul)
                .entity(mensaje)
                .build();
        } else {
            dao.destroy(id);
            statusResul = Response.Status.OK;
            mensaje.put("mensaje", "Equipo con ID " + id + " eliminado");
            response = Response
                .status(statusResul)
                .entity(mensaje)
                .build();
        }
    } catch (Exception ex) {
        statusResul = Response.Status.BAD_REQUEST;
        mensaje.put("mensaje", "Error al procesar la petición");
        response = Response
            .status(statusResul)
            .entity(mensaje)
            .build();
    } finally {
        if (emf != null) {
            emf.close();
        }
    }
    return response;
}

```

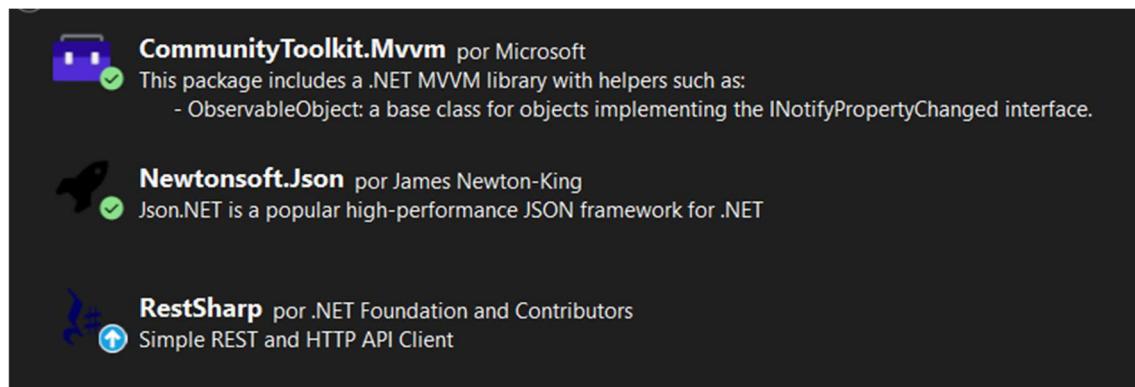
Ejemplo de método DELETE

Con todos los métodos creados, se despliega el API REST en TOMCAT y cuando iniciemos tomcat se puede acceder al API REST para comprobar que funciona correctamente. No se pueden procesar todas las peticiones desde el navegador web, por eso, en mi caso, hago uso de Postman para comprobar que todas las peticiones funcionan correctamente.

Aplicaciones de escritorio

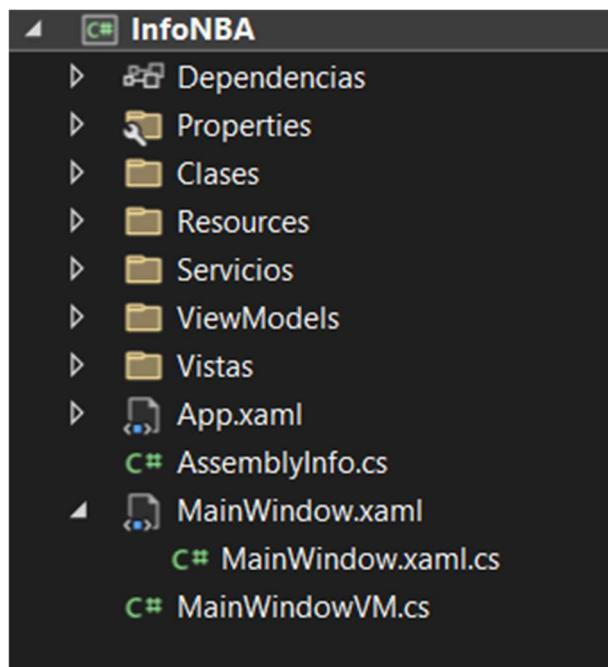
Ambas aplicaciones de escritorio, tanto la de usuario como la de administrador, se han programado en WPF desde Visual Studio.

Ya que se va a seguir el patrón MVVM y se va a acceder a una API, he tenido que instalar varios paquetes NuGet en ambos proyectos, ya que los paquetes NuGet son independientes entre proyectos.



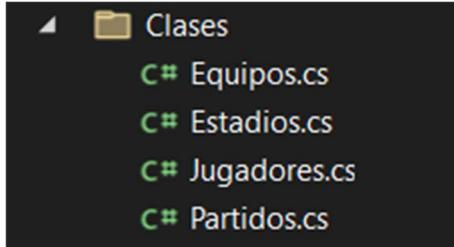
El CommunityToolkit.Mvvm es el que nos permite seguir el patrón MVVM en el proyecto. Los paquetes Newtonsoft.Json y RestSharp son los que nos permiten realizar consultas a la API y que los datos que nos devuelve la API poder serializarlos para mostrar listas o poder mandar datos en estilo Json a la API para poder llevar a cabo los POST, PUT y DELETE.

Los proyectos en WPF los he organizado los dos por carpetas y dentro de las carpetas cada archivo necesario.



Esta es la estructura de carpetas que he utilizado para los dos proyectos. No hay diferencia de estructura entre ellos, lo que cambia es el contenido de los archivos y diferentes necesidades en cada uno.

