



# 21GIIN PROYECTOS DE PROGRAMACIÓN

Actividad UC3

Moisés Sevilla Corrales  
PROF. Doctor Héctor Simarro Moncholí

15 de enero de 2025

## Índice

Desarrollo de una Aplicación en Arquitectura de 3 Capas .....	3
Objetivo .....	3
Descripción de la Actividad .....	3
Requisitos .....	5
1. Capa de Presentación (UI) POSTMAN .....	5
2. Capa de Lógica de Negocio .....	21
3. Capa de Acceso a Datos.....	27
Entregables.....	30
1. Código Fuente .....	30
2. Documentación .....	30
3. Informe.....	30
Evaluación .....	30
• Funcionalidad.....	30
• Diseño de la UI .....	30
• Calidad del Código.....	30
• Documentación .....	30
• Informe.....	30
Conclusión .....	31
Bibliografía .....	35

<https://github.com/moisessevilla/biblioteca>

## Desarrollo de una Aplicación en Arquitectura de 3 Capas.

### Objetivo

Desarrollar una aplicación de gestión de biblioteca utilizando la arquitectura de 3 capas (presentación, lógica de negocio y acceso a datos) para comprender y aplicar los principios de diseño modular y separación de responsabilidades.

### Descripción de la Actividad

Los alumnos deberán crear una aplicación que permita gestionar los libros de una biblioteca. La aplicación debe incluir funcionalidades para agregar, editar, eliminar y buscar libros. Además, debe permitir la gestión de usuarios que pueden tomar prestados los libros.

A modo de ejemplo se nos propone realizar una estructura similar vista en las sesiones.

### Microproyecto de Tres Capas en Desarrollo Web

Este microproyecto ilustra una aplicación simple de tres capas en desarrollo web. Las capas incluyen la capa de presentación del cliente, la capa lógica de negocios en el servidor y la capa de datos en el servidor.

#### Estructura del Proyecto

```
proyecto-tres-capas/  
├── cliente/  
│   └── index.html  
├── servidor/  
│   ├── node_modules/  
│   ├── server.js  
│   ├── package.json  
│   └── .gitignore  
└── datos/  
    └── datos.js
```

Tras instalar MySQL, MySQL Workbench, Postman y Node.js iniciamos el proyecto con:

“**npm init -y**”, de inmediato nos genera el fichero package.json con este contenido.

```
D:\biblioteca>npm init -y
Wrote to D:\biblioteca\package.json:

{
  "name": "biblioteca",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Ahora instalamos los módulos necesarios para el framework Express:

“**npm install express mysql body-parser cors**”

```
D:\biblioteca>npm install express mysql body-parser cors
added 82 packages, and audited 83 packages in 3s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- Ahora configuramos nuestro fichero de conexión “**server.js**”, con los datos necesarios para realizar el vínculo, aprovechando el fichero para configurarlo como API REST y agregar endpoints a modo de ejemplo que se podrían ampliar para el resto de casos por no extender mucho el código en esta demo y lo iniciamos mediante el comando:

“**node server.js**”, si todo está correcto debería mostrar este mensaje:

```
D:\biblioteca>node server.js
Servidor escuchando en http://localhost:3000
Conectado a la base de datos MySQL
```

## Requisitos

### 1. Capa de Presentación (UI) POSTMAN

- a. Desarrollar controladores que deben permitir a los usuarios realizar las siguientes acciones:

En esta actividad vamos a usar **POSTMAN** con una configuración estándar para todos los casos URL: <http://localhost:3000> y `/libros` o `/usuarios` dependiendo del caso.



<http://localhost:3000/libros>



<http://localhost:3000/usuarios>

El Headers lo configuraremos así:

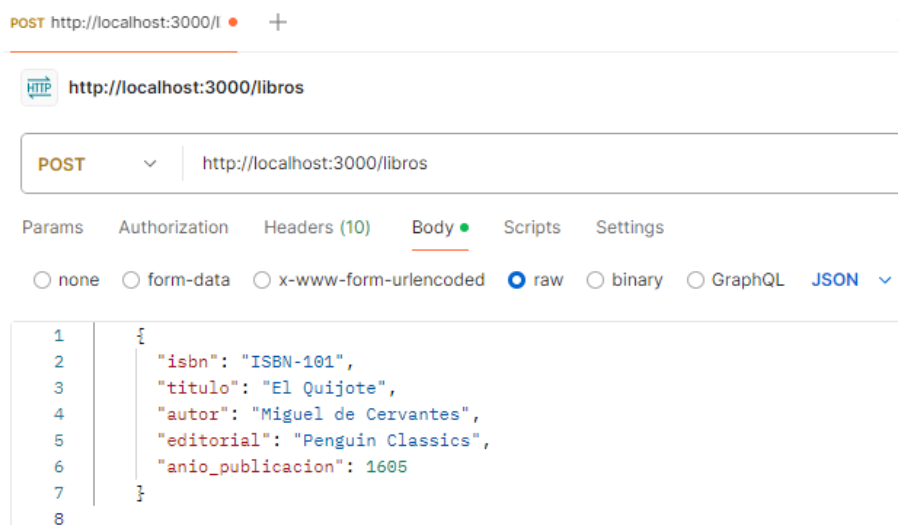
Params   Authorization   **Headers (10)**   Body ●   Scripts   Settings

Headers   9 hidden

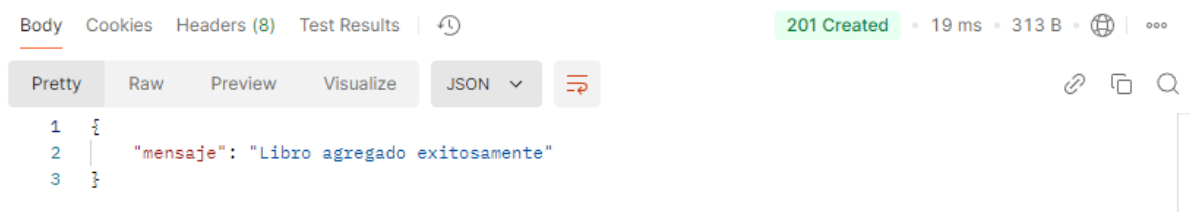
	Key	Value
<input checked="" type="checkbox"/>	Content-Type	application/json

i. Agregar un nuevo libro.

Seleccionamos el método **POST**, colocamos la ruta URL y cargamos en el campo Body, raw, JSON el contenido que queramos del libro, por ejemplo: (Fichero: **libros.json**)



Nos devuelve la respuesta:



Repetimos el proceso para agregar un ejemplo de 10 libros.

Result Grid						
Filter Rows:						
	id	isbn	titulo	autor	editorial	anio_publicacion
▶	1	ISBN-101	El Quijote	Miguel de Cervantes	Penguin Classics	1605
	2	ISBN-102	Cien Años de Soledad	Gabriel García Márquez	Sudamericana	1967
	3	ISBN-103	Rayuela	Julio Cortázar	Editorial Sudamericana	1963
	4	ISBN-104	La Ciudad y los Perros	Mario Vargas Llosa	Seix Barral	1962
	5	ISBN-105	El Amor en los Tiempos del Cólera	Gabriel García Márquez	Oveja Negra	1985
	6	ISBN-106	La Sombra del Viento	Carlos Ruiz Zafón	Planeta	2001
	7	ISBN-107	Pedro Páramo	Juan Rulfo	FCE	1955
	8	ISBN-108	La Muerte de Artemio Cruz	Carlos Fuentes	FCE	1962
	9	ISBN-109	El Túnel	Ernesto Sabato	Seix Barral	1948
	10	ISBN-110	Ficciones	Jorge Luis Borges	Emecé	1944

ii. Editar la información de un libro existente.

