**Logotipo

Descripción generada automáticamente**

**Universidad Veracruzana**

Facultad de Informática

**“Documentación del proyecto de sistema web”**

Alumnos:

Terrones Pacheco Alex Antonio

Fuentes García Edson Jair

De la Caña Pérez Marcos Yahir

Hernández Aburto Erick Adrián

Catedrático:

José Rafael Rojano Cáceres

Tecnologías computacionales, Quinto semestre

Xalapa Ver. Noviembre de 2022

Indice

[Introducción 1](#_Toc121685983)

[Frontend 2](#_Toc121685984)

[Explicación de Código 3](#_Toc121685985)

[Index.html 4](#_Toc121685986)

[Partidos.html 6](#_Toc121685987)

[Estadísticas.html 7](#_Toc121685988)

[Admin.html 8](#_Toc121685989)

[scriptIndex.js 12](#_Toc121685990)

[adminPartido.js 13](#_Toc121685991)

[ScriptIndex.js 17](#_Toc121685992)

[Accesibilidad 18](#_Toc121685993)

[Tabulaciones 19](#_Toc121685994)

[Imágenes 20](#_Toc121685995)

[Elementos 20](#_Toc121685996)

[Gráfica 21](#_Toc121685997)

[Backend 24](#_Toc121685998)

[Base de datos 24](#_Toc121685999)

[Conexión a la base de datos 25](#_Toc121686000)

[Clases 26](#_Toc121686001)

[Data Access Object 29](#_Toc121686002)

[App y Main 39](#_Toc121686003)

# Introducción

En el presente documento se describe el proceso de creación del proyecto de sistemas web, el cual consiste en desarrollar un Backend mediante el uso de Axios, Json y Java, el cual tiene una base de datos en sql, y un Frontend en HTML, Bootstrap y JavaScript, explicaremos cada una de sus características y funciones para que en caso de que se requiera se pueda modificar en un futuro, para agregar funcionalidades o simplemente comprender su funcionamiento.

La temática elegida para nuestro proyecto es un sitio web que muestra datos de los partidos de la Federación Internacional de Voleibol (FIVB) en la cual se muestra el ranking de los equipos participantes en la copa mundial de voleibol, así como los partidos que se jugaron, se están jugando y se jugaran, también una gráfica con estadísticas de los equipos y por último un CRUD (Create, Read, Update and Delete) para manipular la información de los equipos y los partidos.

El propósito de este proyecto es poner en práctica lo aprendido a lo largo del curso, emplearlo en un escenario parecido al real, en donde el Backend es desplegado en un host para que pueda funcionar y responder a las peticiones del Frontend a través de los métodos HTTP, posteriormente devolverá la información solicitada y veremos los cambios efectuados en el Frontend. Y para finalizar, también se agregarán elementos de accesibilidad a nuestro sitio, esto para que personas con dificultades visuales puedan comprender de igual manera nuestro sitio. Sin más que agregar vamos con el contenido.

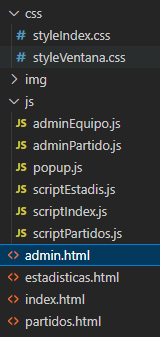
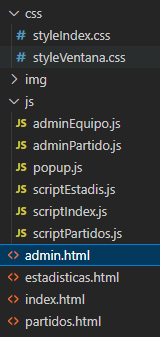
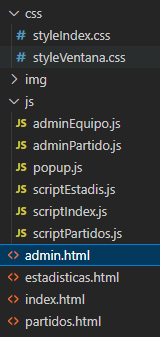
# Frontend

El frontend es toda la parte visible por el usuario y los 3 lenguajes más básicos a dominar en el frontend son HTML, CSS y JavaScript.

Para nuestro proyecto, se utilizaron varios archivos para poder concretar nuestro forntend, de los cuales 4 fueron archivos HTML, 2 fueron archivos CSS y 6 fueron archivos de JavaScript.

Los 4 archivos html se utilizaron para cada una de las ventanas que tiene nuestro sitio (Son 4), una para el inicio otra donde se muestran los partidos, otra donde se muestran las estadísticas de cada equipo; y el último apartado es el de administración, el cual es un apartado en donde se pueden visualizar, agregar, modificar y eliminar los partidos programados, así como los equipos participantes.

## Explicación de Código

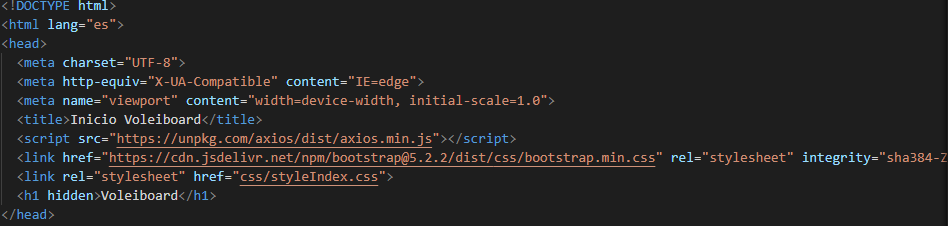
A continuación, se explicará a grandes rasgos el funcionamiento y el código de cada uno de los archivos mencionados anteriormente, así como su relación con sus archivos correspondientes.

Nuestros 4 archivos html comparten varias características entre sí. Por ejemplo, todos los archivos comparten la misma estructura de header, a excepción del archivo admin.html, que es muy similar, pero tiene un par de cosas extras. También, todos los archivos comparten la estructura del body, la cual es dividido en 2 secciones principales, una llamada “Navigation bar” y otra llamada “contenido” y el apartado de Navigation bar, se encuentra en todos los archivos, ya que es el apartado que nos permite movernos entre las distintas secciones del sitio. También, todos los archivos comparten el mismo footer, el cual es una estructura sencilla que muestra el logo y los nombres de los desarrolladores del sitio.

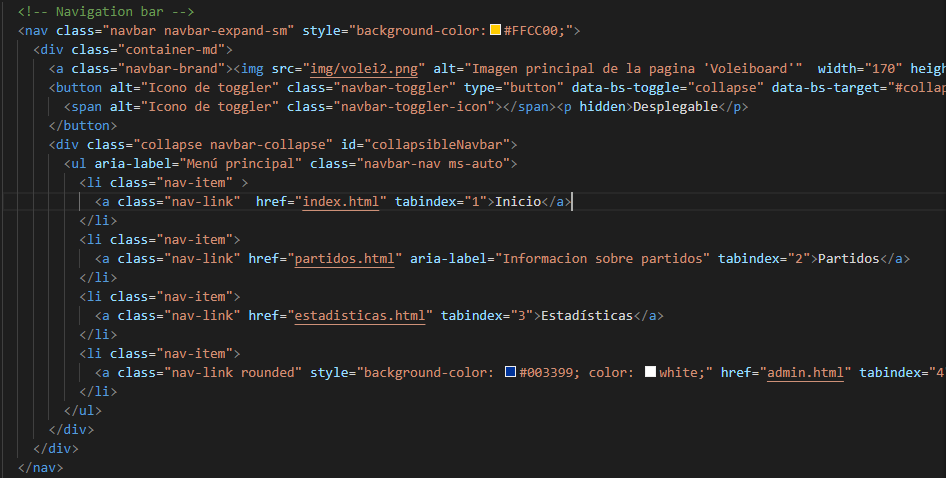
## Index.html

Captura de pantalla de un celular

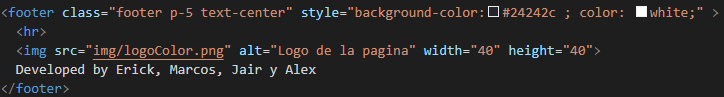
Descripción generada automáticamente

En el head del archivo implementamos un script de axios y también tenemos un enlace con el cual podemos acceder a la gran variedad de elementos de Bootstrap 5, de igual manera se vincula con un archivo css. Esto lo podemos observar con mayor claridad en la siguiente imagen.

Seguido de esta parte, tenemos el body, el cual se dividió en dos partes, una llamada “Navigation bar” y otra llamada “Contenido.” Navigation bar es la sección donde se crea la lista de opciones para navegar dentro del sistema, para poder movernos del inicio a partidos, estadísticas o a la sección de administración.