En la carpeta Clases se encuentran las mismas clases que en la base de datos y en el API REST, solo que en lenguaje C#, que es el utilizado en WPF junto a xaml.



La carpeta Resources contiene el ícono de la aplicación, se podrían añadir los logos de los equipos o imágenes que se vayan a utilizar, pero he decidido obtener todas las imágenes de internet y acceder a ellas por su url, es por eso que en la base de datos se almacenan sus urls.

En la carpeta Servicios encontramos el servicio para acceder al API REST y realizar peticiones.

```
0 referencias
public Estadios GetEstadio(int id)
{
    var client = new RestClient("http://localhost:8080/bdinfonba/infonba");
    var request = new RestRequest("/{estadios}/{id}", Method.Get);
    var response = client.Execute(request);

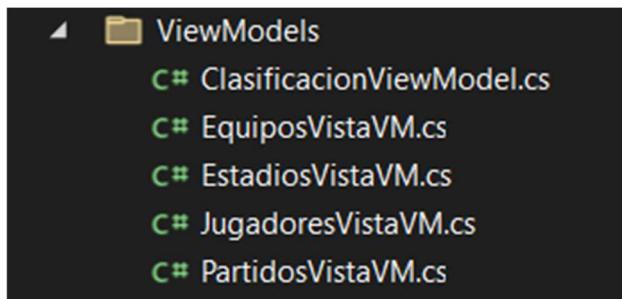
    return JsonConvert.DeserializeObject<Estadios>(response.Content);
}

1 referencia
public ObservableCollection<Estadios> GetEstadios()
{
    var client = new RestClient("http://localhost:8080/bdinfonba/infonba");
    var request = new RestRequest("/estadios", Method.Get);
    var response = client.Execute(request);

    return JsonConvert.DeserializeObject<ObservableCollection<Estadios>>(response.Content);
}
```

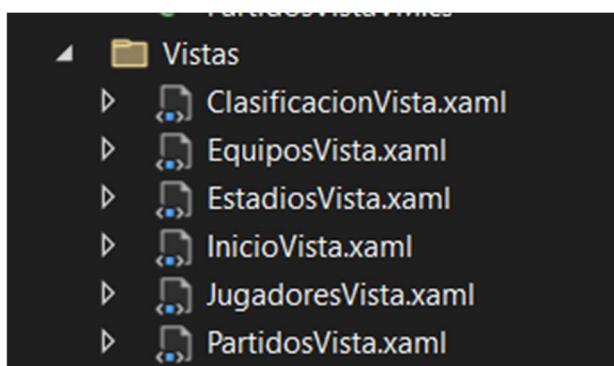
Así se realizan las peticiones al API REST desde C#. Dependiendo de la petición que queramos hacer, los métodos devolverán una lista de objetos o un único objeto.

La carpeta `ViewModels` contiene los viewmodels de cada vista de nuestra aplicación, que nos permiten seguir el patrón MVVM.



En estos ViemModels se almacenan los listados a mostrar luego en las vistas.

En la carpeta `vistas`, se encuentran las vistas de la aplicación. Hay una vista para cada opción de navegación de la aplicación.



Cada una de estas vistas está enlazada a su `ViewModel`, que es de donde obtiene los datos para mostrarlos por pantalla.

La clase `MainWindow.xaml` es la pantalla inicial de la aplicación y desde donde se controla toda la navegación de la aplicación.

6.- Manual de usuario

Ya que el proyecto InfoNBA está dividido en dos aplicaciones distintas, una para usuarios y otra para administradores en este manual de usuario se explicará el funcionamiento de cada una de ellas.

6.1.- Aplicación para usuarios

Pantalla Inicial



Esta es la pantalla que se cargará cuando los usuarios abran la aplicación.

Se puede ver la estructura que mantendrá la aplicación durante toda su ejecución, que es un menú de navegación lateral en el lado izquierdo de la pantalla, con el nombre de la aplicación arriba y las diferentes opciones de navegación que tendrá el usuario cuando utilice la aplicación.

Cuando se clique la opción a la que el usuario quiera navegar, la opción seleccionada se destacará en color azul, siguiendo la paleta de colores de la aplicación, y la vista de la opción seleccionada se cargará en el lado derecho de la aplicación, donde se sitúa en esta pantalla el logo de nuestra aplicación. El nombre de la aplicación funciona también como botón de navegación para volver a esta pantalla inicial.

