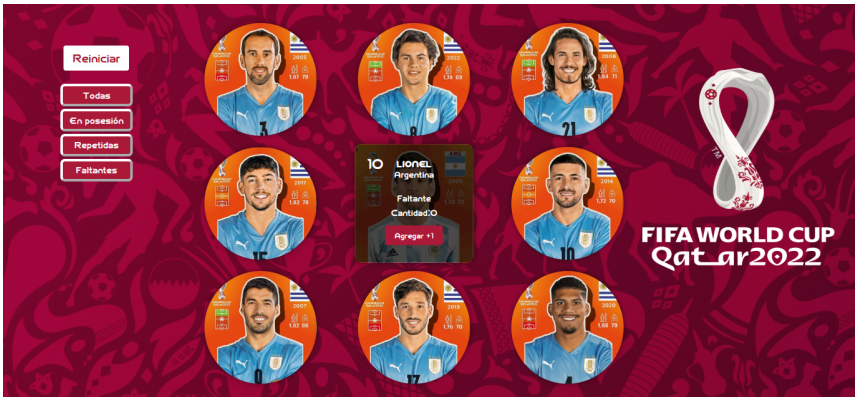
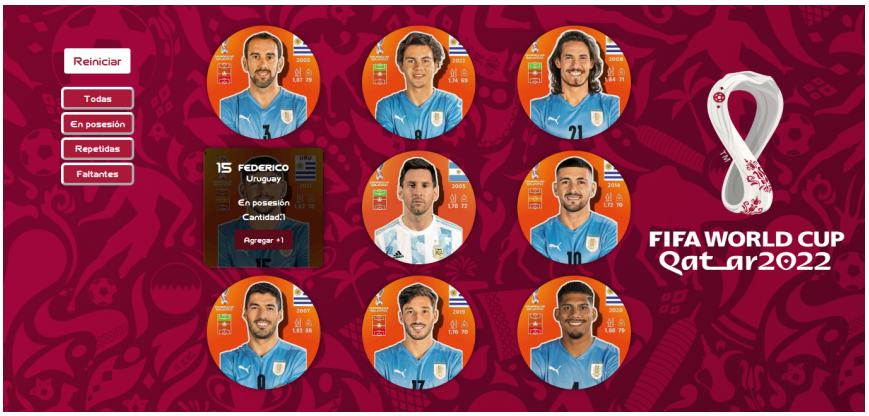
Tracking de Figuritas Mundial Qatar 2022 API

Yessica Sosa


Reiniciar
Todas
En posesión
Repetidas
Faltantes
10 LIONEL
Argentina
Faltante
Cantidad:0
Agregar +1
FIFA WORLD CUP
Qatar2022


Reiniciar
Todas
En posesión
Repetidas
Faltantes
15 FEDERICO
Uruguay
En posesión
Cantidad:1
Agregar +1
FIFA WORLD CUP
Qatar2022


Reiniciar
Todas
En posesión
Repetidas
Faltantes
FIFA WORLD CUP
Qatar2022

ÍNDICE

1. Introducción.

La API de seguimiento de figuritas es una gran herramienta que permite a los usuarios rastrear y administrar su colección de figuritas del Mundial.
Con esta, los usuarios pueden ver todas las figuritas que poseen, las que no tienen y las que tienen repetidas, lo que les brinda una vista completa de su colección en todo momento.

Cuenta con una interfaz fácil de usar que permite a los usuarios realizar consultas a través de diferentes endpoints. Estos endpoints incluyen funciones para agregar figuritas de una colección, resetear la cantidad que se posee y obtener estadísticas sobre la colección de un usuario, como la cantidad que se posee de cada una.

Los desarrolladores pueden personalizar la API para que se ajuste a las necesidades específicas de su plataforma o aplicación, lo que significa que se pueden agregar o eliminar endpoints y funciones según sea necesario.

En resumen, la API de seguimiento de figuritas del Mundial es una herramienta poderosa para los fanáticos del fútbol y los coleccionistas de figuritas. Con su interfaz fácil de usar, es una opción ideal para cualquier persona que quiera tener un seguimiento completo de su colección de figuritas del Mundial.

