

Guía para el backend del proyecto y uso de base de datos

Esta es una guía para la implementación del backend (API) del proyecto y el uso de la base de datos. Esta guía es obligatorio realizarla para poder proseguir con el proyecto.

¿Para qué es la aplicación que estamos creando?

Estás creando una aplicación web con React para gestionar una colección de objetos, ya sea de música, de ropa, de videojuegos, juegos de mesa, cartas de pokémon, etc. Tú eliges de qué vas a hacer la colección.

A través del login cuya interfaz ya tienes creada, van a poder acceder dos tipos de usuario:

- . Usuario con el rol de admin: este usuario podrá acceder a la colección y realizar las operaciones de insertar, borrar y actualizar en la base de datos a través de la interfaz que crearás en la página home.
- . Usuario con el rol user: este usuario podrá acceder a la colección y realizar sólo la operación de insertar en la base de datos a través de la interfaz.

Base de datos del proyecto

En el campus, en la parte de *Base de datos y Ficheros para el backend* (en la parte de Recursos de UT2) tienes el fichero bdgestion.sql que es una base de datos MySQL con dos tablas:

- . Tabla **colección**: tabla con un id (clave principal), nombre, marca, tipo y precio. La tabla está vacía.



The screenshot shows the MySQL Workbench interface with the following details:

- Servidor: 127.0.0.1
- Base de datos: bdgestion
- Tabla: colección
- Toolbar buttons: Examinar, Estructura, SQL, Buscar, Insertar, Exportar, Importar, Print.
- Selected tab: Estructura de tabla
- Table structure:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	id	int(11)			No	Ninguna		AUTO_INCREMENT
2	nombre	varchar(100)	utf8_spanish_ci		Sí	NULL		
3	marca	varchar(100)	utf8_spanish_ci		Sí	NULL		
4	tipo	varchar(100)	utf8_spanish_ci		Sí	NULL		
5	precio	double			Sí	NULL		

- . Tabla **usuarios**: tabla con los usuarios que pueden acceder a la aplicación web

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	id 	int(11)			No	Ninguna		AUTO_INCREMENT
2	nombre	varchar(255)	utf8_spanish_ci		No	Ninguna		
3	login	varchar(50)	utf8_spanish_ci		No	Ninguna		
4	password	varchar(50)	utf8_spanish_ci		No	Ninguna		
5	rol	varchar(25)	utf8_spanish_ci		No	Ninguna		

La tabla de usuarios ya tiene dos usuarios predefinidos: Patricia con el rol de admin y usuario con el rol de user.

	id	nombre	login	password	rol
-	1	Patricia	patricia	123456789	admin
-	2	usuario	user	123456789	user

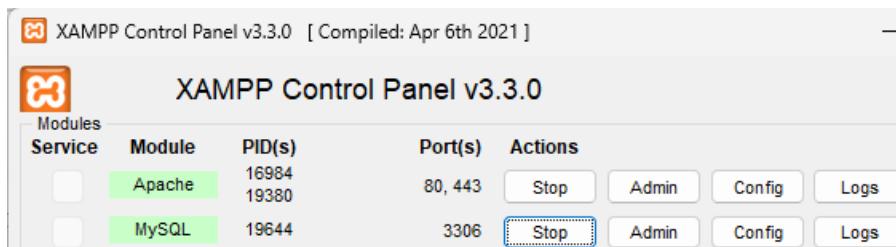
Para poder usar la base de datos tenemos que instalar XAMPP.

Instalar el XAMPP para Windows

<https://www.apachefriends.org/es/index.html>



El XAMPP tiene un servidor Apache y un módulo MySQL. Esos dos son los que vamos a usar. Al picar en Start iniciamos ambos servicios:

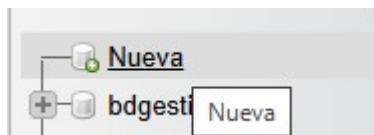


Al picar en Admin en MySQL se nos abre el administrador de la base de datos <http://localhost/phpmyadmin/>

The screenshot shows the phpMyAdmin interface. The top navigation bar includes tabs for 'Servidor: 127.0.0.1', 'Bases de datos', 'SQL', 'Estado actual', 'Cuentas de usuarios', and 'Exportar'. The main area is titled 'Bases de datos'. On the left, a sidebar lists existing databases: 'information_schema', 'mysql', 'performance_schema', 'phpmyadmin', and 'test'. A 'Nueva' (New) item is also listed under the sidebar. On the right, a form for creating a new database is displayed. It has a 'Crear base de datos' (Create database) button, a text input field containing 'bdgestion', and a dropdown menu set to 'utf8mb4_general_ci'. A 'Crear' (Create) button is at the bottom of the form.