Listado de equipos

InfoNBA

- Equipos
- Jugadores
- Partidos
- Estadios
- Clasificación

	Atlanta Hawks Conferencia Este 36 V 46 D 1 Títulos de la NBA Entrenador: Quin Snyder Estadio: State Farm Arena
	Boston Celtics Conferencia Este 64 V 18 D 17 Títulos de la NBA Entrenador: Joe Mazzulla Estadio: TD Garden
	Brooklyn Nets Conferencia Este 32 V 50 D 0 Títulos de la NBA Entrenador: Jacque Vaughn Estadio: Barclays Center
	Charlotte Hornets Conferencia Este 21 V 61 D 0 Títulos de la NBA Entrenador: Steve Clifford Estadio: Spectrum Center
	Chicago Bulls Conferencia Este 39 V 43 D 6 Títulos de la NBA Entrenador: Billy Donovan Estadio: United Center
	Cleveland Cavaliers Conferencia Este 48 V 34 D 1 Títulos de la NBA Entrenador: J. Bickerstaff Estadio: Quicken Loans Arena
	Dallas Mavericks Conferencia Oeste 50 V 32 D 1 Títulos de la NBA Entrenador: Jason Kidd Estadio: American Airlines Center
	Denver Nuggets Conferencia Oeste 57 V 25 D 1 Títulos de la NBA Entrenador: Michael Malone Estadio: Ball Arena

Cuando el usuario seleccione en el menú de navegación la opción ‘Equipos’, se cargará esta vista en la que se muestran los 30 equipos de la liga en una lista.

Todos los equipos en la lista tienen la misma estructura para mostrar los datos, que es, el logo del equipo a la izquierda y el color principal del equipo a la derecha. También en la parte derecha se muestran datos del equipo divididos en cuatro filas. En la primera, se muestra el nombre del equipo. En la segunda, la conferencia a la que pertenecen, las victorias y derrotas de esta temporada del equipo durante la liga regular y los campeonatos que ha conseguido en la historia de la liga. Y en las dos últimas filas se muestra el entrenador del equipo esta temporada y el estadio en el que juegan.

Desde esta pantalla, los usuarios tienen acceso al menú de navegación para poder dirigirse a cualquier ventana de la aplicación que deseen.

Listado de jugadores

The screenshot shows the InfoNBA mobile application interface. On the left, there is a dark sidebar with the 'InfoNBA' logo at the top. Below it are five menu items: 'Equipos' (highlighted in blue), 'Jugadores' (highlighted in red), 'Partidos', 'Estadios', and 'Clasificación'. The main content area displays a grid of player cards. Each card contains the player's name, team, position, weight, height, and a summary of their performance statistics. The cards are arranged in four rows: Row 1 has Trae Young and Dejounte Murray; Row 2 has DeAndre Hunter and Clint Capela; Row 3 has Derrick White and Jaylen Brown; Row 4 has Al Horford and Cam Thomas. Below these is a fifth row with Trendon Watford and Noah Clowney, followed by a sixth row with Bogdan Bogdanovic and Jrue Holiday, and a seventh row with Jayson Tatum and Mikal Bridges.

Nombre	Equipo	Posición	Peso (Kg)	Altura (cm)	Partidos jugados	PPP	APP	RPP
Trae Young	Atlanta Hawks	Base 11	74 Kg	185 cm	54 Partidos jugados	25 PPP	11 APP	3RPP
Dejounte Murray	Atlanta Hawks	Escolta 5	82 Kg	193 cm	78 Partidos jugados	22 PPP	6 APP	5RPP
Bogdan Bogdanovic	Atlanta Hawks	Alero 13	102 Kg	196 cm	79 Partidos jugados	17 PPP	3 APP	3RPP
DeAndre Hunter	Atlanta Hawks	Ala pivot 12	100 Kg	203 cm	57 Partidos jugados	16 PPP	4 APP	2RPP
Clint Capela	Atlanta Hawks	Pivot 15	116 Kg	208 cm	73 Partidos jugados	12 PPP	11 APP	1RPP
Derrick White	Boston Celtics	Escolta 9	86 Kg	193 cm	73 Partidos jugados	15 PPP	5 APP	4RPP
Jaylen Brown	Boston Celtics	Alero 7	101 Kg	198 cm	70 Partidos jugados	23 PPP	4 APP	6RPP
Al Horford	Boston Celtics	Pivot 42	109 Kg	206 cm	65 Partidos jugados	9 PPP	3 APP	6RPP
Cam Thomas	Brooklyn Nets	Base 24	95 Kg	193 cm	66 Partidos jugados	22 PPP	3 APP	3RPP
Trendon Watford								
Noah Clowney								
Jrue Holiday	Boston Celtics	Base 4	93 Kg	193 cm	69 Partidos jugados	8 PPP	5 APP	4RPP
Jayson Tatum	Boston Celtics	Ala pivot 0	95 Kg	203 cm	74 Partidos jugados	27 PPP	5 APP	8RPP
Mikal Bridges	Brooklyn Nets	Escolta 1	95 Kg	198 cm	82 Partidos jugados	20 PPP	4 APP	5RPP
Nicolas Claxton								

En esta ventana de la aplicación, se muestra un listado con todos los jugadores de la liga.

De cada jugador, se muestran datos bastante interesantes para los usuarios, divididos en cuatro filas. En la primera, se destaca el nombre y primer apellido del jugador. En la segunda, se muestra el equipo al que pertenece el jugador actualmente, junto a su posición y dorsal. En la tercera, se ve el peso de cada jugador en kilogramos y su altura en centímetros. En la última fila se muestran sus estadísticas durante la temporada, que son los partidos en los que ha jugado, y los puntos (PPP), asistencias (APP) y rebotes (RBB) que ha promediado por partido.

