

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: USANDO EL PAQUETE BODY-PARSER.

EXERCISE 1: USANDO EL PAQUETE BODY-PARSER

Principales pasos que se realizarán para crear la API de Libros (To-Do):

1. Crear la ruta de añadir un libro.
2. Crear la ruta de eliminar un libro.
3. Crea la ruta para actualizar un libro.
4. Recepción y respuesta de códigos HTTP.

1. CREAR LA RUTA DE AÑADIR UN LIBRO

Previamente, hemos creado la ruta para obtener todos los libros. Pero, ¿cómo podemos hacer si queremos agregar un libro? En las APIs, para agregar datos, simultáneamente estamos usando el método POST.

Para ello, realizamos lo siguiente:

- Enviar los datos del libro a la API (isbn, nombre, autor)
- Manipular o gestionar la inserción del nuevo libro en la API.
- Enviar un resultado (verdadero o falso)

Agregaremos el siguiente código al index.js:

```
1 // POST se usa para AGREGAR datos en la API, en este caso un libro
2 app.post('/libros', (request, response) => {
3   // TODO: funciones de llenado de un libro
4
5   return response.json({
6     success: false
7   })
8 });
```

Para llenar la función, necesitamos recibir parámetros. Usaremos el middleware body-parser, que es un nuevo paquete NPM para tratarlos de manera eficiente, y lo agregamos al proyecto:

```
1 # Agregando al proyecto el body-parser
2 $ npm install body-parser
```

Para utilizar el paquete en nuestro proyecto, debemos importarlo, y luego indicarle a Express.js que lo está usando. Editaremos el index.js de la siguiente manera:

```
1 // Importando Express.js
2 const express = require("express");
3
4 // Importamos la extensión de body-parser
5 const bodyParser = require('body-parser');
6 ...
7 ...
8 // Indica a Express.js que se estará utilizando un complemento adicional
9 // para tratar parámetros
10 app.use(bodyParser.urlencoded({ extended: true }));
```

La primera solicitud de la función `request`, será útil para acceder al cuerpo de la solicitud. Puede hacer lo siguiente para obtener un parámetro: `request.body.parameterName`.

En este caso, tenemos tres:

```
1 request.body.isbn
2 request.body.titulo
3 request.body.autor
```

En el siguiente código definiremos una función, que permita agregar libros sin que se repitan los *isbn*. Para ello, observemos cómo agregamos la funcionalidad, en el siguiente código del método POST.

```
1 // POST se usa para AGREGAR datos en la API, en este caso un libro
2 app.post('/libros', (request, response) => {
3   // obtenemos los parámetros del isbn
4   const isbnLibro = request.body.isbn;
5
6   // verificamos si el isbn se encuentra en la lista de libros
7   let isbnBusqueda = listaLibros.find(l => l.isbn === isbnLibro);
8 }
```

```
9 // De ser cierto, devolvemos 'false', que no se inserto
10 if (isbnBusqueda) return response.json({
11     success: false
12 })
13
14 // De lo contrario, agregamos el nuevo libro a la lista y retornamos
15 'true'
16 listaLibros.push(request.body);
17 return response.json({
18     success: true
19 });
20
21 });
```

Verificamos nuestro método, pero al utilizarlo en el navegador, se puede notar que no tenemos la lista de libros.

Por defecto, una API es diferente de un sitio web típico. No puedes acceder a él con tu navegador. Cuando se accede a un sitio web con su navegador, se envía una solicitud GET (<http://localhost:3000/libros>)

En este sentido, usaremos Postman, que es la herramienta para interactuar con las API, y su función es actuar como el navegador para ellas.

Iniciamos Postman. Para ello, creamos un nuevo proyecto, y seleccionamos del tipo HTTP Request.

Se puede hacer clic en el icono "+", para crear su primera solicitud. Luego, puede ingresar la URL a solicitar en la entrada correspondiente (<http://localhost:3000/libros>). Es equivalente a escribir una URL en la barra de búsqueda de su navegador.