Crear una nueva base de datos llamada bdgestion e importar el fichero bdgestion.sql

Para importar la base de datos bdgestion.sql primero debemos crear una nueva base de datos, así que picamos en Nueva en el panel de la izquierda:



En el panel de la derecha que nos sale escribimos el nombre de la base de datos: **bdgestion** y luego picamos en **Crear**.

Bases de datos

The screenshot shows the 'Crear base de datos' (Create database) form. It has a 'Nombre' (Name) input field containing 'bdgestion', a 'Collation' dropdown set to 'utf8mb4_general_ci', and a 'Crear' (Create) button. The 'Crear' button is highlighted with a red box.

Una vez creada, de nuevo en el panel izquierdo picamos sobre la base de datos (1) y luego en el panel superior picamos en Importar (2), luego seleccionamos el archivo de bdgestion.sql (3)

Importando en la base de datos "fefa"

Archivo a importar:

El archivo puede ser comprimido (gzip, bzip2) o descomprimido.
Un archivo comprimido tiene que terminar en **.[formato].[compresión]**. Por ejemplo: .sql.gz

Buscar en su ordenador: (Máximo: 40MB)

Seleccionar archivo **bdgestion.sql**

Por último, nos vamos a la parte de abajo de esa misma página y picamos en Importar (4):

Opciones específicas al formato:

Modalidad SQL compatible:

NONE

No utilizar AUTO_INCREMENT con el valor 0

Importar

Finalmente nos saldrá algo parecido a lo siguiente:

Importación ejecutada exitosamente, 19 consultas ejecutadas. (bdgestion.sql)

MySQL ha devuelto un conjunto de valores vacío (es decir cero columnas). (La consulta tardó 0.0000 segundos)

```
-- phpMyAdmin SQL Dump -- version 4.8.5 -- https://www.phpmyadmin.net/ -- -- Servidor: 127.0.0.1 -- Tiempo de generación: 15-10-2023 a las 11:10:32 -- Versión del servidor: 10.1.38-MariaDB -- Versión PHP: 7.2.34
```

SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";

[Editar en línea] [Editar] [Crear código PHP]

Y ya podremos ver las tablas que tenemos en nuestra base de datos bdgestion:

Que contengan la palabra:

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
colección	Examinar Estructura Buscar Insertar Vaciar Eliminar	3	InnoDB	utf8_spanish_ci	16.0 KB	-
usuarios	Examinar Estructura Buscar Insertar Vaciar Eliminar	2	InnoDB	utf8_spanish_ci	16.0 KB	-
		5	InnoDB	utf8mb4_general_ci	32.0 KB	0 B

2 tablas Número de filas

Creación del backend en el proyecto: *endpoints*

Nuestro backend será una API a través de la cual el frontend obtendrá, eliminará e insertará datos en la base de datos.

En la carpeta de ProyectoMisiniciales crea la carpeta del backend:

```
mkdir backend  
cd backend
```

Dentro del backend usa el comando npm init para crear el fichero package.json puesto que lo necesita el node.js

```
npm init
```

Te pide los datos del fichero package.json, puedes dar a enter a todo o poner algunos como el author y la description.

```
PS C:\Users\Patricia\Documents\PATRICIA\SISTEMAS 24-25\AD 24-25\ProyectosReact\proyectopatricia\backend> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Press ^C at any time to quit.
package name (backend)
version (1.0.0)
description Backend del proyecto
entry point: (index.js)
test command:
git repository:
Keywords:
author: Patricia
license: (ISC)
About to write to C:\Users\Patricia\Documents\PATRICIA\SISTEMAS 24-25\AD 24-25\ProyectosReact\proyectopatricia\back
{
  "name": "backend",
  "version": "1.0.0",
  "description": "Backend del proyecto",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Patricia",
  "license": "ISC"
}

Is this OK? (yes) yes
PS C:\Users\Patricia\Documents\PATRICIA\SISTEMAS 24-25\AD 24-25\ProyectosReact\proyectopatricia\backend>
```