Ahora seleccionamos **PUT**, y cargamos en la URL el ID del libro que queremos modificar y cargamos en el JSON los nuevos valores

Antes:

3	ISBN-103	Rayuela	Julio Cortázar	Editorial Sudamericana	1963
---	----------	---------	----------------	------------------------	------

Después:

3	ISBN-103	Rayuela - El retorno	Julio Cortázar	Editorial Sudamericana	2024
---	----------	----------------------	----------------	------------------------	------

PUT http://localhost:3000/lib/

HTTP http://localhost:3000/libros/3

PUT http://localhost:3000/libros/3

Params Authorization Headers (10) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☒ JSON

```
1 {  
2   "isbn": "ISBN-103",  
3   "titulo": "Rayuela - El retorno",  
4   "autor": "Julio Cortázar",  
5   "editorial": "Editorial Sudamericana",  
6   "anio_publicacion": 2024  
7 }  
8
```

En caso de indicar una ID de un libro que no existe nos devolverá este valor:

Body Cookies Headers (8) Test Results

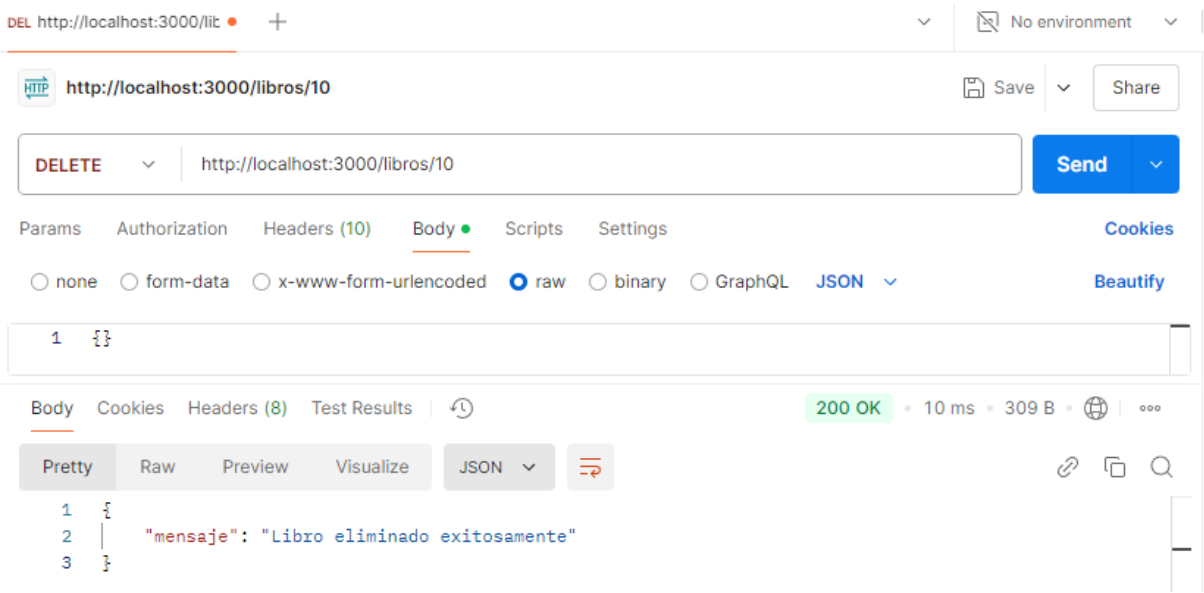
404 Not Found • 3 ms • 307

Pretty Raw Preview Visualize JSON

```
1 {  
2   "mensaje": "Libro no encontrado"  
3 }
```

iii. Eliminar un libro.

Seleccionamos ahora **DELETE**, el id del libro que queremos eliminar:

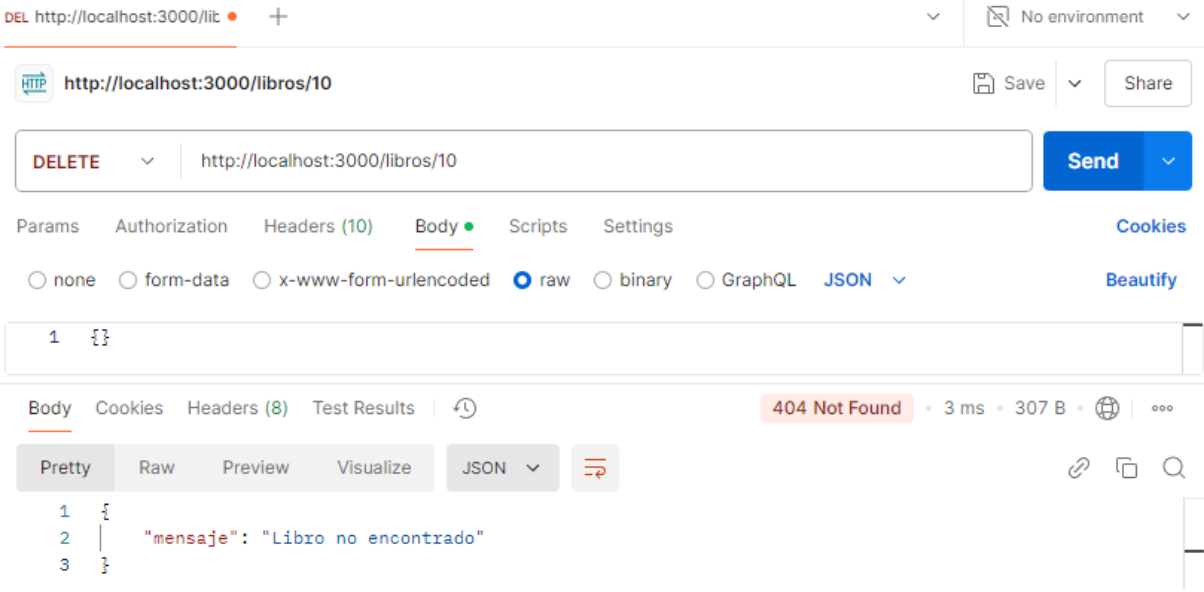


The screenshot shows a REST client interface with the following details:

- Method: **DELETE**
- URL: `http://localhost:3000/libros/10`
- Status: **200 OK** (10 ms, 309 B)
- Response Body (JSON):

```
{
  "mensaje": "Libro eliminado exitosamente"
}
```

En caso de lanzar de nuevo la petición ahora nos mostrará un mensaje de “Libro no encontrado”, ya que fue eliminado correctamente anteriormente.



