

# **Trabajo Práctico Evaluatorio**

## **Servidor ExpressJS**

### **Práctica Profesional**

(Documentacion de los pasos realizados en la actividad propuesta)

#### **1) Servidor web simple**

- Creamos un nuevo proyecto, configuramos npm e inicializamos el proyecto con npm

Esto creó un archivo package.json con la configuración básica del proyecto.

- Instalamos Express con el comando npm
- Creamos el archivo index.js
- Ejecutamos el servidor (node index.js)

En consola vemos el mensaje que dice Server running on port 3001 y al abrir el navegador y visitar <http://localhost:3001> vemos el mensaje "Hello World!".

#### **2) Express**

En este paso, configuramos Express para manejar más rutas y empezaremos a trabajar con datos en JSON.

- Middleware para JSON

Primero, actualizamos nuestro servidor para usar el middleware `express.json()`, que nos permitirá manejar datos en formato JSON en las solicitudes.

Se actualiza `index.js` para incluir el middleware.

- Datos de libros

Definimos datos iniciales de libros para trabajar. Agrega una variable `books` con una lista de libros (en formato json).

- Rutas para obtener todos los libros

Agregamos una nueva ruta para manejar solicitudes GET a `/api/books` y devolver la lista de libros en el navegador.

- Ejecutar el servidor

Ejecutamos el servidor de nuevo (node index.js)

Ahora visitamos <http://localhost:3001/api/books> en el navegador y vemos la lista de libros en formato JSON.

#### **3) Web y Express**

En este paso, mejoramos el servidor agregando una nueva ruta para devolver datos JSON y aprendimos a manejar solicitudes específicas usando parámetros de ruta.

- Obtener un libro por ID

Agregamos una nueva ruta para manejar solicitudes GET a /api/books/:id y devolver un libro específico según su ID.

- Manejo de rutas inexistentes

Agregamos un middleware para manejar rutas inexistentes y devolver un error 404.

El archivo index.js quedó de la siguiente manera:

```
// Middleware para manejar rutas desconocidas
const unknownEndpoint = (req, res) => {
  res.status(404).send({ error: 'unknown endpoint' })
}

app.use(unknownEndpoint)

app.listen(port, () => {
  console.log(`Server running on port ${port}`)
})
```

Ejecutamos el servidor y visitamos en el navegador <http://localhost:3001/api/books/1> (o cualquier otro ID de libro).

#### 4) Nodemon

- Instalamos nodemon

Instalamos nodemon como una dependencia de desarrollo (npm install --save-dev nodemon).

- Configuramos nodemon

Actualizamos el archivo package.json para usar nodemon cuando ejecutemos el servidor. Abrimos el package.json y buscamos la sección "scripts". Actualizamos el script de inicio para usar nodemon:

```
"scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js"
}
```

Ahora, en lugar de ejecutar node index.js cada vez, podemos usar npm run dev para iniciar el servidor con nodemon.

- Ejecutar el servidor con nodemon

Ejecutamos el servidor con nodemon (npm run dev)

Ahora nodemon se encargará de reiniciar automáticamente el servidor cada vez que hagamos cambios en el código.

- Próximo paso: REST

El siguiente paso es mejorar nuestro servidor para que siga los principios RESTful. Vamos a agregar más operaciones CRUD.

#### REST

- a. Crear un libro (POST)

Agregamos una nueva ruta para manejar solicitudes POST a /api/books y crear un nuevo libro.

- b. Eliminar un libro (DELETE)

Agregamos una nueva ruta para manejar solicitudes DELETE a `/api/books/:id` y eliminar un libro por su ID.