Instalación de las librerías necesarias para programar los *endpoints* del backend (API)

El frontend de nuestra aplicación se va a comunicar con la base de datos a través de ***endpoints***. Un ***endpoint*** (punto final de aplicación) es una URL (una dirección) de la API que se encarga de contestar una petición del frontend (cliente) enviada a través de métodos de solicitud HTTP. La contestación de los ***endpoint*** se trata en formato JSON.

Somos nosotros los programadores los que definimos los ***endpoints***. En principio, en nuestra API crearemos un ***endpoint*** llamado **/login** para obtener datos referentes al usuario en la tabla **usuarios** de nuestra base de datos. Más adelante, necesitaremos otros ***endpoints*** para seleccionar, insertar, actualizar y borrar datos en la tabla **colección**. En los ***endpoints*** es donde haremos las consultas SQL. Es por eso que estas funciones serán asíncronas.

Para trabajar con nuestro backend necesitamos instalar las librerías **express** y **cors**. Express es un framework web para node y cors nos permite que la aplicación tenga acceso a la API que estamos creando (desde otro servidor o, como es nuestro caso, desde localhost). Cors nos permite tener tanto la API como la aplicación en localhost.

Así pues instalamos las librerías en nuestra carpeta de backend:

```
cd backend  
npm install express  
npm install cors
```

Vemos que están instaladas en el fichero package.json que está dentro de la carpeta backend.

Instalación de la librería para poder trabajar con base de datos MySQL

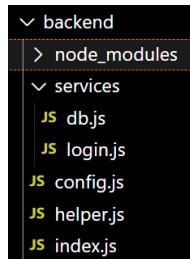
En la carpeta del backend instalamos la librería para trabajar con base de datos MySQL:

```
cd backend  
npm install mysql2
```

Creación del endpoint del login y otro de ejemplo

Para esta parte usas el resto de ficheros que te descargaste del Campus en *Base de datos y Ficheros para el backend* (en la parte de Recursos de UT2). Tienen los ficheros: index.js, helper.js, config.js, db.js y login.js

Deben descargarlos y copiarlos en la carpeta backend del proyecto siguiendo la siguiente estructura:



En la carpeta backend copias los ficheros: config.js, helper.js e index.js

Luego crea la carpeta services dentro de backend. Ahí dentro copias los ficheros db.js y login.js

-Fichero index.js: es donde importamos las librerías que vamos a usar y los ficheros que necesitamos para los endpoints. Definimos el puerto (3030) de nuestra API y también los diferentes *endpoints*.

-Fichero config.js: aquí definimos las credenciales de nuestra base de datos.

-Fichero helper.js: para manejar casos en los que la base de datos no devuelve nada.

-Fichero services/login.js: aquí es donde creamos la consulta para comprobar que los datos proporcionados por el usuario en el frontend se encuentran en la base de datos.

-Fichero services/db.js: aquí creamos la conexión con la base de datos y nos conectamos a la misma realizando la consulta que nos pasan.

Para ejecutar el fichero index.js tenemos dos opciones:

.Escribir node index.js en la consola

.Añadir, en el fichero package.json, en la sección de scripts, la siguiente línea correspondiente a “start”:

```
"scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "start": "node index.js"  
}
```

Un vez hecho esto vamos a probar nuestra API. Iniciamos la API con el siguiente comando, dentro de la carpeta backend:

npm start

Abrimos un navegador en el puerto 3030 que es donde se está ejecutando el backend (la API):
<http://localhost:3030/>

```
{"message": "hola mundo!"}
```

Aquí estamos viendo lo que está programado en el endpoint / que está en el fichero index.js.

Probamos a cambiar el mensaje en el fichero index.js y vemos cómo cambia. Veremos que para aplicar el cambio tenemos que guardar el fichero, parar la ejecución de la API y volver a ejecutarla.