The screenshot shows the same REST client interface after a second request:

- Method: **DELETE**
- URL: `http://localhost:3000/libros/10`
- Status: **404 Not Found** (3 ms, 307 B)
- Response Body (JSON):

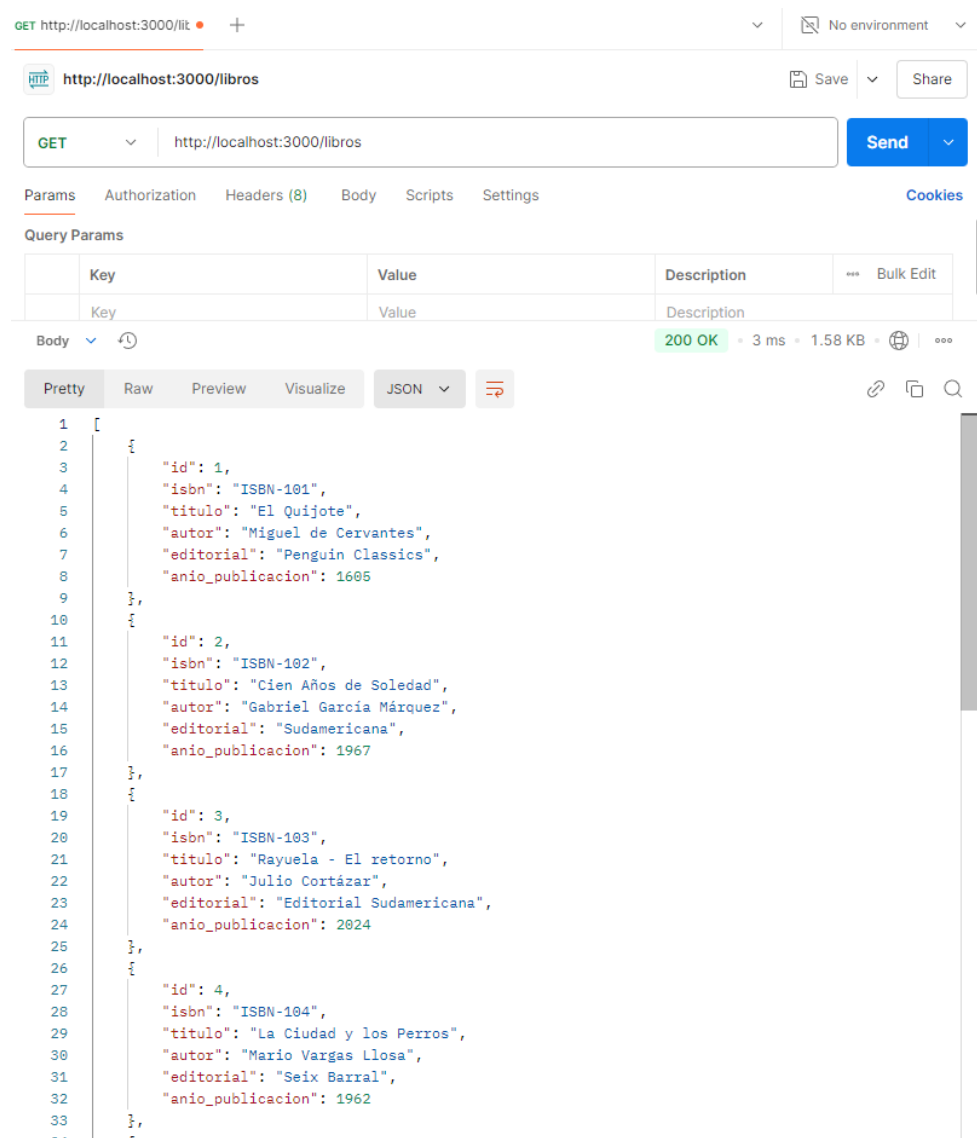
```
{
  "mensaje": "Libro no encontrado"
}
```



iv. Buscar libros por título, autor o ISBN.

Para las búsquedas seleccionamos **GET** y en la URL seleccionamos el campo y el criterio de búsqueda.

Si no aplicamos filtros muestra la tabla entera. Haciendo scroll podríamos ver todos los registros, pero para el ejemplo dejamos esta captura.



The screenshot shows a REST client interface with a GET request to `http://localhost:3000/libros`. The response is a JSON array of 4 book objects. The status is 200 OK, with a response time of 3 ms and a size of 1.58 KB.

Key	Value	Description
Key	Value	Description

```
1 [
2   {
3     "id": 1,
4     "isbn": "ISBN-101",
5     "titulo": "El Quijote",
6     "autor": "Miguel de Cervantes",
7     "editorial": "Penguin Classics",
8     "anio_publicacion": 1605
9   },
10  {
11    "id": 2,
12    "isbn": "ISBN-102",
13    "titulo": "Cien Años de Soledad",
14    "autor": "Gabriel García Márquez",
15    "editorial": "Sudamericana",
16    "anio_publicacion": 1967
17  },
18  {
19    "id": 3,
20    "isbn": "ISBN-103",
21    "titulo": "Rayuela - El retorno",
22    "autor": "Julio Cortázar",
23    "editorial": "Editorial Sudamericana",
24    "anio_publicacion": 2024
25  },
26  {
27    "id": 4,
28    "isbn": "ISBN-104",
29    "titulo": "La Ciudad y los Perros",
30    "autor": "Mario Vargas Llosa",
31    "editorial": "Seix Barral",
32    "anio_publicacion": 1962
33  },
34 ]
```

### Búsqueda por título:

GET http://localhost:3000/libros?titulo=amor

Params • Authorization Headers (8) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	titulo	amor			
	Key	Value	Description		

Body 200 OK • 3 ms • 427 B

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 5,
4     "isbn": "ISBN-105",
5     "titulo": "El Amor en los Tiempos del Cólera",
6     "autor": "Gabriel Garcia Márquez",
7     "editorial": "Oveja Negra",
8     "anio_publicacion": 1985
9   }
10 ]
```

### Búsqueda por autor:

GET http://localhost:3000/libros?autor=Ernesto

Params • Authorization Headers (8) Body Scripts Settings Cookies

Query Params

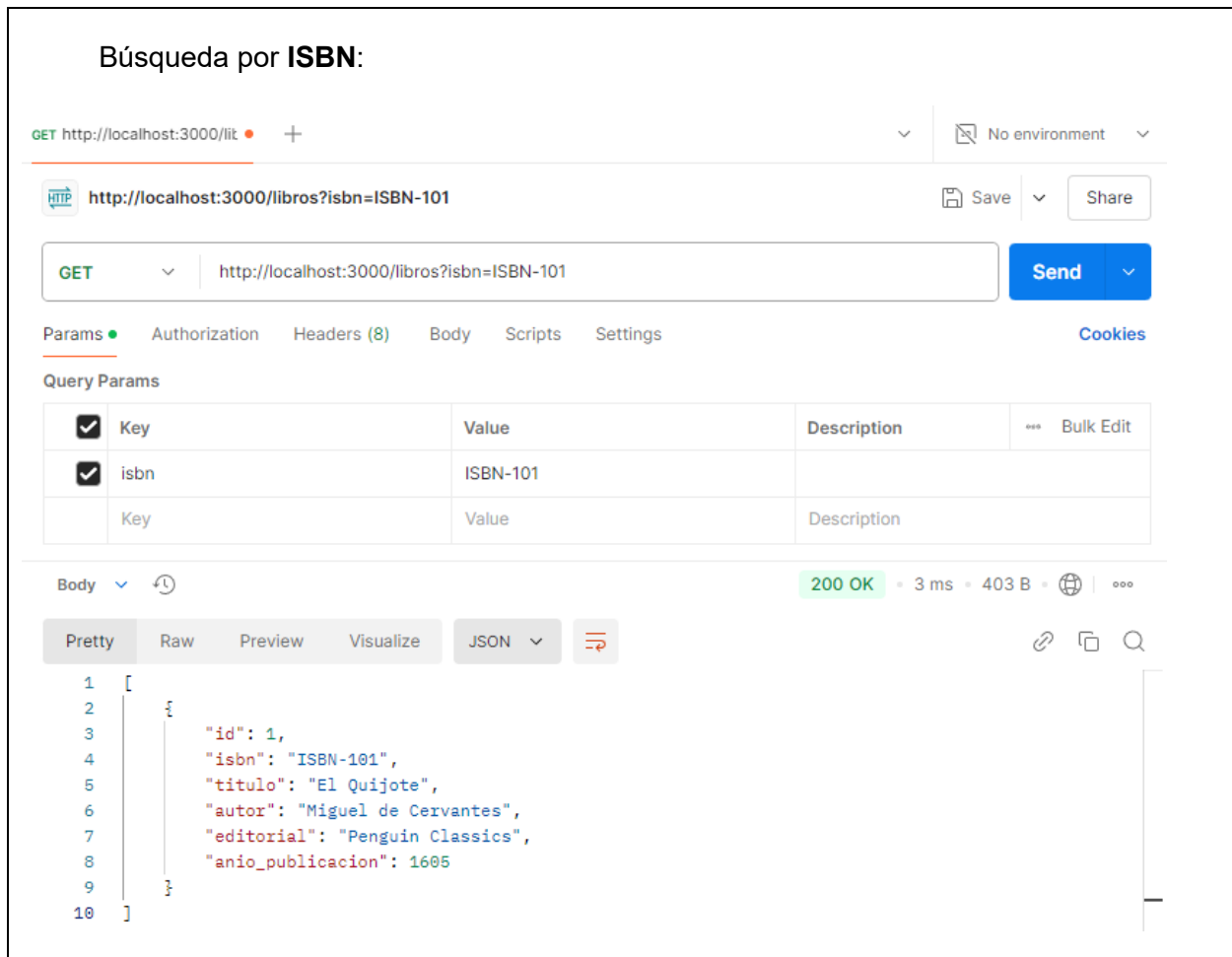
<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	autor	Ernesto			
	Key	Value	Description		

Body 200 OK • 3 ms • 392 B

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 9,
4     "isbn": "ISBN-109",
5     "titulo": "El Túnel",
6     "autor": "Ernesto Sabato",
7     "editorial": "Seix Barral",
8     "anio_publicacion": 1948
9   }
10 ]
```