2. Funcionamiento back-end.

La API utiliza una base de datos en PostgreSQL para almacenar información sobre las figuritas del Mundial, incluyendo detalles sobre las figuritas, jugadores y países representados, así como la cantidad que se posee de cada una, su número de jugador, ID identificador, un valor booleano sobre el estado de posesión y una imagen para cada jugador.

2. 1. Aplicación.

Este programa utiliza el framework Express.js para crear una API RESTful que se encarga de manejar datos de las figuritas del mundial.
Express.js es el framework backend más popular para Node.js, y es una parte extensa del ecosistema JavaScript.

Está diseñado para construir aplicaciones web de una sola página, multipágina e híbridas, también se ha convertido en el estándar para desarrollar aplicaciones backend con Node.js

En el archivo app.js se importan las dependencias necesarias, incluyendo el modelo Figurita que define la estructura de la tabla de figuritas en la base de datos.

Luego se crea una instancia de la aplicación Express, se configura el motor de plantillas EJS para renderizar vistas y se establece la ruta para servir archivos estáticos.

A continuación, se definen las rutas que se encargan de interactuar con la base de datos. La ruta '/' responde a una petición GET y devuelve una vista que muestra todas las figuritas y su cantidad, ordenadas por ID.
La ruta '/figuritas/:id/cantidad' responde a una petición PUT y actualiza la cantidad de una figurita en la base de datos. Si la cantidad es mayor o igual a 1, se marca que el usuario tiene la figurita.
La ruta '/reset' responde a una petición POST y restablece la cantidad y el estado de todas las figuritas en la base de datos.

En caso de ocurrir algún error en el servidor, se envía una respuesta de error al cliente con un mensaje descriptivo. Finalmente, se exporta la aplicación Express para su uso en otros módulos.

2. 2. Rutas.

Cada ruta está asociada a una función de controlador que se encarga de manejar las solicitudes de esa ruta específica.

GET /figuritas: Esta ruta se utiliza para obtener todas las figuritas almacenadas en la base de datos. Esta ruta está asociada a la función *getFiguritas* en el controlador de figuritas.

GET /figuritas/:id: Esta ruta se utiliza para obtener una figurita específica por su ID. Esta ruta está asociada a la función *getFigurita* en el controlador de figuritas.

POST /figuritas: Esta ruta se utiliza para crear una nueva figurita en la base de datos. Esta ruta está asociada a la función *createFiguritas* en el controlador de figuritas.

PUT /figuritas/:id/cantidad: Esta ruta se utiliza para actualizar la cantidad de una figurita por su ID. Esta ruta está asociada a la función *updateFiguritaCantidad* en el controlador de figuritas.

DELETE /figuritas/:id: Esta ruta se utiliza para eliminar una figurita por su ID. Esta ruta está asociada a la función *deleteFiguritas* en el controlador de figuritas.

PUT /figuritas: Esta ruta se utiliza para restablecer todas las figuritas en la base de datos. Esta ruta está asociada a la función *resetearFiguritas* en el controlador de figuritas.

Cada ruta se define utilizando el objeto router de Express y la función de ruta correspondiente (*get, post, put,* o *delete*). Las rutas que incluyen un parámetro dinámico (como /:id) permiten que el controlador acceda a ese parámetro utilizando r*eq.params.id*.

Una vez que se han definido todas las rutas, el router se exporta para que pueda ser utilizado en otros módulos.

2. 3. Controladores.

Los controladores son una parte esencial de la arquitectura del patrón Modelo-Vista-Controlador en aplicaciones web. Su función principal es manejar las solicitudes del usuario y proporcionar una respuesta adecuada.

Cada controlador sigue una estructura similar, comenzando con un bloque try-catch para manejar errores y una serie de operaciones para interactuar con la base de datos a través del modelo Figurita. Dependiendo de la solicitud, los controladores pueden leer, actualizar, crear o eliminar registros en la base de datos, y proporcionan una respuesta adecuada al usuario en formato JSON.