Junto a la URL, puede OBTENER (**GET**). Corresponde al tipo de petición que estamos haciendo. En este caso, queremos agregar un libro, por lo que es una solicitud **POST**. Haga clic en él, y seleccione **POST**.

Para nuestra solicitud, necesitamos ingresar los parámetros. Seleccionamos el cuerpo (body), y luego en "x-www-form-urlencoded". Para agregar un parámetro, colocamos en tabla los mismos, la "Clave" KEY que es el nombre de su parámetro, y "valor" es el valor.



http://localhost:3000/libros

POST http://localhost:3000/libros

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	isbn	123456			
<input checked="" type="checkbox"/>	titulo	Título del Libro			
<input checked="" type="checkbox"/>	autor	autor del libro			
	Key	Value	Description		

Body **Cookies** Headers (6) Test Results 200 OK 13 ms 487 B Save Response

Y la respuesta de nuestra API es:

Body **Cookies** Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "success": true
3 }
```

Se insertó correctamente el libro. ¿Qué pasará si insertamos un libro con el isbn: '978-1-78398-586-9'? Puede obtener todos los libros a través de Postman, y así verificar si el nuevo libro se insertó, y si está en la lista. Para ello, seguimos los mismos pasos, pero eligiendo el método **GET** como modo de solicitud.

2. CREAR LA RUTA DE ELIMINAR UN LIBRO

Ya creamos el método **GET** y **POST**, y el proceso siempre será el mismo. En esta parte, crearemos un método **DELETE** en **/libros**. Tendremos un parámetro de isbn, y el objetivo de nuestra función será eliminar el libro si está en la lista, y corresponde con el isbn.

```
1 app.delete('/libros', (request, response) => {
2   // obtendremos el parámetro isbn del cuerpo
3   const libroAEliminar = request.body.isbn;
4
5   // Se crea una nueva lista de objetos diferentes al libro a eliminar
6   listaLibros = listaLibros.filter((l) => l.isbn !== libroAEliminar);
7
8   // Retornamos la lista de libros
9   return response.json({
10     todosLibros: listaLibros
11   })
12 });
```

3. CREA LA RUTA PARA ACTUALIZAR UN LIBRO

Para actualizar un libro, en este caso lo haremos con el título del libro o el autor, según su isbn, que es único.

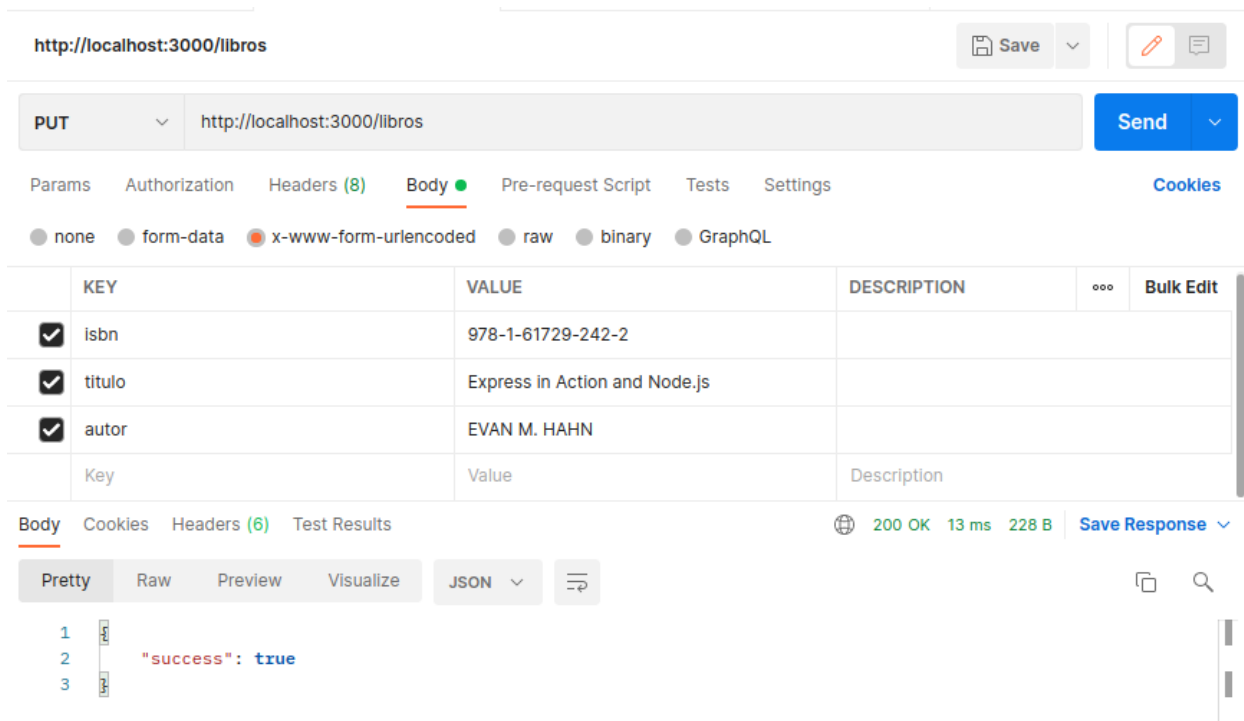
```
1 app.put('/libros', (request, response) => {
2   // obtenemos los parámetros, el isbn a actualizar
3   const libroActualizar = request.body.isbn;
4   const libroActualizado = request.body
5
6   // Buscamos si el libro se encuentra en la lista
7   // de objetos Libros
8   const indiceLibroActualizar = listaLibros.findIndex(
9     (l) => l.isbn === libroActualizar
10  )
11
12  // Si el libro no esta en la lista, retornamos false
13  if (indiceLibroActualizar === -1) return response.json({
14    success: false
15  })
16
17  // Ahora actualizamos el libro los nuevos campos
18  listaLibros[indiceLibroActualizar] = libroActualizado
19  return response.json({
20    success: true
21  })
22 });
```



Tenemos:

```
1 isbn: '978-1-61729-242-2',  
2 titulo: 'Express in Action',  
3 autor: 'EVAN M. HAHN'
```

Lo actualizaremos con su campo título, como *"Express in Action and Node.js"*, y al ejecutar el **PUT** con Postman, observamos lo siguiente:



The screenshot shows the Postman interface for a PUT request to `http://localhost:3000/libros`. The request body is a JSON object with the following data:

KEY	VALUE	DESCRIPTION
isbn	978-1-61729-242-2	
titulo	Express in Action and Node.js	
autor	EVAN M. HAHN	

The response is a 200 OK status with a response time of 13 ms and a body size of 228 B. The response body is a JSON object:

```
{  "success": true }
```

4. RECEPCIÓN Y RESPUESTA DE CÓDIGOS HTTP

En lo particular, podemos gestionar las respuestas de nuestra API según las acciones. Supongamos que el método post, que hemos creado anteriormente, al insertar un *isbn* duplicado responde como false; y el método HTTP de respuesta es 200 según esta función.

```
1 if (isbnBusqueda) return response.json({ success: false })
```

Así mismo, al insertar un libro con un **isbn** que no se encuentra duplicado, responde true; y con una respuesta HTTP de 200, que indica que se insertó correctamente según esta función:

```
1 listaLibros.push(request.body);  
2 return response.json({ success: true });
```

Mejorando la implementación de éste, adecuamos el código con las respuestas correctas de HTTP. Esto sería: devolver una respuesta, y devolver un error 409 de conflicto del lado del cliente cuando inserta un registro duplicado con isbn ya registrado. La nueva función quedaría de la siguiente manera:

```
1 if (isbnBusqueda) return response.status(409).json({  
2   message: 'Conflicto, recurso duplicado por isbn'  
3 });
```

Del mismo modo, si se agrega correctamente un registro, devolvemos un código 201, donde el cliente pasa la solicitud y es procesada en el servidor con éxito. Esto es:

```
1 listaLibros.push(request.body);  
2 return response.status(201).json({  
3   message: 'Se Agrego Correctamente el recurso libro'  
4 });
```

El código actualizado dentro de la función quedaría:

```
1 // POST se usa para AGREGAR datos en la API, en este caso un libro  
2 app.post('/libros', (request, response) => {  
3   // obtenemos los parametros del isbn  
4   const isbnLibro = request.body.isbn;  
5   // verificamos si el isbn se encuentra en la lista de libros  
6   let isbnBusqueda = listaLibros.find(l => l.isbn === isbnLibro);  
7  
8   // De ser cierto, devolvemos 'false', que no se inserto  
9   if (isbnBusqueda) return response.status(409).json({  
10    message: 'Conflicto, recurso duplicado por isbn'  
11  });  
12  // De lo contrario, agregamos el nuevo libro a la lista y retornamos  
13  'true'  
14  listaLibros.push(request.body);  
15  return response.status(201).json({  
16    message: 'Se Agrego Correctamente el recurso libro'  
17  });  
18 });
```

Ejecutamos en el Postman, agregando el siguiente libro:

http://localhost:3000/libros Save

POST ▼ http://localhost:3000/libros Send ▼

Params Auth Headers (8) **Body** ● Pre-req. Tests Settings Cookies

x-www-form-urlencoded ▼

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	isbn	562415			
<input checked="" type="checkbox"/>	titulo	Título del libro 2			
<input checked="" type="checkbox"/>	autor	Autor del libro 2			

Obtenemos la respuesta:

Body Cookies Headers (6) Test Results 201 Created 41 ms 271 B

Pretty Raw Preview Visualize **JSON** ▼

```
1
2  "message": "Se Agrego Correctamente el recurso libro"
3
```

Volviendo a insertar el mismo isbn:

POST ▼ http://localhost:3000/libros Send

Params Auth Headers (8) **Body** ● Pre-req. Tests Settings Cook

x-www-form-urlencoded ▼

	KEY	VALUE	DESCRIPTION	...	Bulk E
<input checked="" type="checkbox"/>	isbn	562415			
<input checked="" type="checkbox"/>	titulo	Título del libro 2			

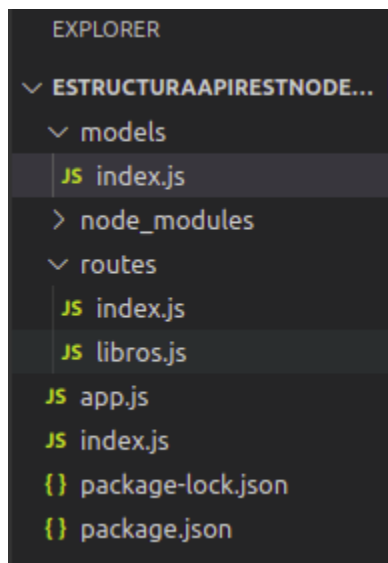
Body Cookies Headers (6) Test Results 409 Conflict 15 ms 269 B Save Respons

Pretty Raw Preview Visualize **JSON** ▼

```
1
2  "message": "Conflicto, recurso duplicado por isbn"
3
```




Inicialmente, la estructura del proyecto contiene en la raíz un archivo **package.json**, donde se especifican la información de nuestro paquete y sus dependencias, y seguidamente, un archivo **index.js** para la conexión a la base de datos y configuración general. Por ejemplo: en mongoose, un archivo **app.js** donde se especificarán las variables necesarias para crear el servidor web con NodeJS, y la configuración de Express; un directorio para los **middlewares**; un directorio de **models**, para crear los modelos y esquemas; un directorio de **controllers**, que contiene las especificaciones de las acciones y operaciones sobre nuestra base de datos; y por último, un directorio de **routes**, que especifican las rutas a las que responderá nuestra aplicación API.



NOTA:

Para una mejor resolución del proyecto, es recomendable tener actualizada la versión de Node.js a la más reciente, y codificar con ES6.