Búsqueda por ISBN:



GET http://localhost:3000/libros?isbn=ISBN-101

Params • Authorization Headers (8) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	isbn	ISBN-101			
	Key	Value	Description		

Body 200 OK • 3 ms • 403 B

Pretty Raw Preview Visualize JSON

```

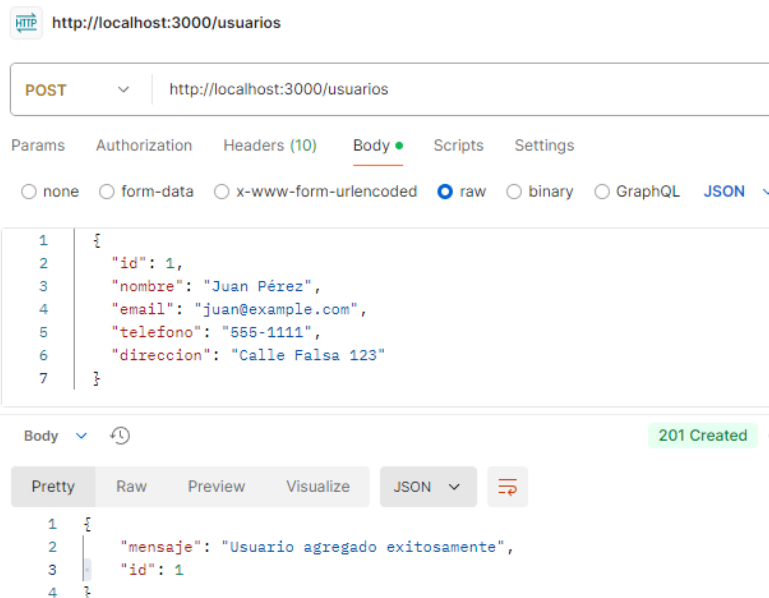
1  [
2    {
3      "id": 1,
4      "isbn": "ISBN-101",
5      "titulo": "El Quijote",
6      "autor": "Miguel de Cervantes",
7      "editorial": "Penguin Classics",
8      "anio_publicacion": 1605
9    }
10 ]
    
```

v. Registrar y gestionar usuarios de la biblioteca.

El tratamiento en postman para esta parte sería la misma que para los libros, pero usando la tabla de usuario, pero utilizando los campos correspondientes de la tabla usuarios.

Para no repetirse de nuevo, se hace un breve resumen recordatorio.

Para agregar, usamos **POST**: (Repetimos el proceso con 10 usuarios a modo de ejemplo)



HTTP `http://localhost:3000/usuarios`

**POST** `http://localhost:3000/usuarios`

Params Authorization Headers (10) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▾

```
1 {
2   "id": 1,
3   "nombre": "Juan Pérez",
4   "email": "juan@example.com",
5   "telefono": "555-1111",
6   "direccion": "Calle Falsa 123"
7 }
```

Body ▾ ↻ **201 Created**

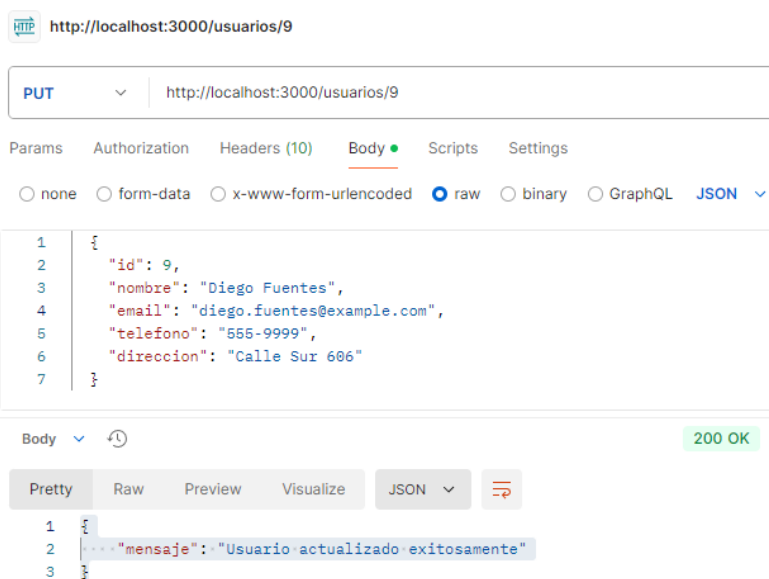
Pretty Raw Preview Visualize JSON ▾ ↻

```
1 {
2   "mensaje": "Usuario agregado exitosamente",
3   "id": 1
4 }
```

Para editar usamos **PUT**. Por ejemplo, editamos el email.

Antes: `9` `Diego Fuentes` `diego@example.com` `555-9999` `Calle Sur 606`

Después: `9` `Diego Fuentes` `diego.fuentes@example.com` `555-9999` `Calle Sur 606`



HTTP `http://localhost:3000/usuarios/9`

**PUT** `http://localhost:3000/usuarios/9`

Params Authorization Headers (10) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▾

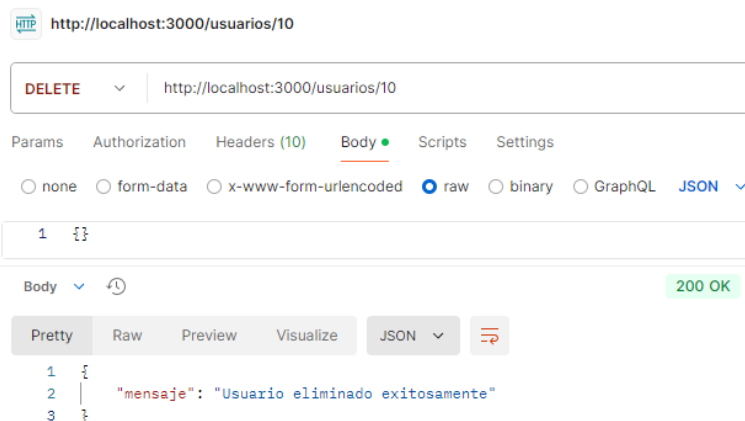
```
1 {
2   "id": 9,
3   "nombre": "Diego Fuentes",
4   "email": "diego.fuentes@example.com",
5   "telefono": "555-9999",
6   "direccion": "Calle Sur 606"
7 }
```