En este caso trabajamos con los siguientes controladores:

updateFiguritaCantidad: Este controlador se utiliza para incrementar la cantidad de una figurita. Primero encuentra la figurita según su ID y luego incrementa el valor en la tabla "cantidad" y lo guarda.

getFiguritas: Este controlador se utiliza para obtener todas las figuritas. Encuentra todas las figuritas y las ordena de forma ascendente.

getFigurita: Este controlador se utiliza para obtener una figurita según su ID.

createFiguritas: Este controlador se utiliza para crear una nueva figurita (en caso de posible futuro uso). Define los parámetros necesarios y crea una nueva figurita.

deleteFiguritas: Este controlador se utiliza para eliminar una figurita según su ID (en caso de posible futuro uso). Encuentra la figurita según su ID y la elimina.

resetearFiguritas: Este controlador se utiliza para resetear el valor de "cantidad" a 0 y el valor "tengo" a false. Actualiza los valores de "cantidad" y "tengo" donde {} (todos).

2. 4. Modelo.

Se define el modelo de datos para la tabla de *figuritas* utilizando Sequelize. Sequelize es una biblioteca ORM (Object-Relational Mapping) para Node.js que permite interactuar con bases de datos relacionales.

En el modelo, se especifica la estructura de la tabla de la base de datos, es decir, los campos que la tabla tendrá y los tipos de datos que tendrán esos campos.

En este caso, la tabla *figuritas* tiene seis campos: *id, nombre, país, número, tengo, cantidad* e *imagen*.

Cada campo tiene un tipo de dato asociado, como INTEGER para *id* y *número*, STRING para *nombre, país* e *imagen*, y BOOLEAN para *tengo*. También se establecen otras propiedades para los campos, como si serán valores por defecto, si serán claves primarias, si se auto-incrementarán y si se permiten valores nulos.

Se especifica que no se crearán las columnas *created_At* y *updated_At* por defecto en la tabla mediante la opción *timestamps: false*, lo que significa que no se registrarán automáticamente las fechas de creación y actualización de cada fila.

2. 5. Base de datos.

PostgreSQL es un sistema de gestión de bases de datos relacionales de código abierto que se utiliza ampliamente en aplicaciones empresariales y web. PgAdmin4 es una plataforma de administración de bases de datos de código abierto que proporciona una interfaz gráfica de usuario para administrar bases de datos.

En este caso, la base de datos *"figuritas"* se ha creado utilizando PgAdmin4, con la ayuda de una instancia de PostgreSQL. Esta base de datos contiene una tabla llamada *"figuritas"*, que almacena información relacionada a las figuritas del mundial.

La tabla contiene seis columnas: *"id", "nombre", "pais", "numero", "tengo", "cantidad"* e *"imagen"*. La columna *"id"* es la clave principal de la tabla y se utiliza para identificar de forma única cada registro en la tabla.
Las columnas "nombre", "pais", "numero", "tengo", "cantidad" e "imagen" almacenan información relacionada, como su nombre, país, número, si se tiene o no y la cantidad que se tiene de cada una, junto con una imágen de cada jugador.

La base de datos proporciona una manera eficiente y escalable de almacenar y administrar información relacionada con las figuritas del mundial.

Se crea una instancia de la clase Sequelize para establecer una conexión con una base de datos PostgreSQL. La instancia se llama *sequelize* y toma tres argumentos: el nombre de la base de datos, el nombre de usuario y la contraseña. Luego, se especifican algunas opciones, como la dirección del host y el dialecto de la base de datos (en este caso, 'postgres').

2. 6. Inicialización.

Se da inicio a la aplicación mediante un archivo index.js.
Este código es la función principal de la aplicación. Aquí se establece la conexión con la base de datos a través de Sequelize, se sincronizan los modelos con la base de datos y se inicia el servidor.

Primero, la función *main* utiliza *sequelize.sync()* para sincronizar los modelos definidos en la base de datos. La opción *{force: false}* indica que no se debe forzar la eliminación y re-creación de las tablas de la base de datos.

Luego, se agregan las rutas definidas en *figuritasRoutes* a la aplicación express app.

Finalmente, se inicia el servidor con *app.listen(PORT)*. La aplicación se escuchará en el puerto especificado en *PORT*, o en el puerto 3000 por defecto si no se especifica ningún puerto.
Si todo funciona correctamente, se imprimirá un mensaje en la consola indicando que el servidor está en ejecución en el puerto especificado. En caso de errores, se captura la excepción y se muestra un mensaje de error en la consola.