Listado de partidos

Equipo Local	Equipo Visitante	Estadio	Resultado	Conferencia
Boston Celtics	Toronto Raptors	TD Garden	117 - 94	Conferencia Este 64 V 18 D
Boston Celtics	Utah Jazz	TD Garden	126 - 97	Conferencia Oeste 31 V 51 D
Boston Celtics	Los Angeles Clippers	TD Garden	96 - 115	Conferencia Oeste 64 V 18 D
Brooklyn Nets	Chicago Bulls	Barclays Center	118 - 109	Conferencia Este 32 V 50 D

Cuando el usuario seleccione esta opción, se cargará un listado con los partidos que se han jugado esta temporada.

De cada partido el usuario podrá ver el estadio en el que se ha disputado el partido, que se muestra en la parte de arriba, y los equipos que se han enfrentado junto al resultado del encuentro. En este listado, el equipo local es el que se muestra en el lado izquierdo y el visitante el que se muestra en el derecho. De dichos equipos, aparte de ver el nombre, también se pueden ver datos de interés como su conferencia y las victorias y derrotas del equipo. Esta información de interés es resumida ya que si se quiere conocer más se puede mirar en el listado de equipos.

Listado de estadios

InfoNBA

- Equipos
- Jugadores
- Partidos
- Estadios
- Clasificación

			
Nombre: TD Garden Ciudad: Boston Capacidad: 18624	Nombre: Barclays Center Ciudad: Brooklyn, Nueva York Capacidad: 17732	Nombre: Madison Square Garden Ciudad: Nueva York Capacidad: 19763	Nombre: Wells Fargo Center Ciudad: Filadelfia Capacidad: 19519
			
Nombre: Scotiabank Arena Ciudad: Toronto Capacidad: 18800	Nombre: State Farm Arena Ciudad: Atlanta Capacidad: 18750	Nombre: Spectrum Center Ciudad: Charlotte Capacidad: 18800	Nombre: FTX Arena Ciudad: Miami Capacidad: 19600
			

Esta opción de la aplicación muestra un listado de cada estadio en el que juegan todos los equipos de la liga.

En este listado se muestran los datos que más pueden interesar a los usuarios, que son el nombre del estadio, su capacidad y donde están situados.

Clasificación

The screenshot shows the InfoNBA app interface. On the left, a dark sidebar menu lists 'Equipos', 'Jugadores', 'Partidos', 'Estadios', and 'Clasificación'. The 'Clasificación' item is highlighted with a blue bar. The main content area is divided into two sections: 'ESTE' (Eastern Conference) and 'OESTE' (Western Conference). Each section contains a table with three columns: 'Equipo' (Team), 'Victorias' (Wins), and 'Derrotas' (Losses). The Eastern Conference table lists 15 teams, while the Western Conference table lists 15 teams.

Equipo	Victorias	Derrotas
Boston Celtics	64	18
New York Knicks	50	32
Milwaukee Bucks	49	33
Cleveland Cavaliers	48	34
Indiana Pacers	47	35
Orlando Magic	47	35
Philadelphia 76ers	47	35
Miami Heat	46	36
Chicago Bulls	39	43
Atlanta Hawks	36	46
Brooklyn Nets	32	50
Toronto Raptors	25	57
Charlotte Hornets	21	61
Washington Wizards	15	67
Detroit Pistons	14	68

Equipo	Victorias	Derrotas
--------	-----------	----------

Esta vista de la aplicación muestra la clasificación de la liga separada por las dos conferencias existentes.

Se muestran dos tablas, una para cada conferencia, encabezada por el nombre de la conferencia. En cada tabla, la primera fila es un campo que nos explica que vamos a ver a lo largo de la tabla y tiene los títulos de nombre, que mostrará los nombres de los equipos, victorias, que mostrará las victorias del equipo, y derrotas, que mostrará las derrotas de cada equipo.