Body ▾ ↻ **200 OK**

Pretty Raw Preview Visualize JSON ▾ ↻

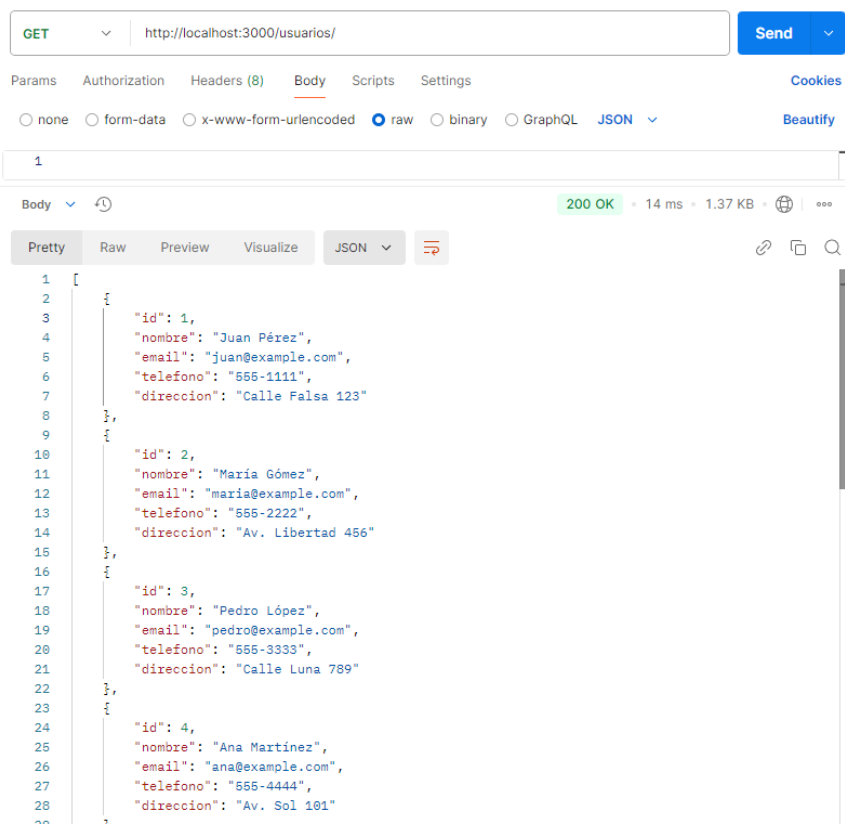
```
1 {
2   "mensaje": "Usuario actualizado exitosamente"
3 }
```

Para eliminar usamos **DELETE**. Por ejemplo, eliminamos el registro 10.



Para buscar usamos **GET** sin o con la condición de búsqueda deseada, en esta ocasión podemos filtrar por id, nombre o teléfono.

Utilizamos un **GET** sin filtros, nos muestra toda la tabla de usuarios:



### Una búsqueda por id: Ejemplo, **id=4**

GET http://localhost:3000/usuarios?id=4

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

1

Body 200 OK

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 4,
4     "nombre": "Ana Martínez",
5     "email": "ana@example.com",
6     "telefono": "555-4444",
7     "direccion": "Av. Sol 101"
8   }
9 ]
```

### Una búsqueda por nombre: Ejemplo, **nombre=Rodrigo**

GET http://localhost:3000/usuarios?nombre=Rodrigo

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

1

Body 200 OK

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 7,
4     "nombre": "Rodrigo Díaz",
5     "email": "rodrigo@example.com",
6     "telefono": "555-7777",
7     "direccion": "Calle Central 404"
8   }
9 ]
```

Una búsqueda por telefono: Ejemplo, **telefono=555-3333**

The screenshot shows a web browser interface for an HTTP client. The address bar displays the URL `http://localhost:3000/usuarios?telefono=555-3333`. The method is set to **GET**. Below the address bar, there are tabs for **Params**, **Authorization**, **Headers (8)**, **Body**, **Scripts**, and **Settings**. The **Body** tab is selected, and the content type is set to **raw**. The response status is **200 OK**. The response body is displayed in a **JSON** format, showing a single user object:

```
1 [
2   {
3     "id": 3,
4     "nombre": "Pedro López",
5     "email": "pedro@example.com",
6     "telefono": "555-3333",
7     "direccion": "Calle Luna 789"
8   }
9 ]
```

vi. Registrar préstamos y devoluciones de libros.

Agregamos los endpoint “para registrar un préstamo y una devolución al fichero **server.js**

Registramos un préstamo con **POST**:

HTTP <http://localhost:3000/prestamos>

**POST** <http://localhost:3000/prestamos>

Params Authorization Headers (10) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "usuarioId": 1,
3   "libroId": 3
4 }
```

Body **201 Created** • 18 ms

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "mensaje": "Préstamo registrado exitosamente",
3   "prestamoId": 1
4 }
```

Definimos que el máximo de libros que puede tener cada usuario sea 2 mediante la variable:

**const maxPrestamos = 2;**



Result Grid

Filter Rows:

Edit:

Export/Import:

	id	usuario_id	libro_id	fecha_prestamo	fecha_devolucion	devuelto
▶	1	1	3	2024-12-28 22:19:30	NULL	0
	2	1	1	2024-12-28 22:24:05	NULL	0
✱	NULL	NULL	NULL	NULL	NULL	NULL

GET

http://localhost:3000/pi

+

HTTP

http://localhost:3000/prestamos?usarioid=1

GET

▼

http://localhost:3000/prestamos?usarioid=1

Params

●

Authorization

Headers (8)

Body

Scripts

Settings

○ none

○ form-data

○ x-www-form-urlencoded

● raw

○ binary

○ GraphQL

JSON

1

Body

▼

↺

200 OK

Pretty

Raw

Preview

Visualize

JSON

▼

≡

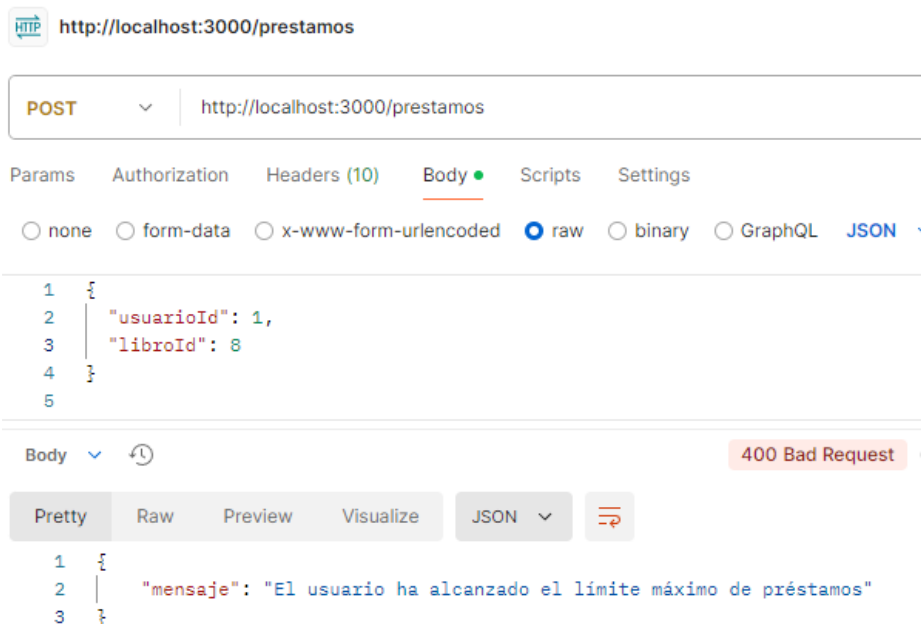
```

1  [
2    {
3      "id": 1,
4      "usuario_id": 1,
5      "libro_id": 3,
6      "fecha_prestamo": "2024-12-28T21:19:30.000Z",
7      "fecha_devolucion": null,
8      "devuelto": 0
9    },
10   {
11     "id": 2,
12     "usuario_id": 1,
13     "libro_id": 1,
14     "fecha_prestamo": "2024-12-28T21:24:05.000Z",
15     "fecha_devolucion": null,
16     "devuelto": 0
17   }
18 ]