3. Funcionamiento front-end.

Del lado del cliente se ha creado una composición de archivos front-end que generan una interfaz amigable y fácil de entender.
Esto se ha hecho mediante EJS, JavaScript y CSS.

EJS (Embedded JavaScript) es un motor de plantillas para JavaScript que permite generar dinámicamente HTML y otros tipos de archivos que se envían al navegador.

3. 1. Diseño y generación dinámica.

Mediante EJS y CSS se ha creado un diseño llamativo para el usuario.
En el archivo EJS, la sección <*head*> incluye algunos metadatos importantes, como el título de la página, el icono que se mostrará en la pestaña del navegador, la escala de la página y la codificación de caracteres. También se incluye un archivo de estilo CSS externo.

En el <*body*> se encuentra el contenido principal de la página. Se usa un bucle for para recorrer un array de objetos *figuritas* y se crea una sección de información para cada objeto. En cada sección se muestra la imagen, nombre, país, número y cantidad de cada figurita, y se incluye un botón para agregar +1 a la cantidad. Además, se muestra si la *figurita* está en posesión o es faltante.

También se agrega un botón para reiniciar la cantidad de todas las *figuritas* y un conjunto de botones para filtrarlas por diferentes categorías.

Por último, se incluye un archivo JavaScript externo llamado *principal.js*, que contiene el código para manipular los elementos de la página y realizar las acciones de agregar y filtrar.

3. 2. Comportamiento dinámico.

Mediante JavaScript se han creado las funcionalidades dentro de la interfaz de usuario de esta API.

El código JavaScript es una parte importante, tiene varias funciones, como incrementar la cantidad de una figurita cuando se hace clic en el botón *"+1"*, reiniciar el conteo de las figuritas, y filtrarlas según ciertos criterios.

La primera sección del código obtiene todos los botones de *"Agregar +1"* utilizando el método *document.querySelectorAll()*. Luego, se itera sobre cada botón y se agrega un event listener al evento *"click"* utilizando el método *button.addEventListener()*.
Cuando se hace clic en un botón, el código obtiene el ID de la *figurita* asociada a este y luego hace una solicitud PUT al servidor para incrementar la cantidad de la *figurita* en la base de datos. Si la solicitud es exitosa, el código hace una solicitud GET a la API para obtener la cantidad actualizada de la figurita y actualiza la cantidad en la página.

La siguiente sección del código define un botón de reinicio y agrega un *event listener* al evento "*click*". Cuando se hace *clic* en este botón, el código hace una solicitud POST al servidor para reiniciar la cantidad de todas las *figuritas* a cero. Si la solicitud es exitosa, la página se recarga para mostrar los nuevos valores.

La última sección del código define cuatro botones de filtro y agrega *event listeners* al evento "*click*". Cada botón tiene una función que muestra un conjunto específico de figuritas.
El botón "*Todas*" muestra todas las figuritas, el botón "*Posesión*" muestra solo las figuritas que el usuario tiene en su colección, el botón "*Faltantes*" muestra solo las figuritas que el usuario no tiene en su colección, y el botón "*Repetidas*" muestra solo las figuritas que el usuario tiene más de una vez en su colección. Para mostrar u ocultar las figuritas según los criterios, el código utiliza el método *style.display* para cambiar la propiedad CSS "*display*" de los elementos HTML correspondientes.

El código incluye funciones para incrementar la cantidad de *figuritas*, reiniciar el conteo de las *figuritas*, y *filtrarlas*.

4. Usos futuros y posibles mejoras.

Esta API actualmente ofrece una funcionalidad básica para la gestión de una colección de figuritas, permitiendo al usuario agregar, eliminar y actualizar la cantidad de cada una de ellas. Sin embargo, existen una serie de mejoras y funcionalidades adicionales que podrían ser implementadas en el futuro para mejorar la experiencia del usuario.

Una posible mejora en el código podría ser la optimización del rendimiento de la API, especialmente en la gestión de grandes colecciones de figuritas. Esto podría lograrse mediante la implementación de técnicas de caché, como el almacenamiento en caché de respuestas a solicitudes frecuentes para reducir la carga en el servidor.