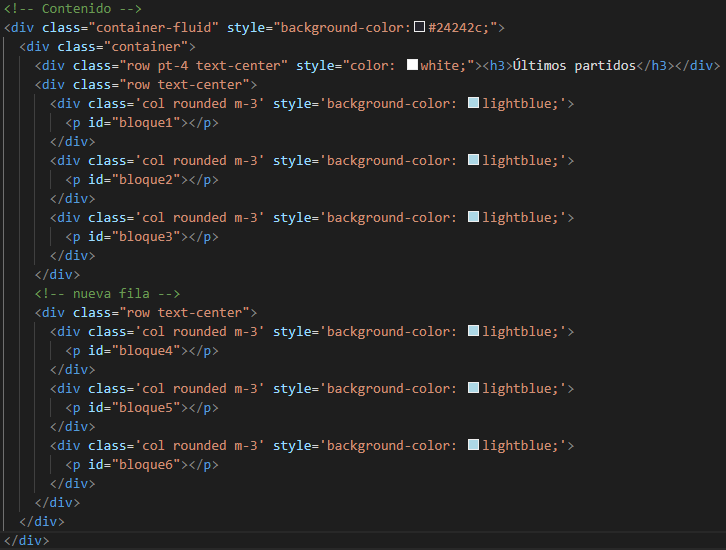


En la parte de contenido, se encuentra lo que se muestra principalmente en la página de inicio, que sería el ranking de la copa mundial, el cual muestra un top de los mejores equipos dentro del torneo, así como su puntación.

Y finalmente, en el apartado del footer, se creó un apartado sencillo con el logo del sistema y los nombres de los desarrolladores de este.

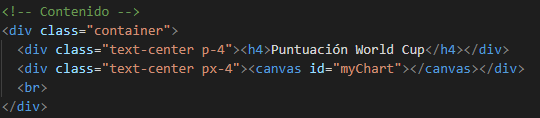
## Partidos.html

Como en el apartado del archivo anterior, ya se explicó el funcionamiento del head, la primera sección del body (Navigation bar), y el footer; los cuales son los mismos en este archivo, sólo se hará una explicación sobre la parte de “Contenido” del Body.

En este apartado de contenido se crean 6 contenedores, en los cuales se mostrarán los datos de los partidos. Dichos datos serán obtenidos de una base de datos a la cual accederemos desde el backend.

## Estadísticas.html

De igual manera que en archivo anterior, el head, la mitad del body y el footer se mantienen iguales, por lo que en la explicación que se realizará sobre este archivo será únicamente de la parte de “Contenido del body.”

Es un apartado muy sencillo visualmente, pero hay gran complejidad detrás. Lo que se hacer es llamar a un api que nos ayuda a crear una gráfica con respecto a los datos que le proporcionemos, y al igual que en el apartado anterior, estos datos son obtenidos desde una base de datos a la cual accedemos con el backend.

La implementación de esta librería genera gráficos dinámicos bitmap, con lo cual dificulta la adaptación a sistemas de accesibilidad.

## Admin.html

Captura de pantalla de computadora

Descripción generada automáticamente

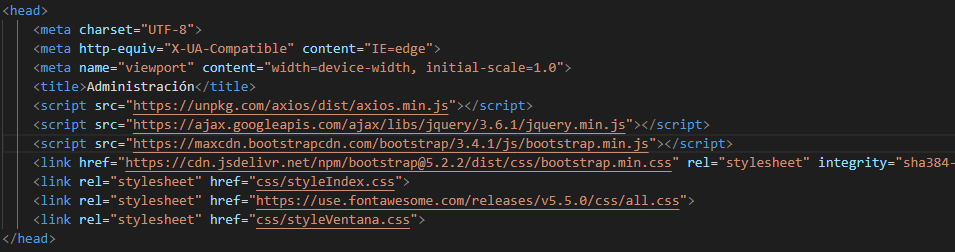
Captura de pantalla de un celular

Descripción generada automáticamente

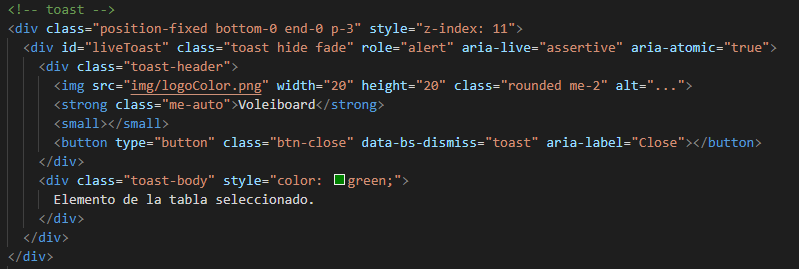
Este archivo es el más extenso, esto por todas las funcionalidades que ofrece. Como se puede observar en las imágenes anteriores, el contenido se divide en dos: una sección de Partidos que nos muestra los partidos guardados en la base de datos; y otra sección con los equipos registrados en la base de datos. Cada una de estas secciones tiene opciones de un CRUD (Create, Read, Update & Delete) que actian sobre nuestra base de datos.

Como se puede observar en la imagen de arriba, cuando seleccionamos el botón de agrega, nos despliega este popup en el cual nos indica que campos tenemos que llenar para poder agregar un partido a la base de datos. Este es el caso con los botones tanto del apartado de “Partidos” como en el apartado de “Equipos”. A continuación, se mostrará otro ejemplo mostrando el popup del botón Modificar en el apartado de “Equipos”.

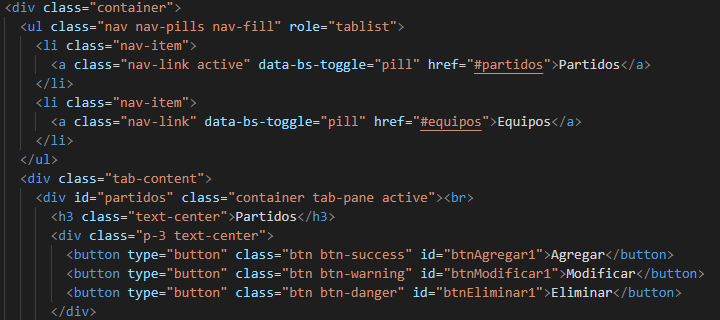
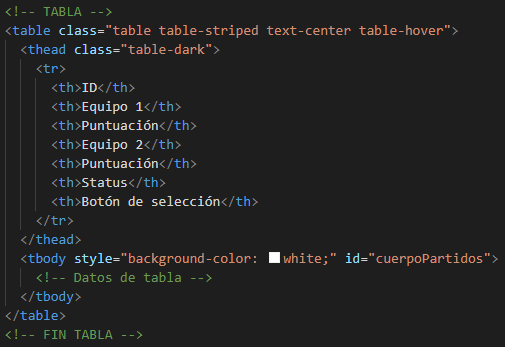
En cuanto al código, es similar a los anteriores mencionados, pero no es igual, por lo que en breve se harán las observaciones correspondientes.



A diferencia de los otros archivos, en este head, se importaron mas scripts para facilitar la implementación de varias funciones del sistema. Por ejemplo, se importó la librería jquery, la cual nos ayuda a realizar acciones más sencillas con Js para optimizar nuestro sistema. También se implementó BootstrapCDN, el cual nos permite acceder a sitios remotos de Bootstrap, los cuales almacenan funciones más recientes. Por último, se ligaron al archivo nuestros 2 archivos css. Así como también se ligó al archivo la librería de Font awesome, la cual nos permite utilizar imágenes que podemos manipular utilizando css.

Seguido de esto, el body fue dividido ahora en 3 partes. La primera es un apartado donde se crea un mensaje que se vera en pantalla cada que el usuario presione un botón azul que diga “Seleccionar”

Seguido de esto, pasamos a la segunda parte del body, la cual, si coincide que es igual con los demás archivos, ya que esta parte es fundamental para que el usuario pueda navegar entre las opciones de nuestro sistema. La sección de la cual estamos haciendo referencia es a la sección del “Navigation bar”.