El código completo con estos cambios queda de la siguiente manera:

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.get('/api/books', (req, res) => {
  res.json(books)
})

app.get('/api/books/:id', (req, res) => {
  const id = Number(req.params.id)
  const book = books.find(book => book.id === id)
  if (book) {
    res.json(book)
  } else {
    res.status(404).end()
  }
})

app.post('/api/books', (req, res) => {
  const book = req.body

  if (!book.title || !book.author) {
    return res.status(400).json({ error: 'title or author missing' })
  }

  book.id = books.length + 1
  books.push(book)
  res.status(201).json(book)
})

app.delete('/api/books/:id', (req, res) => {
  const id = Number(req.params.id)
  books = books.filter(book => book.id !== id)
  res.status(204).end()
})

// Middleware para manejar rutas desconocidas
const unknownEndpoint = (req, res) => {
  res.status(404).send({ error: 'unknown endpoint' })
}

app.use(unknownEndpoint)

app.listen(port, () => {
  console.log(`Server running on port ${port}`)
})
```

## 5. Trabajando los recursos

- a. Obtener recurso. Ruta para obtener un libro por ID.

En este paso, nos aseguramos de que podamos obtener un solo recurso (en este caso, un libro) correctamente usando su ID.

Ya hemos creado la ruta para obtener un libro por ID en el paso anterior.

Esta ruta maneja solicitudes GET a `/api/books/:id`, donde `:id` es el ID del libro que queremos obtener. Si el libro existe, lo devolvemos en formato JSON. Si no existe, respondemos con un código de estado 404 (Not Found).

b. Eliminar recursos. Ruta para eliminar un libro por ID.

Agregamos una nueva ruta para manejar solicitudes DELETE a `/api/books/:id` y eliminar un libro por su ID.

Esta ruta elimina el libro cuyo ID coincida con el proporcionado en la solicitud. Utiliza el método `Array.prototype.filter()` para filtrar los libros y mantener solo aquellos cuyo ID sea diferente al ID que queremos eliminar. Finalmente, respondemos con un código de estado 204 (No Content) para indicar que la operación se ha realizado con éxito y no hay contenido para devolver.

El código completo quedó de la siguiente forma:

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.get('/api/books', (req, res) => {
  res.json(books)
})

app.get('/api/books/:id', (req, res) => {
  const id = Number(req.params.id)
  const book = books.find(book => book.id === id)
  if (book) {
    res.json(book)
  } else {
    res.status(404).end()
  }
})

app.post('/api/books', (req, res) => {
  const book = req.body

  if (!book.title || !book.author) {
    return res.status(400).json({ error: 'title or author missing' })
  }

  book.id = books.length + 1
  books.push(book)
  res.status(201).json(book)
})

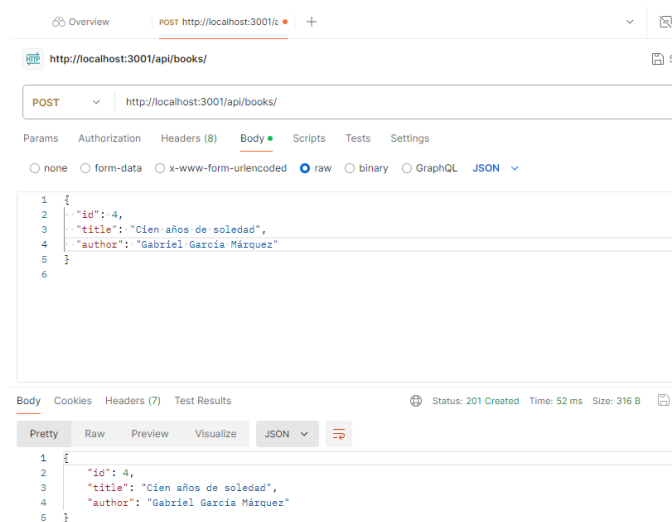
// Middleware para manejar rutas desconocidas
const unknownEndpoint = (req, res) => {
  res.status(404).send({ error: 'unknown endpoint' })
}

app.use(unknownEndpoint)

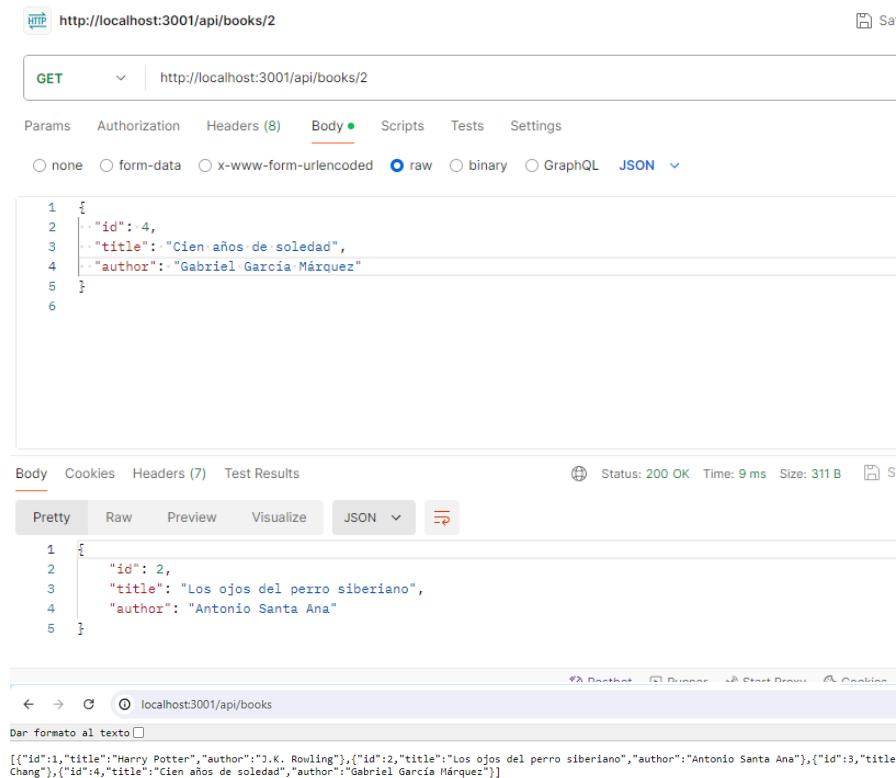
app.listen(port, () => {
  console.log(`Server running on port ${port}`)
})
```