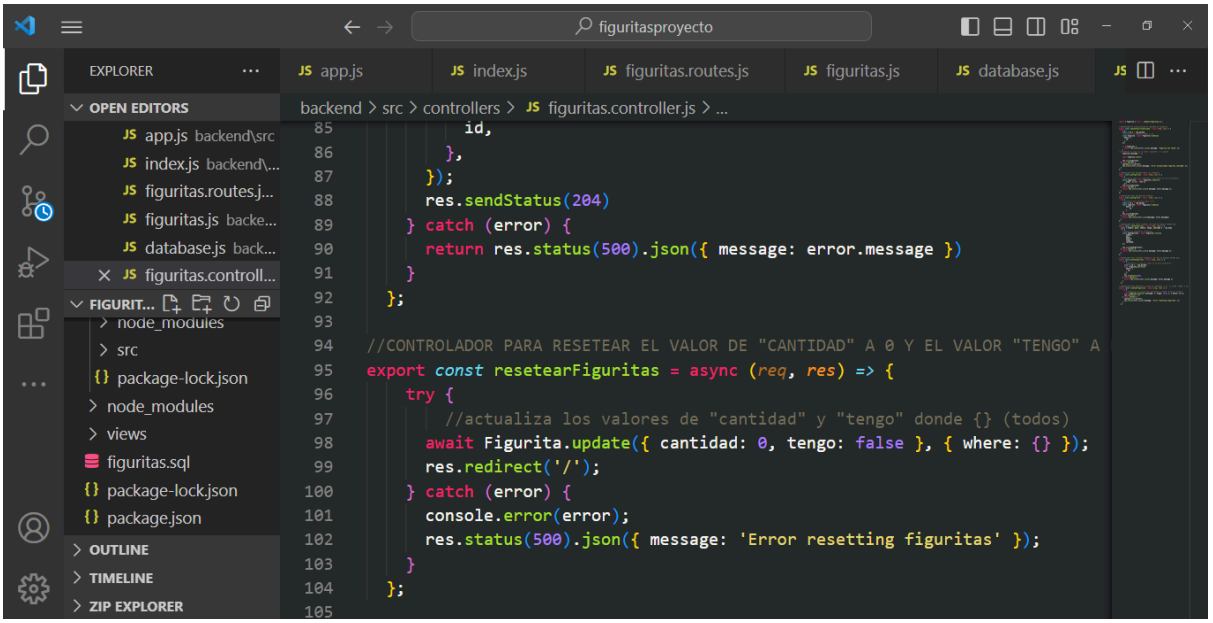
La autenticación podría ser una funcionalidad muy útil para evitar el acceso no autorizado a la API y para proporcionar una experiencia personalizada para cada usuario. Una posible implementación sería la utilización de tokens de autenticación generados y proporcionados al cliente en el momento del inicio de sesión.

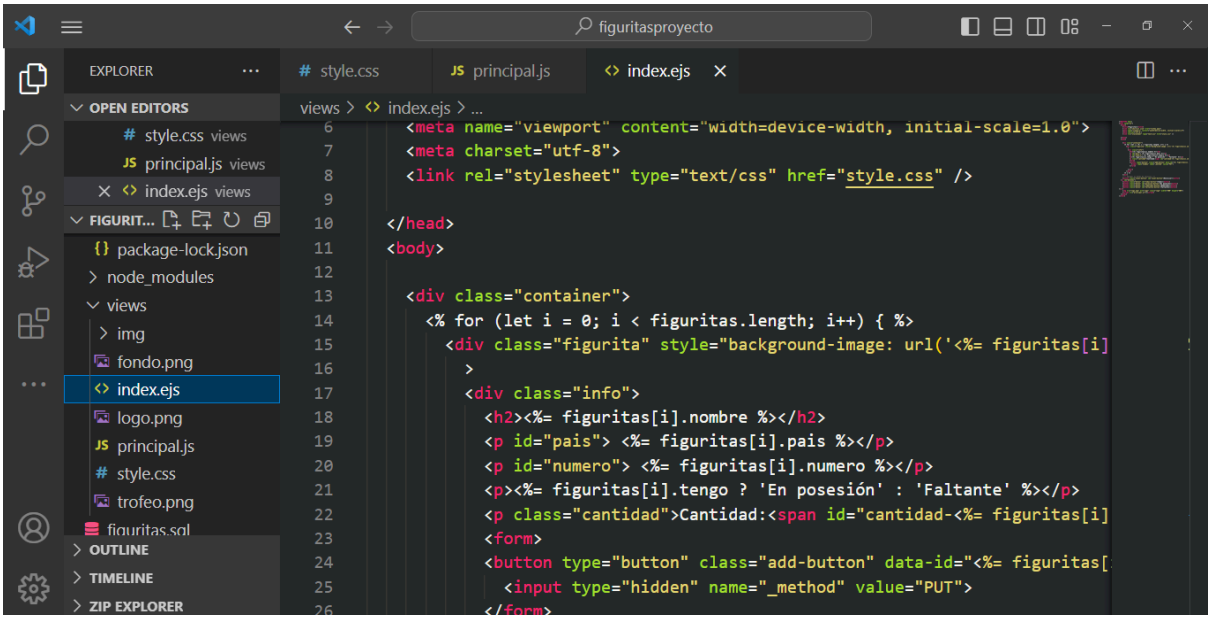
Otra funcionalidad que podría ser interesante para los usuarios sería la posibilidad de intercambiar figuritas con otros. Esto podría implementarse mediante la creación de una función que permita la búsqueda de usuarios con figuritas específicas y la organización de intercambio entre ellos.