Aquí, se crea un contenedor comuna lista de dos elementos, estos serán Partidos y Equipos, esto para poder alternar entre las dos opciones. Después, se crearán los botones de Agregar, modificar y eliminar.

En este apartado se crea la tabla de contenido, que nos mostrará los datos almacenados en nuestra base de datos. Los datos mostrados serán el ID del partido, el nombre del equipo 1 con su respectiva puntuación, nombre del equipo 2 con su respectiva puntuación, el estatus del partido (en emisión, finalizado, por comenzar) y un botón de selección.

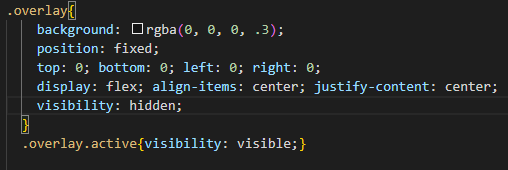
Todo esto que se acaba de mostrar fue para el apartado de partidos, después se lleva a cabo el mismo proceso para el apartado de Equipos, también se agrega los botones de agregar, modificar y eliminar, así como también se agrega una tabla de contenido donde se muestran los datos de la base de datos con respecto a la información de los partidos.

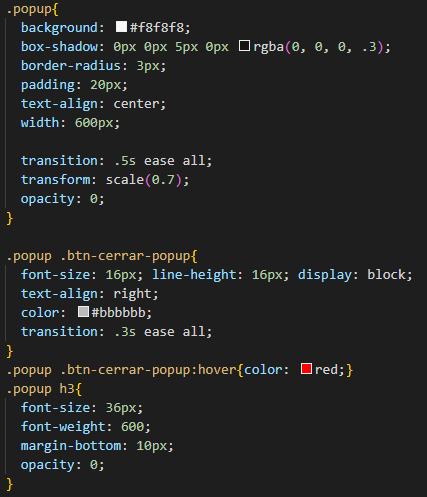
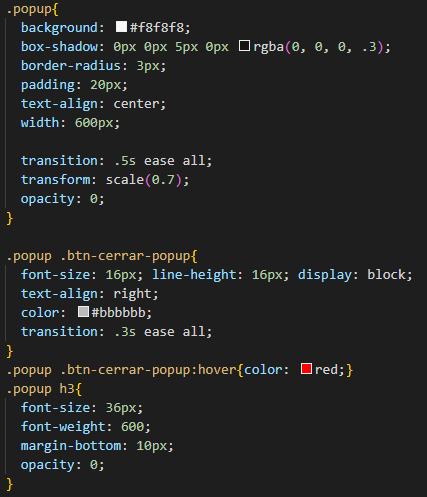
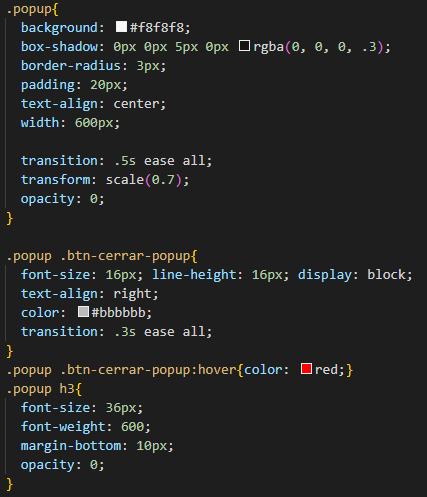
Para la parte final del contenido, se realizó el diseño de 6 ventanas popup para que el administrador ingresara los datos necesarios para realizar una acción sobre la base de datos (agregar, modificar y eliminar), esto para el apartado de Equipos como para el de Partidos. Todas tienen una estructura similar, sólo se diferencian por la cantidad de datos solicitados, lo cual depende de la consulta que se está solicitando.

## styleIndex.htm

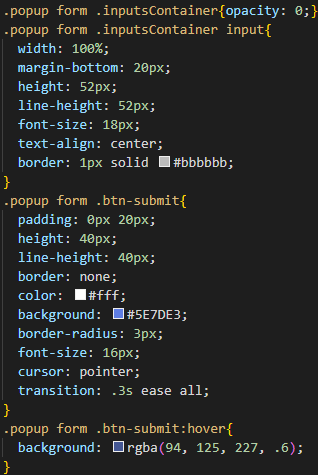
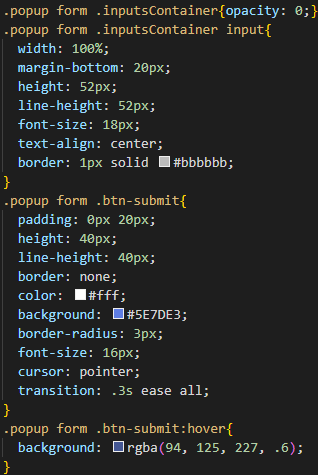
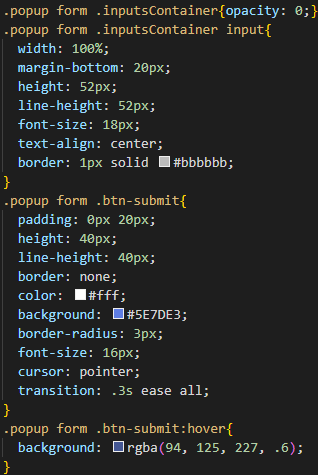
Este archivo es muy sencillo, únicamente es para darle un estilo único a las letras que aparecen en el navBar de todas las ventanas del sistema.

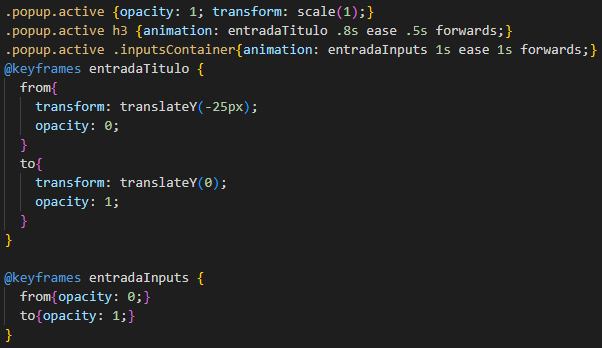
## styleVentana.css

En este archivo se encuentran las especificaciones requeridas para poder darle estilo a las ventanas emergentes (popup) que son llamadas en la sección de Administración

Primero, tenemos el overlay, el cual es para darle un fondo un poco oscuro a toda la pantalla, simulando un efecto de que está bloqueada. Aquí fue fundamental colocarla como hidden, para que no se viera en todo momento. Para que se vea la ventana, el usuario seleccionará una de las opciones y cambiará el estado del overlay a active. Una vez que el overlay esté en active, su visibilidad cambiará a visible.

En este apartado, se crea la ventana emergente donde el usuario colocará los datos para interactuar con la base de datos. También se establece el diseño de su icono, el cual utilizaremos como botón para cerrar la ventana. Seguido de esto, tenemos el diseño que se utilizara para la parte del título de la ventana



Aquí se le da diseño a los inputs en los cuales el administrador ingresará la información. También se le da diseño al botón que utilizará el administrador para aceptar y hacer los cambios correspondientes a la base de datos.

Finalmente, tenemos este apartado en el cual le daremos unas pequeñas animaciones a nuestra ventana, estas animaciones se activarán cuando la ventana pupup tenga un estado “active” el cual se le otorgará cuando el administrador seleccione alguno de los botones que accionan estas ventanas. Donde dice @keyframes es similar a una función, la cual se manda a accionar cuando el popup está en estado active. En esta función se agrega un from-to, donde en from pondremos el estado inicial del elemento y en el “to” pondremos como terminará.