Para no tener que parar la API cada vez que hacemos cambios vamos a instalar el paquete **nodemon** en la carpeta del backend:

```
cd backend  
npm install nodemon
```

Ahora cambio en el package.json el script de start:

```
"start": "nodemon index.js"
```

Cuando ejecuto ahora el proyecto en la consola con **npm start**, vemos que nos permite hacer cambios en la API y no se para:

```
PS C:\Users\Patricia\Documents\PATRICIA\SISTEMAS 24-25\DAD 24-25\ProyectosReact\proyectopatricia\backend> npm start  
> backend@1.0.0 start  
> nodemon index.js  
  
[nodemon] 3.1.7  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node index.js`  
● API escuchando desde el puerto 3030  
[nodemon] restarting due to changes...  
[nodemon] starting `node index.js`  
API escuchando desde el puerto 3030
```

Acceder a los endpoints desde el navegador

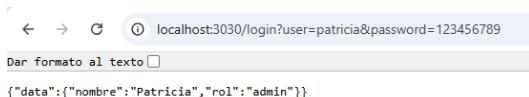
Ya hemos visto en el ejemplo anterior como cuando ponemos la URL: <http://localhost:3030/> estamos accediendo al endpoint /, el cual hemos programado en el fichero index.js

De la misma forma podemos acceder a cualquier endpoint que tengamos programado, como por ejemplo el endpoint de /login. En este endpoint vemos que tenemos que pasarle dos parámetros, el user y el password. Así que en la URL tenemos que poner lo siguiente:

<http://localhost:3030/login?user=patricia&password=123456789>

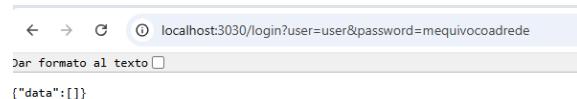
Fíjate que estamos accediendo el endpoint login (/login) y le estamos pasando el parámetro user con el valor patricia y el parámetro password con el valor 123456789. Esto llama a la función que realiza la consulta en la base de datos y nos devuelve los datos de ese usuario que están almacenados en la base de datos.

Vemos que nos devuelve en formato JSON el data que tiene dentro el nombre y el rol.



```
localhost:3030/login?user=patricia&password=123456789
Dar formato al texto □
{"data": {"nombre": "Patricia", "rol": "admin"}}
```

Si me equivoco adrede la base de datos me devuelve datos vacíos:



```
localhost:3030/login?user=user&password=mequivocoadrede
Dar formato al texto □
{"data": []}
```

¿Cómo llamar a los endpoints desde el frontend?

Para llamar a los endpoints desde el frontend se usa un fetch, que nos sirve para acceder y manipular llamadas HTTP. El fetch lo tendremos que colocar en la parte del código donde vamos a comprobar si el usuario y contraseña son los correctos (una vez que piquemos en el botón ACceder).

Ejemplo de llamada al endpoint /login desde el componente <Login /> del frontend y de cómo tenemos que esperar la respuesta (al ser una llamada asíncrona):

```
async function isVerifiedUser () {
    fetch(`http://localhost:3030/login?user=${data.user}&password=${data.passwd}`)
        .then(response => response.json())
        .then (response => {
            console.log('Lo que nos llega de la base de datos: ')
            console.log(response.data)
            if (response.data.length !== 0){
                //Si hay datos es que el usuario y contraseña son los correctos. Hago el dispatch y el
                //navigate
            } else{
                //Si no, realizo la lógica para alertar al usuario con usuario/contraseña son incorrectas
            }
        })
}
```

El fetch se suele poner dentro de una función asíncrona puesto que esperamos por la respuesta de la base de datos.

Con el fetch llamamos al endpoint /login pasándole el login de usuario y la contraseña. En este caso, debemos usar las variables donde se almacenan el login (el nombre que usa el usuario para ingresar en la aplicación) y la contraseña.

Después de llamar espero la respuesta (response) que la paso a json. Si hay respuesta, la mando a la consola para ver lo que trae la base de datos (esto lo tendrías que eliminar cuando la aplicación está en producción). Si la longitud de los datos que trae de la base de datos es distinta de cero es que encontró el usuario y la contraseña, por lo tanto son correctos.