6.2.- Aplicación para administradores

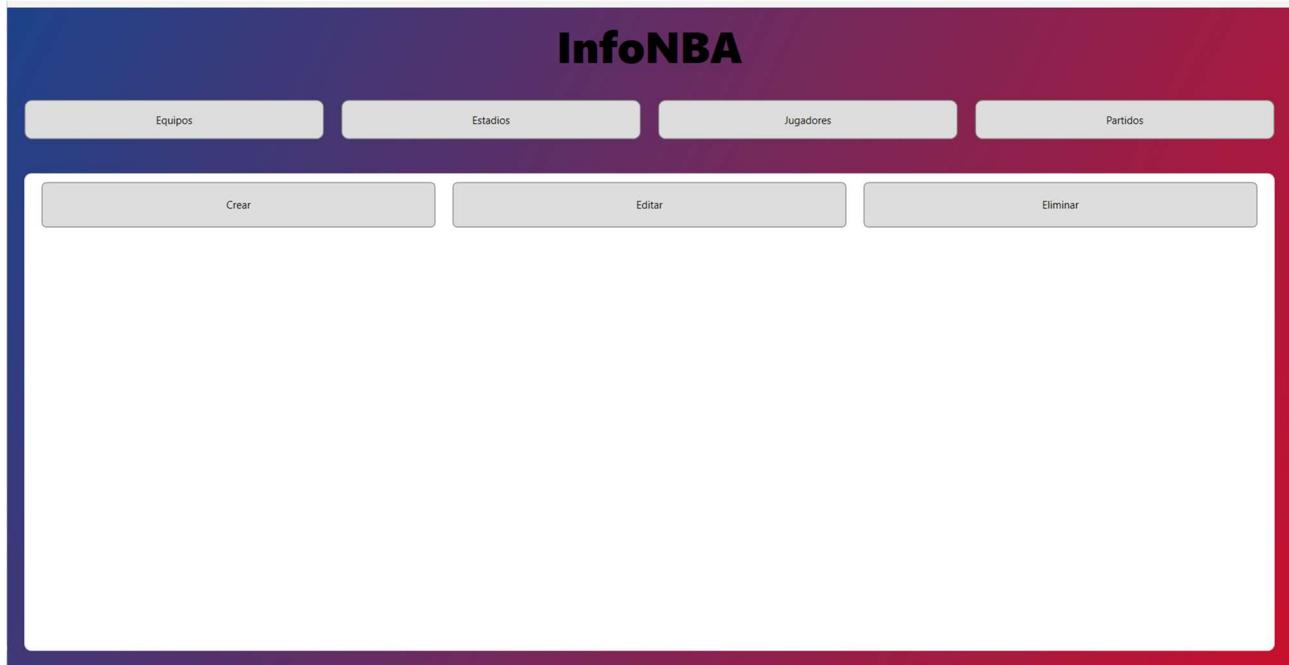
Pantalla inicial



Esta es la pantalla que ven los administradores una vez que abren la aplicación para administradores.

En esta aplicación la paleta de colores sigue siendo la misma que en la aplicación de usuario, pero a diferencia de la otra aplicación, en esta si para cambiar de vista o de opción que queramos ver ya no tenemos un menú lateral sino que tenemos una fila con cuatro botones indicándonos que datos de la aplicación queremos modificar, ya sea equipos, estadios, jugadores o partidos.

Botón de navegación clicado



Cuando se pulse cualquier botón de navegación, se cargará esta pantalla que es para los cuatro botones igual.

Esta pantalla nos muestra las opciones que tenemos para cada opción, que son crear, editar o eliminar.

Cuando cliquemos alguna de las opciones que nos aparecen, se cargará, debajo de estos botones, los datos que tenemos que llenar para poder llevar a cabo la acción seleccionada.

Botón crear clicado

InfoNBA

Equipos Estadios Jugadores Partidos

Crear Editar Eliminar

Nombre Campeonatos
Entrenador Fundacion
Ciudad Logo
Color principal Conferencia
Victorias Derrotas
Estadio
Crear Equipo

Figura 1: Botón crear equipo clicado

InfoNBA

Equipos Estadios Jugadores Partidos

Crear Editar Eliminar

Nombre Apellido
Posicion Altura
Peso Dorsal
Partidos jugados PPP
APP RPP
Equipo
Crear Jugador

Figura 2: Botón crear estadio clicado

InfoNBA

Equipos Estadios Jugadores Partidos

Crear Editar Eliminar

Nombre Capacidad

Ciudad Imagen

Crear Estadio

This screenshot shows the 'Estadios' section of the InfoNBA application. At the top, there are four main navigation buttons: 'Equipos', 'Estadios', 'Jugadores', and 'Partidos'. Below them are three secondary buttons: 'Crear', 'Editar', and 'Eliminar'. The main content area contains fields for 'Nombre' (Name) and 'Capacidad' (Capacity), along with dropdown menus for 'Ciudad' (City) and 'Imagen' (Image). A prominent 'Crear Estadio' (Create Stadium) button is centered at the bottom of the form.

Figura 3: Botón crear jugador clicado

InfoNBA

Equipos Estadios Jugadores Partidos

Crear Editar Eliminar

Equipo local Equipo visitante

Estadio

Puntos local Puntos visitante

Crear Partido

This screenshot shows the 'Partidos' section of the InfoNBA application. It features the same top-level navigation as Figure 3. The main form includes dropdown menus for 'Equipo local' (Local Team) and 'Equipo visitante' (Visiting Team), a dropdown for 'Estadio' (Stadium), and input fields for 'Puntos local' (Local Points) and 'Puntos visitante' (Visiting Points). A 'Crear Partido' (Create Match) button is located at the bottom of the form.

Figura 4: Botón crear partido clicado

Como se puede ver en las imágenes la forma en la que se muestran los datos es muy similar siguiendo la misma estructura siempre, pidiendo los datos y después de todos los datos un botón. Cuando se pulse el botón, se creara una nueva entidad de la opción que hayamos seleccionado, con los datos que se han rellenado en este formulario de creación.

Botón editar clicado

The screenshot shows the InfoNBA application interface. At the top, there is a navigation bar with four tabs: 'Equipos', 'Estadios', 'Jugadores', and 'Partidos'. Below the tabs, there are three buttons: 'Crear' (Create), 'Editar' (Edit), and 'Eliminar' (Delete). The main area is titled 'Id equipo' with a dropdown menu. It contains several input fields for team information: 'Nombre' (Name), 'Entrenador' (Coach), 'Ciudad' (City), 'Color principal' (Primary Color), 'Victorias' (Wins), 'Campeonatos' (Championships), 'Fundacion' (Foundation), 'Logo' (Logo), 'Conferencia' (Conference), and 'Derrotas' (Losses). There is also a 'Estadio' (Stadium) dropdown and a 'Editar Equipo' (Edit Team) button at the bottom.