## scriptIndex.js

Texto

Descripción generada automáticamente

Este script contiene la consulta al backend para el llenado de la tabla desde la base de datos del ranking. En este caso se hace un recorrido con una estructura “for let in” desde la función “tablaEquipos()” y se asigna los valores a la tabla previamente declarada desde el archivo index.html, para esto realiza un llamado de elemento mediante la id que tiene indicada mediante la instrucción “getElementById()”

## scriptPartido.js

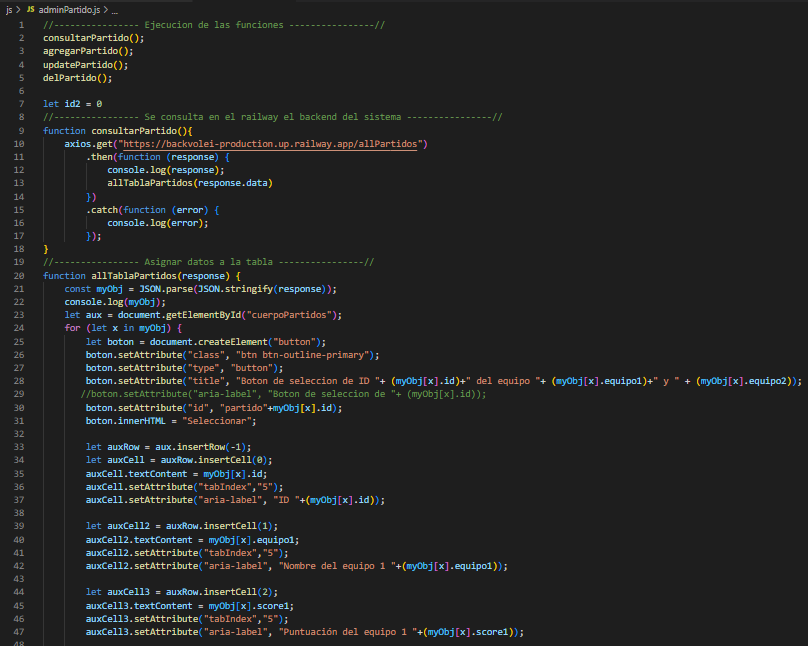
Texto

Descripción generada automáticamente

El script de partido es el encargado de gestionar la estructura de los últimos seis partidos. Mediante una combinación de JavaScript y HTML podemos obtener la estructura en que se visualiza los equipos contendientes, el botón de ver más y las opciones de accesibilidad correspondientes.

En el método del recorrido se puede apreciar los métodos de inserción en el documento HTML.

## adminPartido.js





Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

En este script podemos ver implementada las acciones Create, Read, Update y Delete (CRUD). Mediante la consulta (método “.get”) se accede a la base de datos remota mediante railway y con la inserción (método “.post”) podemos realizar los cambios pertinentes para el apartado de partidos. Para instanciar lo elementos es necesario mandarlos a llamar desde variables mediante la id predefinida desde HTML con el método “getElementById()”

## adminEquipo.js

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

En este script podemos ver implementada las acciones Create, Read, Update y Delete (CRUD). Mediante la consulta (método “.get”) se accede a la base de datos remota mediante railway y con la inserción (método “.post”) podemos realizar los cambios pertinentes para el apartado de equipos. Para instanciar lo elementos es necesario mandarlos a llamar desde variables mediante la id predefinida desde HTML con el método “getElementById()”

## scriptEstadis.js

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Este script es el responsable de organizar la información de la base de datos remota a manera de representar los datos contenidos en forma de una gráfica de barras. Como tal, al ser una librería, no posee métodos de accesibilidad, pero podemos solventar esta ausencia de una manera rudimentaria pero funcional.

## popup.js

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Esta clase es la isntanciadora de los eventos determinantes como ventanas emergentes dentro del sistema, esto como medio retroalimentación a los usuarios de interfaz grafica y que manda avisos pertinentes acorde a las acciones que éste lleva a cabo

# Accesibilidad

Para evaluar la accesibilidad de nuestro sistema fue necesario la incorporación de una extensión llamada WAVE Evaluation Tool. Con esta herramienta podemos visualizar los errores y advertencias de accesibilidad que posee nuestro sistema web, de una manera gráfica.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Cómo asistente de voz, se efectuó la instalación de un sistema de terceros de código abierto llamado NVDA, el cual permite al usuario tener en su equipo un sistema de narración para interactuar con los componentes gráficos de los sistemas de cómputo.



### Tabulaciones

La navegación por bloques mediante la estructura HTML predefinida es necesaria al momento de definir la interacción de usuarios, para aquellos casos en que se presente un usuario con alguna deficiencia motriz, o simplemente se trate de alguna persona que lleve tiempo manejando el sistema y desee agilizar los tiempos de interacción con la plataforma.

Es por esto por lo que fue necesario implementar la funcionalidad “tabindex” dentro de la sintaxis de nuestro sistema. Debido a que la estructura de nuestra propuesta se encuentra programada en HTML, CSS y JSON, el atributo se debe instanciar de las siguientes maneras:

Para establecer la estructura dentro del archivo HTML, se debe colocar dentro de los símbolos de agrupación “mayor que – menor que”, esto funcionaria para cualquier tipo de estructura o bloque dentro de nuestra clase, en el caso que se muestra en la imagen a continuación es un ejemplo de la sintaxis aplicada en un elemento de tipo lista, la cual pertenece al menú de navegación superior.

Texto

Descripción generada automáticamente

Dentro de las comillas debe ir indicado el orden en que se efectuará el recorrido de las tabulaciones ( 1 – 2 – 3 – 4 - … ) aunque igual, la sintaxis puede contener un 0 el cuál interpretará como un elemento del recorrido omitido al inicio de la secuencia, y que en el momento en que no se detecte un valor secuencial lo recorrerá. De igual manera, varios elementos pueden poseer el mismo valor indicativo de recorrido; puede existir dos elementos que posean un “tabindex” igualado con 4, y el recorrido se realizará acorde a la manera en que sean cargados los elementos en la estructura de la página web.

Al momento de ejecutar los elementos generados por script JSON, es necesario ejecutar la implementación del “tabindex” como si de un método de clase se tratara.

Texto

Descripción generada automáticamente

En la imagen anterior podemos observar un recorrido para asignar desde el script valores y celdas a una tabla que se mostrará en la página principal, para signar el atributo tabindex es necesario instanciar la celda y mandar a llamar el método “setAttribute()”, en el cuál podremos insertar el tipo de atributo y el contenido del mismo que se desea implementar separados por una coma.

### Imágenes

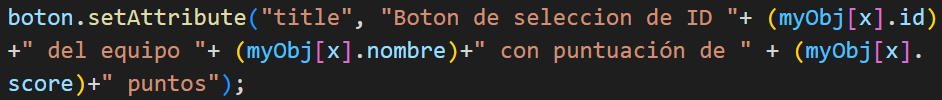
Para aplicar accesibilidad a las imágenes se emplea la etiqueta “alt” dentro de la estructura html, la cual, funcionará como descripción en caso de que la sintaxis no localice el elemento referenciado dentro del directorio especificado.



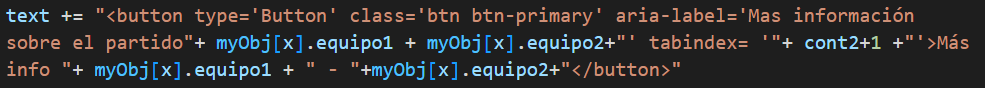
Al momento de que se pase el cursor sobre la imagen, aparecerá una leyenda que indicará lo que se asigne al recuso, y de igual manera, permitirá a las personas con alguna discapacidad visual entender el contexto y uso del elemento en que se encuentran posicionados

### Elementos

Para el resto de los elementos, como serían botones, celdas de tablas y etiquetas se hace uso de un par de atributos diferentes a los anteriores. En primer lugar, tenemos la implementación del atributo “title”, el cual, de la misma manera que la etiqueta “alt”, permite añadir una descripción flotante a los elementos que componen nuestra página.



Por otro lado, para el apartado de etiquetas referentes a botones es posible aplicar “aria-label”, el cual contiene el texto entre comillas de lo que el narrador detectará. Es posible concatenar mediante, JavaScript variables para generar contenido dinámico desde base de datos locales o remotas, como lo podemos ver a continuación



En caso de añadir estas variables desde una función de scripts JSON es necesario mandar a llamar al método “setAttribute” sobre el nombre del elemento, como se puede ver a continuación.



De igual manera podemos observar que mediante la implementación de JavaScript podemos concatenar variables dinámicas para personalizar el mensaje y brindar mejor retroalimentación sobre la posición del usuario.