```

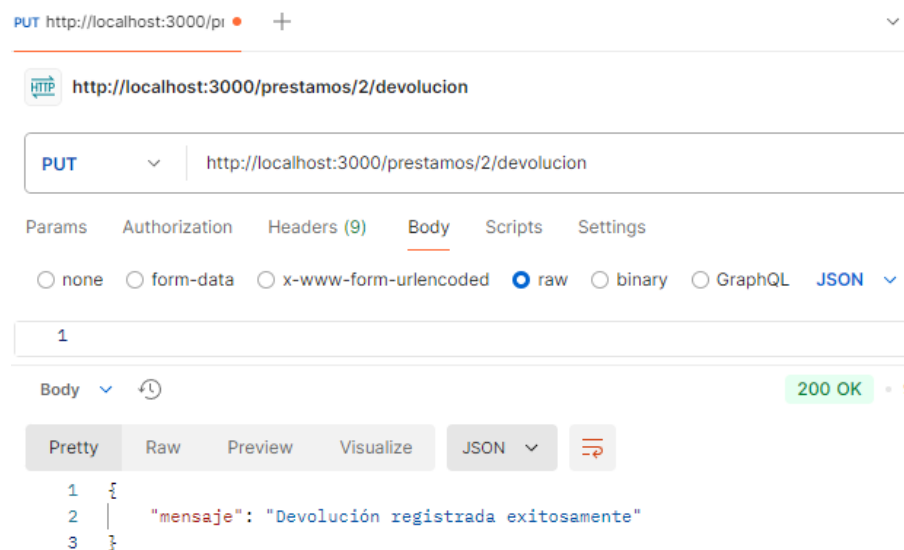
Tras alcanzar el límite, si intenta coger otro libro prestado nos alertara con el mensaje:

**“El usuario ha alcanzado el límite máximo de préstamos”.**



Para registrar la fecha y el estado de devolución del libro seleccionamos **PUT**:

Por ejemplo **“http://localhost:3000/prestamos/2/devolucion”**



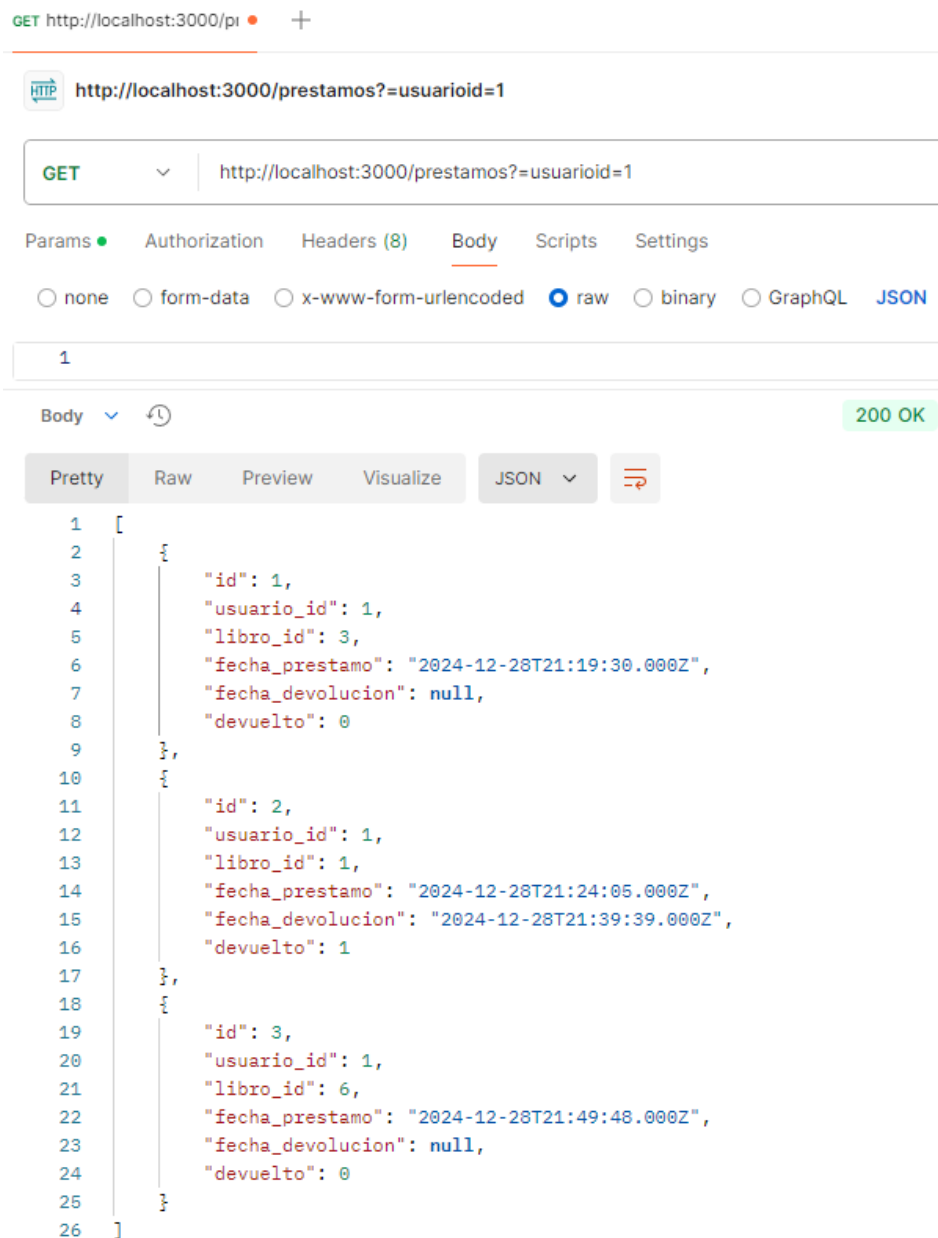
Podemos comprobar, que se ha registrado correctamente la fecha de devolución en **id=2** y el campo **devuelto=1**:

```
Body 200 OK
Pretty Raw Preview Visualize JSON
1 [
2   {
3     "id": 1,
4     "usuario_id": 1,
5     "libro_id": 3,
6     "fecha_prestamo": "2024-12-28T21:19:30.000Z",
7     "fecha_devolucion": null,
8     "devuelto": 0
9   },
10  {
11    "id": 2,
12    "usuario_id": 1,
13    "libro_id": 1,
14    "fecha_prestamo": "2024-12-28T21:24:05.000Z",
15    "fecha_devolucion": "2024-12-28T21:39:39.000Z",
16    "devuelto": 1
17  }
18 ]
```

Ahora el usuario que tenía el límite alcanzado de 2 libros que no podía tener más libros prestados, puede tener otro libro, realizamos la comprobación:

```
HTTP http://localhost:3000/prestamos/
POST http://localhost:3000/prestamos/
Params Authorization Headers (10) Body Scripts Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "usuarioId": 1,
3   "libroId": 6
4 }
Body 201 Created
Pretty Raw Preview Visualize JSON
1 {
2   "mensaje": "Préstamo registrado exitosamente",
3   "prestamoId": 3
4 }
```