Además, la posibilidad de crear y personalizar figuritas podría ser una funcionalidad muy atractiva para los usuarios de la API. Esto les permitiría crear y agregar sus propias figuritas a sus colecciones, lo que aumentaría el valor y la personalización de la experiencia.

Otras posibles funcionalidades incluyen la posibilidad de organizar eventos y concursos relacionados a las colecciones, la implementación de funciones de gamificación para motivar a los usuarios a coleccionar más figuritas y la integración con otras aplicaciones y servicios de terceros relacionados con los hobbies y la cultura.

Esta API de gestión de colecciones de figuritas tiene un gran potencial para seguir creciendo y ofreciendo nuevas funcionalidades y mejoras para incrementar la experiencia del usuario.


figuritasproyecto
EXPLORER
OPEN EDITORS
app.js backend\src
index.js backend\...
figuritas.routes.j...
figuritas.js backe...
database.js back...
figuritas.controll...
FIGURIT...
node_modules
src
package-lock.json
node_modules
views
figuritas.sql
package-lock.json
package.json
OUTLINE
TIMELINE
ZIP EXPLORER
app.js
index.js
figuritas.routes.js
figuritas.js
database.js
backend > src > controllers > figuritas.controller.js > ...
id,
},
});
res.sendStatus(204)
} catch (error) {
return res.status(500).json({ message: error.message })
}
};
//CONTROLADOR PARA RESETEAR EL VALOR DE "CANTIDAD" A 0 Y EL VALOR "TENGO" A
export const resetearFiguritas = async (req, res) => {
try {
//actualiza los valores de "cantidad" y "tengo" donde {} (todos)
await Figurita.update({ cantidad: 0, tengo: false }, { where: {} });
res.redirect('/');
} catch (error) {
console.error(error);
res.status(500).json({ message: 'Error resetting figuritas' });
}
};


figuritasproyecto
EXPLORER
OPEN EDITORS
style.css views
principal.js views
index.ejs views
FIGURIT...
package-lock.json
node_modules
views
img
fondo.png
index.ejs
logo.png
principal.js
style.css
trofeo.png
figuritas.sql
OUTLINE
TIMELINE
ZIP EXPLORER
style.css
principal.js
index.ejs
views > index.ejs > ...
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>

<% for (let i = 0; i < figuritas.length; i++) { %>
<div class="figurita" style="background-image: url('<%= figuritas[i]
>

<h2><%= figuritas[i].nombre %></h2>
 <%= figuritas[i].pais %>
 <%= figuritas[i].numero %>
<%= figuritas[i].tengo ? 'En posesión' : 'Faltante' %>
Cantidad:<span id="cantidad-<%= figuritas[i]
<form>
<button type="button" class="add-button" data-id="<%= figuritas[
<input type="hidden" name="_method" value="PUT">
</form>