### Gráfica

Para la gráfica, debido a que como se mencionó previamente, la librería myChart no posee una manera específica de instanciar mediante las barras pintadas el contenido para que lo detecte el narrador, es por esto que mediante el uso de la variable general en la que se genera, se añadió la información del contenido, al igual que se le anexó un tabindex para mejorar la navegación.

Texto

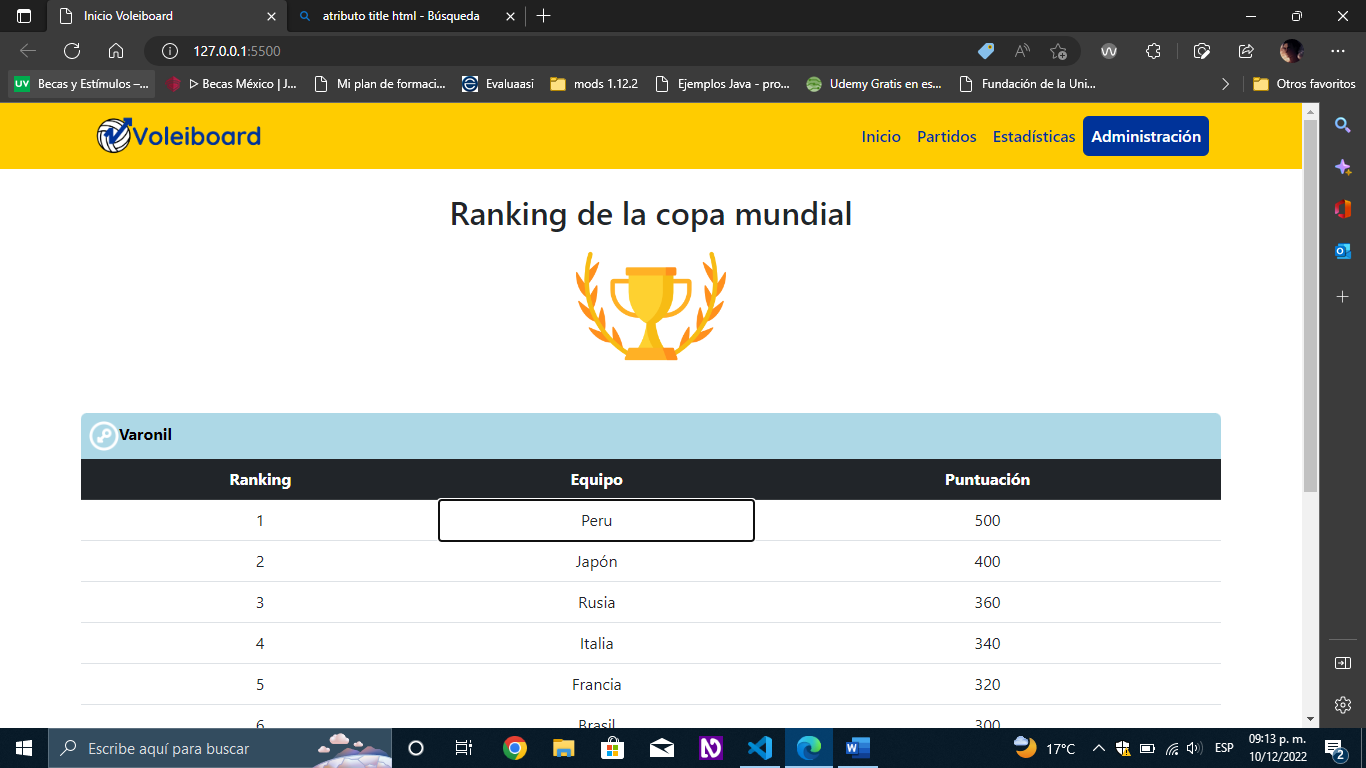
Descripción generada automáticamente

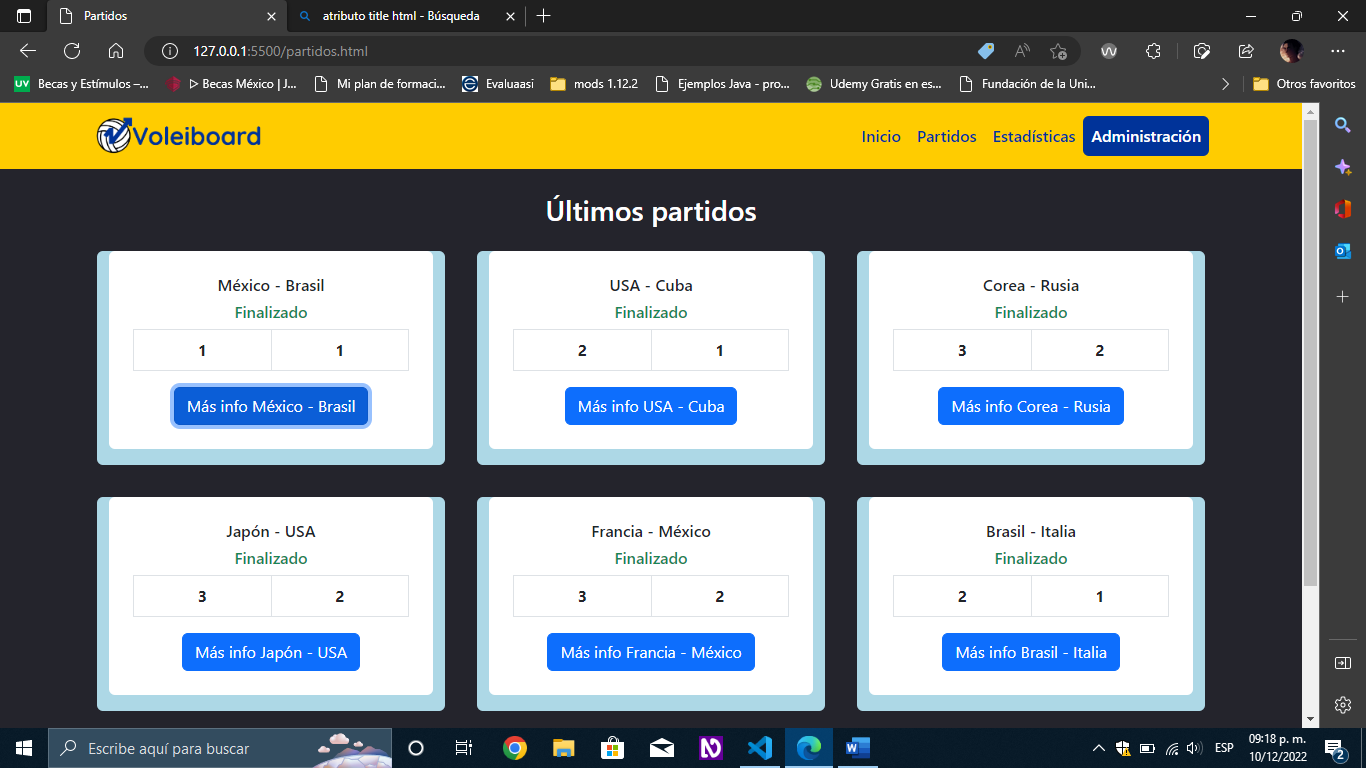
Texto

Descripción generada automáticamente

El resultado obtenido fue el siguiente:

1. Navegación con tabulación (recuadro negro)

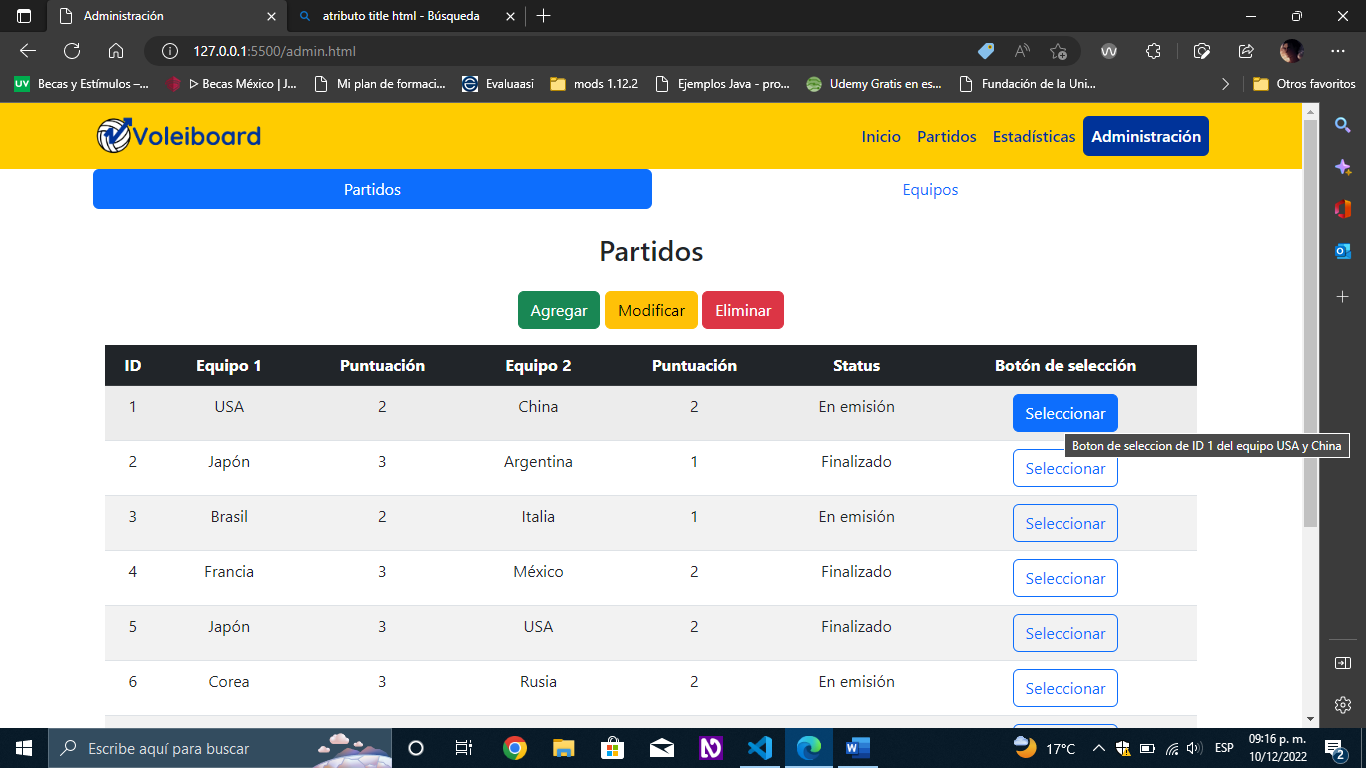




Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

1. Evento emergente con el contenido establecido dentro del “Title” / “alt”



# Backend

El backend es el encargado de procesar toda la información que alimenta a un frontend. Se compone de marcos, bases de datos o códigos. Para que un sitio web o aplicación opere efectivamente, se requiere mucha información y datos que se almacenan en «la parte trasera» de un sistema informático. En oposición al frontend, el usuario no puede ver o acceder a esta información.

# Base de datos

Nuestra base de datos esta alojada en “db4free.net” una aplicación web la cual permite crear bases de datos con phpMyAdmin para realizar pruebas y es perfecto para nuestro caso, así que nos registramos y creamos la base de datos con dos tablas, una para los “Equipos” la cual tiene como atributos el “nombre” del equipo, la “puntuación” y por supuesto un “id” único para identificar y manipular más fácilmente las tuplas como se muestra en la Fig. 1.

Texto

Descripción generada automáticamente

Fig. 1 Script de la tabla "Equipos"

Posteriormente creamos nuestra segunda tabla la cual corresponde a los partidos, esta tabla cuenta con los atributos “equipo1”, “score1”, “equipo2”, “score2”, “status” y “id”, como indica el nombre el atributo esta tabla muestra a los equipos participantes de partido, las puntuaciones que llevan y el estatus actual del partido, el script se muestra a continuación en la Fig. 2.

Texto

Descripción generada automáticamente

Fig. 2 Script de la tabla "Partidos"

# Conexión a la base de datos

Con la base de datos ya creada y lista para trabajar procedimos a crear la conexión, como estamos trabajando con Java para el código del Backend utilizamos los drivers correspondientes, el código se ve a continuación en la Fig. 3.

Texto

Descripción generada automáticamente

Fig. 3 Script conexión a la base de datos en Java

En el String “driver” ponemos el nombre de los drivers para conectar Java con mysql, en “url” guardamos el link de nuestra base de datos, en este caso se coloca “processBuilder.environment().get(“NOMBRE DE VARIABLE LOCAL”);” porque esa información se declarara en una variable local de nuestro host “Railway.app” igualmente con el “user” donde guardamos el nombre del usuario con el que accederemos a la base de datos y la password se guarda en “pass”.

Se define un método estático para poder acceder a el sin crear una instancia de la clase, este método retorna la conexión de tipo “Connection”, a través del método “getConnectión()” del DriverManager obtenemos dicha conexión, para ello tenemos que mandarle los datos previamente guardados, todo dentro de un “try/catch” para atrapar las excepciones en caso de que las haya.

# Clases

Como nuestra base de datos almacena “Equipo” y “Partido” definimos sus clases, así nos facilita su manipulación y trabajamos de manera más organizada, en la Fig. 4 se observa cómo está conformada la clase.

Texto

Descripción generada automáticamente

Fig. 4 Script de la clase equipo

Los atributos que contiene esta clase son “id”, “nombre” y “score” como se mencionó previamente en la creación en la base de datos, se definió su constructor y sus getters y setter, finalmente se sobre escribió su método “toString()” para que retornara todos los datos y ver el contenido más fácilmente al crear una instancia de esta clase.

La siguiente clase es “Partido” la cual se muestra en la Fig. 5 y la Fig. 6.

Texto

Descripción generada automáticamente

Fig. 5 Primera parte del script de la clase Partido.

Se definió su constructor y los getters y setters para cada uno de sus respectivos atributos que son “id”, “equipo1”, “score1”, “equipo2”, “score2” y “status”.Texto

Descripción generada automáticamente

Fig. 6 Segunda parte del script de la clase Partido.

# Data Access Object

Posteriormente para almacenar los datos en la base de datos creamos un DAO (Data Access Object) para manipular de manera más organizada la información, en esta clase se encuentran todos los métodos que obtienen o ingresan datos en nuestro almacenamiento, en este caso nuestra base de datos, el código se muestra a continuación en la Fig. 7.

Texto

Descripción generada automáticamente

Fig. 7 Script del metodo "GetAllEquipos" de la clase DAO.

Nuestro primer método llamado “GetAllEquipos” se encarga de obtener todas las tuplas de la tabla “equipos” de nuestra base de datos, a través de la consulta “SELECT \* from equipos”, y almacena dicha información en un objeto de la clase “Equipo” el cual es almacenado en una lista llamada “resultado”, para ejecutar la consulta tenemos que almacenar la instrucción en un String, crear un Statement con la conexión que obtenemos de la clase “Conexion” a través del método “getConnection()”, luego ejecutamos la instrucción con el método “executeQuery()” del Statement, obtendremos un ResultSet el cual contiene las tuplas de la tabla, a través de un ciclo For creamos un objeto de la clase “Equipo” por cada tupla y lo ingresamos en la lista, todo debe ir dentro de un “try/catch” en caso de que se presente una excepción, y finalmente se cierra el ResultSet con el método “close()”, igualmente se cierra el Statement y finalmente la conexión, este metodo retorna la lista con los objetos de tipo “Equipo”.

El siguiente método se llama “GetEquipos” el cual realiza lo mismo que el método “GetAllEquipos” con la diferencia que la consulta es diferente en el aspecto a que los ordena de manera descendente según el “score” y se limita a solo devolver las diez primera tuplas, en la Fig. 8 se puede observar el método.

Texto

Descripción generada automáticamente

Fig. 8 Script del método "GetEquipos" de la clase DAO.

Para ingresar información a la tabla de “Equipos” se creó el método “agregarEquipo()” que se muestra en la Fig. 9.