Realizamos la consulta para verificar y vemos que tiene un registro de 2 libros sin devolver y 1 devuelto por lo que el registro esta funcionando con éxito, se pueden ampliar más filtros de búsqueda como que solo liste los libros no devueltos y todo el historial, pero esto es a modo de ejemplo.



```
GET http://localhost:3000/prestamos?=usuarioid=1

GET http://localhost:3000/prestamos?=usuarioid=1

Params Authorization Headers (8) Body Scripts Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Body 200 OK

Pretty Raw Preview Visualize JSON

1 [
2   {
3     "id": 1,
4     "usuario_id": 1,
5     "libro_id": 3,
6     "fecha_prestamo": "2024-12-28T21:19:30.000Z",
7     "fecha_devolucion": null,
8     "devuelto": 0
9   },
10  {
11    "id": 2,
12    "usuario_id": 1,
13    "libro_id": 1,
14    "fecha_prestamo": "2024-12-28T21:24:05.000Z",
15    "fecha_devolucion": "2024-12-28T21:39:39.000Z",
16    "devuelto": 1
17  },
18  {
19    "id": 3,
20    "usuario_id": 1,
21    "libro_id": 6,
22    "fecha_prestamo": "2024-12-28T21:49:48.000Z",
23    "fecha_devolucion": null,
24    "devuelto": 0
25  }
26 ]
```

## 2. Capa de Lógica de Negocio

- a. Implementar la lógica de negocio para gestionar las operaciones de la biblioteca.

Fichero: **logicaNegocio.js**

- b. Validar los datos ingresados por los usuarios (por ejemplo, verificar que el ISBN sea único).

Ficheros: **validarUsuario.js** y **validarLibro.js**

- c. Gestionar las reglas de negocio, como la cantidad máxima de libros que un usuario puede tomar prestados.

Ficheros: **verificarPrestamo.js** y **registrarPrestamo.js**

## 1. Funciones principales utilizadas:

### I. **esISBNUnico(isbn):**

- Verifica si un ISBN ya está registrado en la base de datos.
- Utiliza una consulta SQL `SELECT COUNT(*)` para contar cuántos libros existen con el ISBN dado.

### II. **puedePrestar(usuariold):**

- Comprueba si un usuario ha alcanzado el límite máximo de libros prestados (2 préstamos activos).
- Utiliza una consulta SQL `SELECT COUNT(*)` para contar los préstamos activos.

### III. **registrarPrestamo(usuariold, libroid):**

- Registra un nuevo préstamo en la tabla prestamos.
- Valida previamente si el usuario puede tomar más préstamos, invocando `puedePrestar(usuariold)`.

### IV. **validarUsuario(datosUsuario):**

- Valida los datos de un usuario (nombre y email obligatorios).
- Lanza errores descriptivos en caso de que los datos sean inválidos.

### V. **agregarLibro(datosLibro):**

- Valida que el ISBN, título y autor sean válidos antes de insertar un nuevo libro.
- Verifica que el ISBN sea único antes de agregarlo a la base de datos.

## 2. Validar los datos ingresados por los usuarios

### Validaciones utilizadas:

#### VI. Validación del ISBN único:

- Función: esISBNUnico(isbn)
- Comprueba en la base de datos que el ISBN no esté ya registrado.

#### VII. Validación de datos obligatorios (usuario):

- Función: validarUsuario(datosUsuario)
- Comprueba que:
  - nombre no sea nulo ni vacío.
  - email sea válido y contenga un @.

#### VIII. Validación de datos obligatorios (libro):

- Comprueba que:
  - isbn, titulo y autor no sean nulos ni vacíos.
  - isbn sea único mediante esISBNUnico(isbn).

### **3. Gestionar las reglas de negocio**

#### **Reglas implementadas:**

##### **I. Cantidad máxima de préstamos por usuario:**

- Regla: Un usuario puede tener un máximo de 3 libros prestados al mismo tiempo.
- Función: puedePrestar(usuarioId)
- Valida que el número de préstamos activos (devuelto = 0) no exceda el límite.

##### **II. Registro de préstamos:**

- Función: registrarPrestamo(usuarioId, libroId)
- Valida si el usuario puede realizar el préstamo antes de registrar uno nuevo.

##### **III. Validación de datos antes de operaciones:**

- Los datos de usuario y libro son validados antes de insertarse o actualizarse en la base de datos, para evitar inconsistencias.



#### 4. Técnicas y Herramientas Utilizadas

##### I. Manejo de Errores:

- Uso de excepciones (throw new Error) para gestionar errores en las validaciones y reglas de negocio.

```
if (!isbn || isbn.trim() === "") {  
    throw new Error('El ISBN es obligatorio');  
}
```

##### II. Consultas SQL:

- Consultas dinámicas para validar y gestionar operaciones.
- Verificar ISBN único:

```
SELECT COUNT(*) AS count FROM libros WHERE isbn = ?
```

- Contar préstamos activos:

```
SELECT COUNT(*) AS count FROM prestamos WHERE usuario_id = ? AND devuelto = 0
```

##### III. Modularidad:

- Separación de funciones en un archivo dedicado a la lógica de negocio (logicaNegocio.js), desacoplado de las capas de presentación y acceso a datos.

##### IV. Integración con la Base de Datos:

- Uso de callbacks para manejar las operaciones de la base de datos de manera asíncrona.

##### V. Pruebas y Validaciones:

- Pruebas de cada función individualmente con diferentes casos de uso.
- Validación de datos en el servidor antes de interactuar con la base de datos.

A modo de resumen podríamos clasificarlo como:

Apartado	Funciones / Reglas Implementadas
Lógica de negocio para operaciones	esISBNUnico, puedePrestar, registrarPrestamo, validarUsuario, agregarLibro
Validación de datos ingresados	Validaciones de ISBN único, campos obligatorios (nombre, email, etc.)
Reglas de negocio	Límite de préstamos, validación previa de datos, manejo de errores
Técnicas y herramientas	Excepciones, consultas SQL dinámicas, modularidad, integración con la base de datos

### 3. Capa de Acceso a Datos

- a. Diseñar y crear una base de datos para almacenar la información de los libros y usuarios.

- Creamos la base de datos “**biblioteca.sql**” que contiene las tablas de **libros**, **usuarios** y **prestamos**, con un script en **MySQL Workbench**.



- La tabla “**libros**” contiene el id, isbn, titulo, autor, editorial, anio\_publicación. En este caso ponemos como clave única que no se repita como nos indica la actividad la columna de “**isbn**”.

Table Name:  Schema: **biblioteca**

Charset/Collation:   Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
isbn	VARCHAR(20)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
titulo	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
autor	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
editorial	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
anio_publicacion	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

- La tabla “**usuarios**” contiene el id, nombre, email, teléfono y dirección.

En este caso la clave única que no se repite es la columna “**email**”.

Table Name:  Schema: **biblioteca**

Charset/Collation:   Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
nombre	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
email	VARCHAR(100)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
telefono	VARCHAR(20)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
direccion	VARCHAR(150)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

- La tabla “**prestamos**” contiene el id, usuario\_id, libro\_id, fecha\_prestamo, fecha\_devolucion, devuelto.

Table Name:  Schema: **biblioteca**

Charset/Collation:   Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
usuario_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
libro_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
fecha_prestamo	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
fecha_devolucion	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
devuelto	TINYINT(1)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'0'

- b. Implementar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para los libros y usuarios.