Figura 5: Editar equipo clicado

The screenshot shows the InfoNBA application interface. At the top, there is a navigation bar with four tabs: 'Equipos', 'Estadios', 'Jugadores', and 'Partidos'. Below the tabs, there are three buttons: 'Crear' (Create), 'Editar' (Edit), and 'Eliminar' (Delete). The main area is titled 'Id estadio' with a dropdown menu. It contains several input fields for stadium information: 'Nombre' (Name), 'Ciudad' (City), 'Capacidad' (Capacity), and 'Imagen' (Image). There is also an 'Editar Estadio' (Edit Stadium) button at the bottom.

Figura 6: Editar estadio clicado

InfoNBA

Equipos Estadios Jugadores Partidos

Crear Editar Eliminar

Id jugador ▾

Nombre <input type="text"/>	Apellido <input type="text"/>
Posición <input type="text"/>	Altura <input type="text"/>
Peso <input type="text"/>	Dorsal <input type="text"/>
Partidos jugados <input type="text"/>	PPP <input type="text"/>
APP <input type="text"/>	RPP <input type="text"/>

Equipo ▾

Editar Jugador

Figura 7: Editar jugador clicado

InfoNBA

Equipos Estadios Jugadores Partidos

Crear Editar Eliminar

Id partido ▾

Equipo local ▾

Equipo visitante ▾

Estadio ▾

Puntos local

Puntos visitante

Editar Partido

Figura 8: Editar partido clicado