Texto

Descripción generada automáticamente

Fig. 9 Método "agregarEquipo" de la clase DAO

Este método usa el query “INSERT INTO” para crear una tupla en la tabla “equipos” para esto recibe un objeto de la clase “Equipo” crea un Statement con la conexión devuelta por el metodo “getConnection()” de la clase “Conexion” y le setea en los “values” primero el atributo nombre usando el método “getNombre()” del objeto que recibe nuestro método y después el score con utilizando el método “getScore()”, después se ejecuta la actualización con el metodo “executeUpdate()” del Statement dentro de una condicional “if” y si regresa un valor mayor a cero significa que se ejecutó correctamente, sino significa que no se pudo realizar el update, dependiendo de esto el mensaje puede ser “Equipo agregado” o “Equipo no agregado” que es seteado a la variable “msj” y retornado por el método, todo va dentro de su respectivo “try/catch” y en el “finally” se cierra el Statement y la conexión a través del método “close()”.

Para eliminar los equipos de la base de datos se creó el método “eliminarEquipo()” el cual recibe un id para usarlo como referencia a la hora de realizar la actualización en la base de datos, a continuación en la Fig. 10 se muestra su funcionamiento.

Texto

Descripción generada automáticamente

Fig. 10 Método "eliminarEquipo" de la clase DAO.

En el String “sql” se concatena el query que es “DELETE FROM equipos WHERE id=” con el id que recibimos como parámetro para mandar la instrucción a la base de datos y efectuar el update, se crea el Statement utilizando el query definido en la variable “sql” y en la condicional “if” se ejecuta la actualización a través del método “executeUpdate()” del Statement, si se efectúa la actualización devolverá un valor mayor que cero sino significa que no se puedo realizar la actualización, entonces la variable “msj” obtendrá el valor de “Equipo eliminado” o “El equipo no se pudo eliminar” según sea el caso, dicho mensaje será retornado por el método, todo va dentro de su respectivo “try/catch” y su “finally” para cerrar el Statement y la conexión.

El siguiente método de nuestra clase DAO es “actualizarEquipo()” el cual tiene como finalidad actualizar los datos de una tupla de la tabla equipos, para esto recibe como parámetro un objeto de la clase “Equipo” como se observa en la Fig. 11.

Texto

Descripción generada automáticamente

Fig. 11 Método "actualizarEquipo" de la clase DAO.

Dentro de un “try/catch” se almacena el query dentro del String “sql”, el cual es un “UPDATE equipos” y se le concatena el “Nombre”, el “Score” y el “Id” del objeto recibido como parámetro a través de sus getters, se prepara un Statement con la conexión y el query, se llama al método “executeUpdate()” del Statement y devuelve un valor mayor a cero si se ejecutó correctamente o cero si no se pudo ejecutar, finalmente se cierra el Statement y la conexión, el método retorna el mensaje.

Los siguientes métodos corresponden a la tabla “partidos” de la base de datos, son métodos que manipulan la información de dicha tabla, comenzando con el método “GetAllPartidos()” que se muestra a continuación en la Fig. 12.

Texto

Descripción generada automáticamente

Fig. 12 Metodo "GetAllPartidos" de la clase DAO

Este método extrae todas las tuplas de la tabla “partidos” de nuestra base de datos, los ingresa en una ArrayList y lo retorna, para esto dentro de un “try/catch” se define el query en un String llamado “sql” en este caso es un “SELECT \* FROM partidos”, se crea un Statement con la conexión y el Statement, para luego ejecutar el query, en este caso se requiere de una variable de tipo ResultSet para guardar el contenido devuelto por la consulta, dicha variable es “rs”, con un while iteramos el ResultSet para acceder a las tuplas obtenidas y creamos un objeto de tipo “Partido” por cada una de estas tuplas, finalmente almacenamos el objeto en el Array, que es devuelto al final del método, finalmente cerramos el ResultSet, el Statement y la conexión con su respetivo método “close()”.

El siguiente método es “GetPartidos()” el cual realiza casi lo mismo que el método “GetAllPartidos()” explicado anteriormente con la diferencia que limita la consulta solamente a las primeras 6 tuplas, de la tabla, ordenadas de manera descendente por su ID en la Fig. 13 podemos observar el método.

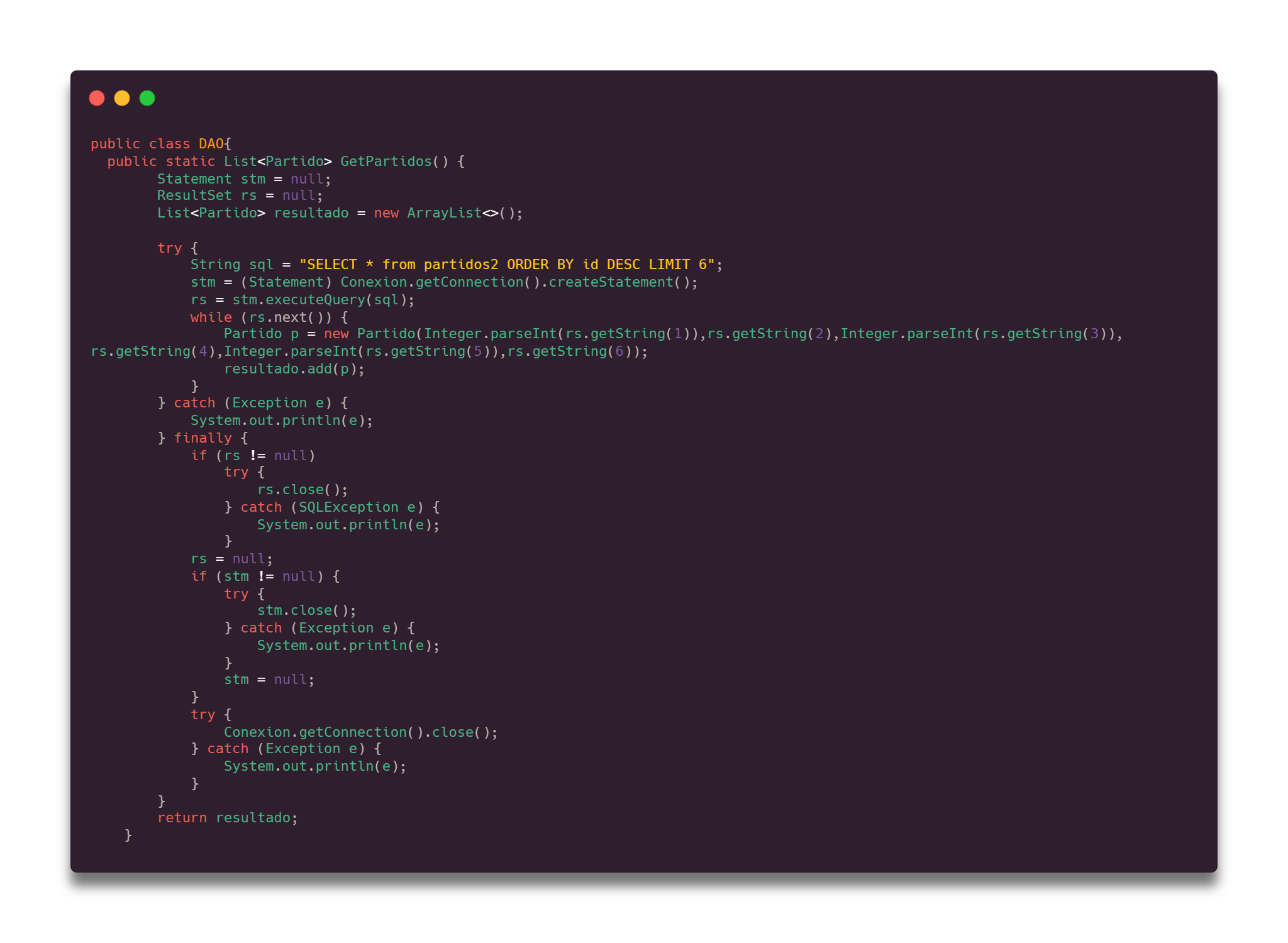


Fig. 13 Método "GetPartidos" de la clase DAO.