- Generamos el fichero “**server.js**” para los CRUD que contiene la configuración de la base de datos para su conexión.
- Se generan endpoints específicos para cada caso de los CRUD para las tablas libros y usuarios:

Crear (POST)

Leer todos sin filtro o por titulo, autor o ISBN. (GET) (tabla libros)

Leer todos sin filtro o por id, nombre o teléfono. (GET) (tabla usuarios)

Actualizar (PUT)

Eliminar (DELETE)

- c. Asegurar la integridad y consistencia de los datos.

Aseguramos la integridad en dos niveles.

**BD:** Campos, *UNIQUE*, *NOT NULL*, tipos de datos adecuados.

**Servidor:** Validaciones antes de realizar la operación, (por ejemplo:

*if (!isbn || !titulo || !autor)).*

## Entregables

1. **Código Fuente:** El código fuente completo de la aplicación, organizado en las tres capas mencionadas.
2. **Documentación:** Incluir una breve documentación que describa la arquitectura de la aplicación, las tecnologías utilizadas y las instrucciones para ejecutar la aplicación.
3. **Informe:** Un informe que detalle el proceso de desarrollo, los desafíos encontrados y cómo se resolvieron.

## Evaluación

La evaluación se basará en los siguientes criterios:

- **Funcionalidad:** La aplicación cumple con todos los requisitos funcionales especificados.
- **Diseño de la UI:** La interfaz es intuitiva y fácil de usar. (**POSTMAN**)
- **Calidad del Código:** El código está bien estructurado, comentado y sigue buenas prácticas de programación.
- **Documentación:** La documentación es clara y completa.
- **Informe:** El informe es detallado y refleja el proceso de desarrollo y resolución de problemas.

## Conclusión

Esta actividad nos ha permitido diseñar, desarrollar e implementar una aplicación de gestión de biblioteca utilizando una arquitectura de 3 capas (presentación, lógica de negocio y acceso a datos). Este enfoque nos ha facilitado la separación de responsabilidades, promoviendo la modularidad, escalabilidad y mantenibilidad del sistema. A continuación, mostramos los logros y aprendizajes obtenidos durante el desarrollo:

### 1. Organización y modularidad

Se estructuró el proyecto en componentes claros:

- **Capa de presentación:** Se desarrollaron endpoints RESTful que permiten realizar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y otras funcionalidades específicas como la búsqueda con filtros, registro de préstamos y devoluciones. Todo esto se probó mediante JSON y Postman, garantizando una interfaz efectiva para interactuar con la API.
- **Capa de lógica de negocio:** Aquí se implementaron las validaciones necesarias para asegurar la integridad de los datos y se definieron reglas de negocio como el límite máximo de libros que un usuario puede tomar prestados.
- **Capa de acceso a datos:** A través de consultas SQL, se garantizó una comunicación robusta con la base de datos MySQL, incluyendo la creación de tablas y la introducción de datos iniciales.

## 2. Desarrollo de funcionalidades clave

Entre las funcionalidades destacadas, se encuentran:

- **Gestión de libros:** Incluye la validación del ISBN único, búsqueda por criterios, y operaciones CRUD completas.
- **Gestión de usuarios:** Asegura la validación de datos obligatorios y permite la creación, edición, eliminación y búsqueda de usuarios por filtros.
- **Gestión de préstamos y devoluciones:** Implementa reglas de negocio para controlar el límite máximo de préstamos por usuario, así como el registro de devoluciones con actualización automática del estado del préstamo.

## 3. Herramientas y tecnologías utilizadas

- **Backend:** Node.js con Express.js para la creación del servidor y manejo de rutas.
- **Base de datos:** MySQL, junto con MySQL Workbench para la gestión y diseño de tablas.
- **Middleware:** body-parser y cors para manejar solicitudes HTTP y habilitar conexiones externas.
- **Postman:** Para probar los endpoints de manera exhaustiva con diferentes escenarios y validaciones.
- **Arquitectura modular:** Organización del proyecto en carpetas específicas para lógica de negocio, rutas y configuración, facilitando la mantenibilidad y escalabilidad.

## 4. Retos superados

- **Validaciones complejas:** Asegurar la integridad de los datos mediante validaciones como el ISBN único, datos obligatorios y reglas de negocio para préstamos.



- **Gestión de errores:** Manejo adecuado de errores tanto en la interacción con la base de datos como en la lógica del servidor, garantizando que las respuestas sean claras para el usuario.
- **Búsquedas dinámicas:** Implementación de filtros flexibles para usuarios y libros, permitiendo búsquedas parciales o específicas según múltiples criterios.

## 5. Impacto y aprendizajes

La implementación de esta actividad ha reforzado conocimientos fundamentales en:

- **Diseño de arquitecturas modulares:** Asegurando la separación de responsabilidades.
- **Interacción con bases de datos relacionales:** Desde la creación de tablas hasta la ejecución de consultas dinámicas.
- **Desarrollo de APIs RESTful:** Con un enfoque en las mejores prácticas, manejando errores y validaciones de manera efectiva.
- **Pruebas y depuración:** Uso de Postman y logs detallados para validar el comportamiento de la aplicación en diferentes escenarios.

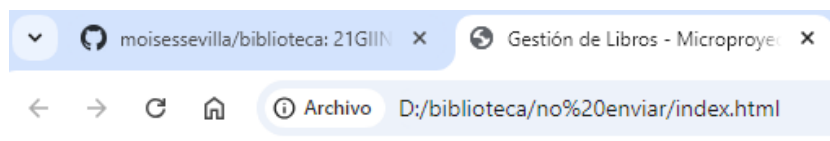
## 6. Perspectivas futuras

El sistema desarrollado es un excelente punto de partida para seguir ampliando funcionalidades, como:

- Gestión avanzada de usuarios, roles y permisos.
- Implementación de autenticación y autorización.
- Generación de reportes sobre préstamos y devoluciones.
- Integración de una interfaz gráfica para usuarios finales.

Esta actividad no solo cumplió con los objetivos planteados, sino que también proporcionó una experiencia práctica completa en el desarrollo de aplicaciones modulares con arquitectura de 3 capas, dejando un sistema funcional, eficiente y preparado para su ampliación futura.

Para concluir la actividad se muestra un escueto **index.html**, donde se da como ejemplo la conexión con la base de datos y una GUI muy simple para una ampliación futura:



## Gestión de Libros

Cargar Libros

### Agregar un nuevo libro

Título Autor Agregar

- Tras cargar los libros.

## Gestión de Libros

Cargar Libros

- 1 - El Quijote (Autor: Miguel de Cervantes)
- 2 - Cien Años de Soledad (Autor: Gabriel García Márquez)
- 3 - Rayuela - El retorno (Autor: Julio Cortázar)
- 4 - La Ciudad y los Perros (Autor: Mario Vargas Llosa)
- 5 - El Amor en los Tiempos del Cólera (Autor: Gabriel García Márquez)
- 6 - La Sombra del Viento (Autor: Carlos Ruiz Zafón)
- 7 - Pedro Páramo (Autor: Juan Rulfo)
- 8 - La Muerte de Artemio Cruz (Autor: Carlos Fuentes)
- 9 - El Túnel (Autor: Ernesto Sabato)
- 11 - Ficciones (Autor: Jorge Luis Borges)

### Agregar un nuevo libro

Título Autor Agregar

## **Bibliografía**

*Universidad Internacional de Valencia (VIU).*

*Libro Proyectos de programación, por Dr. Roger Clotet Martínez.*

*El código utilizado para la realización de esta actividad se encuentra en GitHub.*

<https://github.com/moisessevilla/biblioteca>