Esta vista es igual a la de crear, lo único que se le añade un desplegable para seleccionar que entidad queremos editar. Una vez que seleccionemos que vamos a editar con el desplegable, se rellenan los datos y cuando tengamos todo actualizado se pulsa el botón de editar.

Botón eliminar clicado

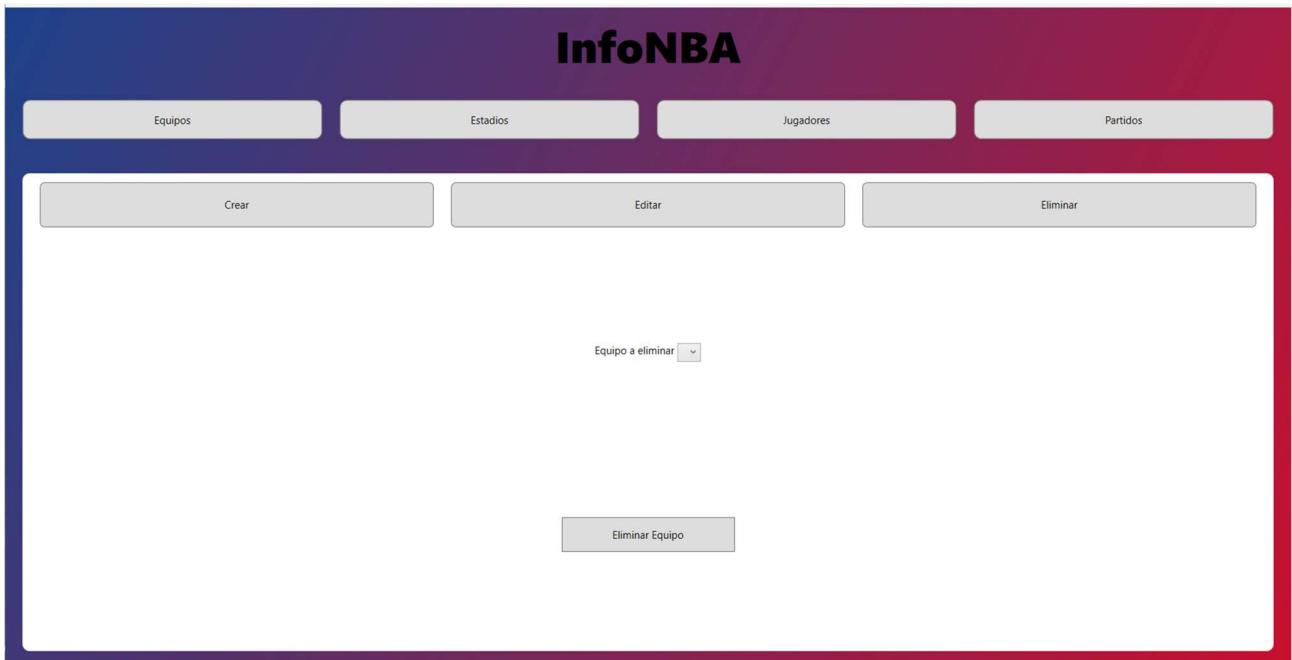


Figura 9: Eliminar equipo clicado

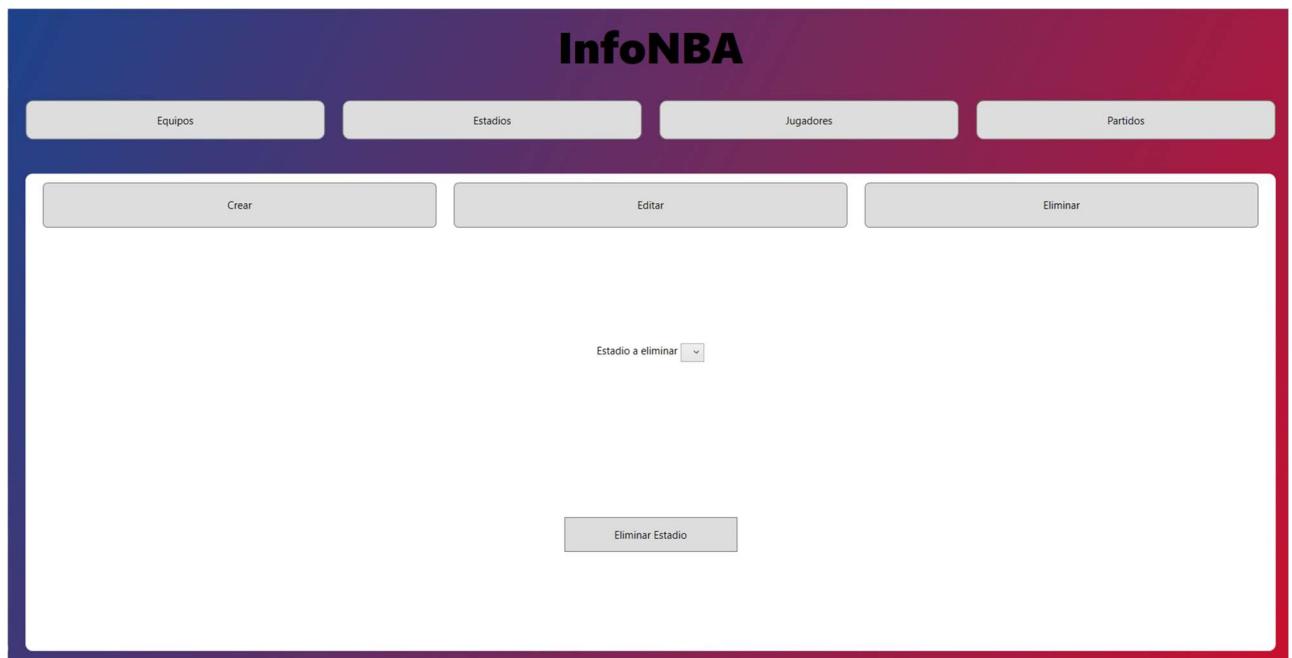


Figura 10: Eliminar estadio clicado

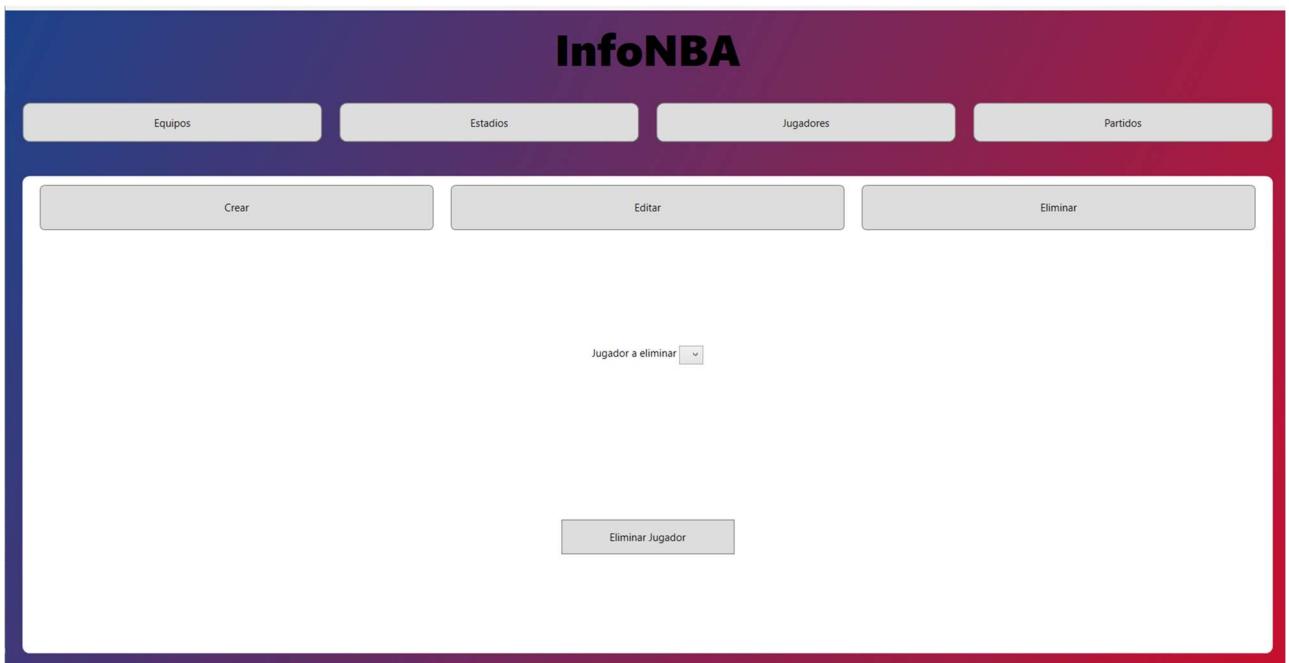


Figura 11: Eliminar jugador clicado

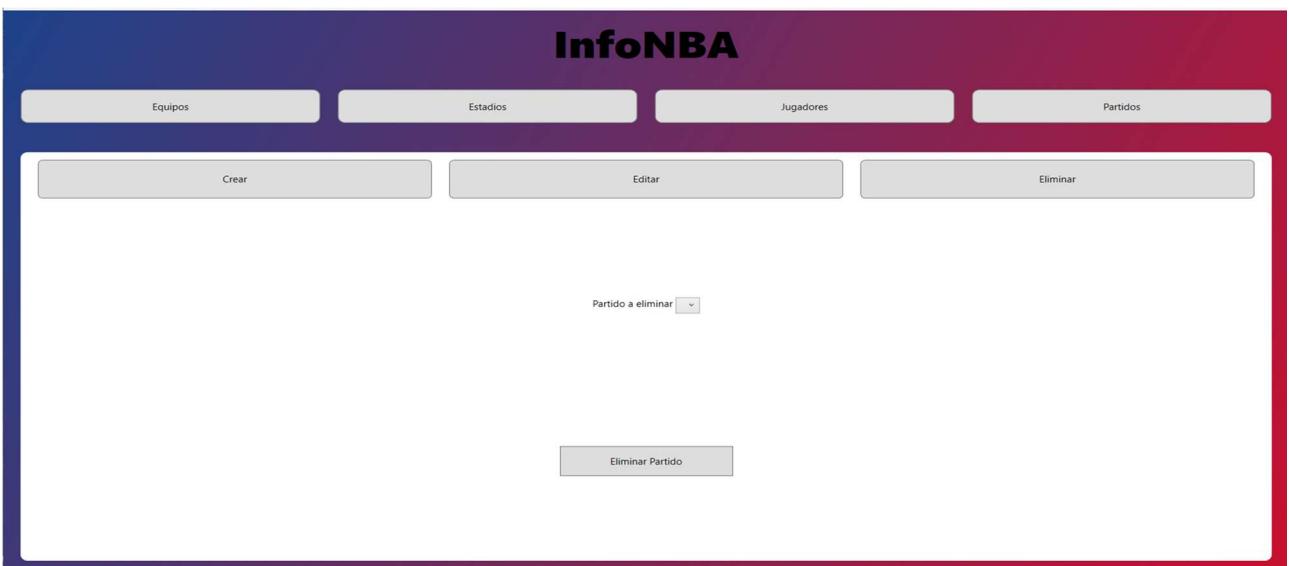


Figura 12: Eliminar partido clicado

Como se puede ver, esta vista es la más sencilla de todas, ya que solo muestra un desplegable, en el que seleccionamos la entidad que queremos eliminar, y un botón. Cuando tengamos una opción seleccionada, se pulsa el botón y se eliminará la opción que hayamos seleccionado en el desplegable.

7.- Requisitos de instalación

Para este proyecto se ha entregado:

- Script SQL
- Aplicación web en Java (API REST)
- Ejecutable de la aplicación de usuario
- Ejecutable de la aplicación de administrador

Para ejecutar la aplicación, ya sea la de usuario o la de administrador, se deben seguir algunos pasos, ya que la aplicación está diseñada y desarrollada para usarse en local.

Pasos de Instalación:

1. Cargar la Base de Datos en Local:

- Activar el servicio MySQL. Para ello, presiona las teclas Windows + S para abrir el panel de búsqueda. Escribe 'servicios' y abre la aplicación de servicios.
- Busca el servicio MySQL y actívalo si está configurado en manual. Si está en automático, se activará al encender el ordenador.
- Una vez activo el servicio, ejecuta el archivo SQL proporcionado en MySQL Workbench para cargar e iniciar la base de datos con los datos necesarios.

2. Iniciar el Servidor Tomcat:

- Recomiendo iniciar lo desde NetBeans, ya que es el método utilizado durante el desarrollo y el curso.
- Abre NetBeans y crea un nuevo servidor Apache Tomcat. Una vez creado, podrás iniciar o pausarlo desde ahí.

3. Crear una Conexión a la Base de Datos en NetBeans:

- En NetBeans, crea una conexión a la base de datos recién creada. Introduce los datos necesarios para hacer referencia a la base de datos y crea la conexión.

4. Desplegar el API REST:

- Abre el proyecto del API REST en NetBeans y ejecútalo para desplegarlo en el servidor Tomcat.

5. Ejecutar las Aplicaciones:

- Una vez realizados todos estos pasos, podrás ejecutar cualquiera de los dos ejecutables proporcionados (usuario o administrador) haciendo doble clic.
- Si ejecutas la aplicación de usuario, podrás ver todos los listados o la clasificación.
- Si ejecutas la aplicación de administrador, podrás administrar los datos de la aplicación.