Dentro de un “try/catch” almacenamos el query en un String llamado “sql” con la consulta que deseamos hacer, en este caso “SELECT \* FROM partidos ORDER BY id DESC LIMIT 6”, creamos el Statement y un ResultSet donde se almacenara lo que devuelva el “executeQuery()” del Statement, iteramos en el “rs” para crear objetos de tipo “Partido” y los almacenamos en el ArrayList que retorna el método, una vez hecho esto procedemos a cerrar el ResultSet después el Statement y finalmente la conexión.

El método que sigue de nuestra clase DAO es “agregarPartido()” que sirve para agregar tuplas a nuestra tabla “partidos” de la base de datos, para esto recibe como parámetro un objeto de la clase “Partido” del cual extraerá la información requerida para agregarlo a la tabla, para esto hacemos el query “INSERT INTO partidos () VALUES ()” que esta guardado en una variable de tipo String llamada “sql”, preparamos el Statement con la conexión y el query previamente creado, a este Statement le setteamos los valores deseados tomando los atributos del objeto “Partido” que recibimos, con sus getters y sus setters, y ejecutamos el query con el método “executeUpdate()” del Statement, como se muestra en la Fig. 14.

Texto

Descripción generada automáticamente

Fig. 14 Método "agregarPartido" de la clase DAO

En una condicional “if” comprobaremos lo que nos devolvió el “executeUpdate()” si es mayor que cero significa que se ejecutó correctamente, si no, significa que no se pudo realizar la actualización, esto asignara un valor a nuestra variable “msj” que será retornada por nuestro método, para terminar la actualización cerramos el Statement previamente creado, así como la conexión con su respectivo método “close()”.

El siguiente método que creamos es “eliminarPartido()” el cual recibe un “id” del partido que queremos eliminar para tomar como referencia y poder eliminar la tupla deseada de la tabla de “partidos”, concatenamos dicho “id” con nuestro query “DELETE FROM partidos” y se lo mandamos al método “prepareStatement()” de la conexión y lo almacenamos en una variable Statement, ejecutamos la actualización con el método “executeUpdate()” del Statement, finalmente cerramos el Statement y la conexión, como se muestra en la Fig. 15.

Texto

Descripción generada automáticamente

Fig. 15 Método "eliminarPartido" de la clase DAO.

El ultimo método de la clase DAO es “actualizarPartido()” el cual sirve para actualizar una tupla de nuestra tabla de “partidos” recibiendo como parámetro un objeto de la clase “Partido” del cual obtiene la información requerida para ejecutar la actualización, con esta información construimos un String con el query deseado, en este caso un “UPDATE partidos” como se muestra en la Fig. 16.

Texto

Descripción generada automáticamente

Fig. 16 Método "actualizarPartido" de la clase DAO.

Concatenamos la información nueva al query, usando los getters del objeto “Partido” que recibimos, y creamos un Statement con el método “prepareStatement” de la conexión, después ejecutamos la actualización con el método “executeUpdate()” del Statement, y nos devolverá un valor mayor a uno en caso de que se haya efectuado correctamente, finalmente cerramos el Statement y la conexión con el método “close()” y retornamos el mensaje guardado en la variable “msj”.

Esos son todos los métodos de nuestra clase Data Access Object con esto podemos manipular los datos de nuestra base de datos, listo para hacer consultar y crear un CRUD sobre los equipos y los partidos de Voleibol de nuestro sitio web.

# App y Main

La clase App es nuestra clase principal donde se encuentra el Main, que es la encargada de responder a las solicitudes del Frontend, nuestro Backend le devuelve la información en Json porque es una forma fácil de enviar los datos y más ligera, primero creamos el CORS para que nuestro servidor mande el “Access-Control-Allow-Origin” y no nos bote un error a la hora de querer consultarlo, como se muestra en la Fig. 17.

Texto

Descripción generada automáticamente

Fig. 17 Código CORS.

Posteriormente creamos el primer “get” el cual retorna el contenido que nos proporciona el método “GetEquipos()” de la clase “DAO” en forma de Json cuando solicitamos un “get” a la dirección raíz, esto se muestra en la Fig. 18.

Texto

Descripción generada automáticamente

Fig. 18 Get a la raíz.

El segundo “get” que creamos es a la dirección “/allEquipos” el cual manda en formato Json el Array que nos proporciona el método “GetAllEquipos()” de la clase “DAO” como vemos en la Fig. 19.

Texto

Descripción generada automáticamente

Fig. 19 Get "/allEquipos".

Creamos un “post” con la dirección “/addEquipo” el cual guarda los datos del “request” en una variable, convierte el objeto Json en un objeto de la clase “Equipo” y llama el método “agregarEquipo()” de la clase “DAO” le manda el objeto creado previamente y devuelve un objeto Json con la propiedad “status” que contiene el mensaje retornado por el método “agregarEquipo()” como se ve en el código de la Fig. 20.



Fig. 20 Post "/addEquipo".

Creamos otro “Post” a la dirección “/delEquipo” el cual se encarga de llamar el método “eliminarEquipo()” de la clase “DAO”, para esto toma los datos del “request” y los convierte en un objeto de la clase “Equipo” utiliza su método “getId()” para mandárselo al método “eliminarEquipo()” y le retorna al cliente un objeto Json con la propiedad “status” con el mensaje devuelto por el método “eliminarEquipo()” como muestra la Fig. 21.

Texto

Descripción generada automáticamente

Fig. 21 Post "/delEquipo".

El siguiente “Post” creado es a la dirección “/updateEquipo” este toma del “request” los datos y los convierte en un objeto de la clase “Equipo”, dicho objeto lo manda al método “actualizarEquipo()” de la clase DAO, y devuelve al cliente un objeto Json con la propiedad “status” que contiene el mensaje si pudo proceder la actualización, el código se muestra en la Fig. 22.



Fig. 22 Post "/updateEquipo".

Para las consultas y actualizaciones de la tabla “partidos” de nuestra base de datos creamos los siguientes “gets” y “post”, el primero de ellos es un “get” con la dirección “/partidos” el cual devuelve el Array que nos proporciona el método “GetPartidos()” de la clase DAO y lo manda al cliente en forma de Json, código en la Fig. 23.



Fig. 23 Get "/partidos".

El siguiente “get” hace lo mismo que el anterior con la diferencia de que el Array que devuelve contienen todas las tuplas de la tabla “partidos” de la base de datos, el código se muestra en la Fig. 24.

Texto

Descripción generada automáticamente

Fig. 24 Get "/allPartidos".

Se creo un “post” con la dirección “/addPartido” el cual toma los datos del “request” para convertirlos en un objeto de tipo “Partido”, dicho objeto lo manda al método “agregarPartido()” de la clase “DAO” lo que retorne dicho método es mandado como respuesta al cliente en un objeto Json, se muestra en la Fig. 25.

Texto

Descripción generada automáticamente

Fig. 25 post "/addPartido".

El siguiente “post” tiene la dirección “/delPartido” este toma el body del “request” y lo guarda en un String, después lo convierte de Json a un objeto de nuestra clase “Equipo” y usa su método “getId()” para mandárselo como parámetro al método “eliminarPartido()” de nuestra clase “DAO” finalmente retorna el valor que nos devolvió este último método, como se muestra en la Fig. 26.

Texto

Descripción generada automáticamente

Fig. 26 post "/delPartido".

Finalmente creamos un “post” con la dirección “/updatePartido” el cual se encarga de actualizar los datos de nuestras tuplas de la tabla “partidos” de la base de datos, para esto toma los datos del “request” y convierte el objeto Json en un objeto de nuestra clase “Partido” dicho objeto lo manda al método “actualizarPartido()” de nuestra clase “DAO” el mensaje que nos retorne este método es agregado como propiedad de un objeto Json que es devuelto al cliente como se puede ver en la Fig. 27.

Texto

Descripción generada automáticamente

Fig. 27 post "/updatePartido".

Este es todo el contenido de nuestra clase App, y es todo por parte del Backend, se realizaron prueba para comprobar el correcto funcionamiento y está listo para ser conectado con el Frontend.