## 6. Utilizamos Postman para verificar las solicitudes

a. Hacemos método Post en Postman para agregar libro



## b. Hacemos método get para obtener información de un libro



## c. Realizamos Put para cambiar información

HTTP PUT http://localhost:3001/api/books/1

Send

Params Authorization Headers (8) Body Scripts Tests Settings

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

```
1 {
2   "id": 5,
3   "title": "Harry potter y la camara secreta",
4   "author": "J.K. Rowling"
5 }
6
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 10 ms Size: 310 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "title": "Harry potter y la camara secreta",
4   "author": "J.K. Rowling"
5 }
```

localhost:3001/api/books

Dar formato al texto

```
[{"id":1,"title":"Harry potter y la camara secreta","author":"J.K. Rowling"}, {"id":2,"title":"Los ojos del perro siberiano","author":"Antonio Santa Ana"}, {"id":3,"title":"Química general","author":"Raymond Chang"}, {"id":4,"title":"Cien años de soledad","author":"Gabriel García Márquez"}]
```

## Validación de errores, intentamos ingresar un libro sin datos

HTTP POST http://localhost:3001/api/books/

Send

Params Authorization Headers (8) Body Scripts Tests Settings

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

```
1 {
2   "id": 5,
3   "title": "",
4   "author": ""
5 }
6
```

Body Cookies Headers (7) Test Results Status: 400 Bad Request Time: 11 ms Size: 279 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": "Title or author missing"
3 }
```

## 7. El cliente REST de Visual Studio Code

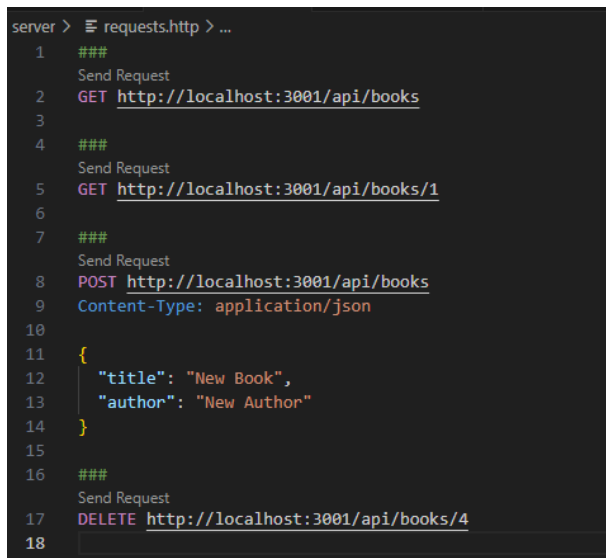
En este paso, aprendimos cómo usar el cliente REST integrado en Visual Studio Code para probar las rutas de nuestro servidor RESTful de libros.

- Instalar la extensión REST Client

Instalamos la extensión REST Client en Visual Studio Code, desde el Marketplace de VS Code.

- Crear y ejecutar solicitudes HTTP

Usamos un archivo con extensión .http para escribir y ejecutar nuestras solicitudes HTTP, es por eso que creamos un archivo llamado requests.http en la raíz de tu proyecto y agregamos las siguientes solicitudes:



```
server > requests.http > ...
1  ###
  Send Request
2  GET http://localhost:3001/api/books
3
4  ###
  Send Request
5  GET http://localhost:3001/api/books/1
6
7  ###
  Send Request
8  POST http://localhost:3001/api/books
9  Content-Type: application/json
10
11  {
12    "title": "New Book",
13    "author": "New Author"
14  }
15
16  ###
  Send Request
17  DELETE http://localhost:3001/api/books/4
18
```

En este archivo:

- Cada solicitud comienza con ### seguido de la solicitud HTTP y la URL.
- Para solicitudes POST, también especificamos el tipo de contenido y el cuerpo JSON del libro nuevo que queremos crear.
- Para solicitudes DELETE, especificamos la URL con el ID del libro que queremos eliminar.

### Ejecutar las solicitudes

Dentro de requests.http, se puede ver un botón "Send Request" junto a cada solicitud. Al hacer clic en este botón se envía la solicitud correspondiente al servidor.

### Verificar los resultados

Visual Studio Code muestra la respuesta de cada solicitud directamente en el editor. Podemos verificar que las operaciones CRUD funcionan correctamente para el servidor RESTful de libros.

## 8. Morgan

Ejecutamos en la terminal el comando correspondiente para instalarlo (npm install Morgan).

- Configuración de Morgan en la Aplicación Express

Configuramos Morgan para que registre mensajes en la consola según la configuración tiny y para mostrar los datos enviados en las solicitudes POST.

```
const express = require('express');
const morgan = require('morgan');
const app = express();
const port = 3001;
```

En nuestro código:

- Importación de Morgan:

`const morgan = require('morgan');`: Importa la biblioteca Morgan para registrar mensajes de solicitud en la consola.

- Configuración de Morgan:

`app.use(morgan('tiny'));`: Configura Morgan para registrar mensajes en la consola utilizando el formato tiny. Este middleware se coloca al inicio para que registre todas las solicitudes HTTP entrantes.

- Funcionalidad Existente:

Se mantienen todas las rutas y la lógica existente para manejar las solicitudes GET, POST, PUT y DELETE de los libros.

- Middleware para manejar rutas desconocidas:

`const unknownEndpoint = ...`: Se define un middleware para manejar cualquier ruta que no esté definida explícitamente. Esto devuelve un código de estado 404 y un mensaje de error cuando se accede a una ruta desconocida.

- Inicio del Servidor:

`app.listen(port, ...)`: Inicia el servidor en el puerto especificado (port) y muestra un mensaje en la consola cuando el servidor se inicia correctamente.

El código quedo de la siguiente forma:



```
const express = require('express');
const morgan = require('morgan');

const app = express();
const port = 3001;

// Middleware para registrar mensajes en la consola según configuración 'tiny'
app.use(morgan('tiny'));

// Middleware para analizar solicitudes JSON
app.use(express.json());

let books = [
  { id: 1, title: 'Harry Potter', author: 'J.K. Rowling' },
  { id: 2, title: 'Los ojos del perro siberiano', author: 'Antonio Santa Ana' },
  { id: 3, title: 'Química general', author: 'Raymond Chang' }
];

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.get('/api/books', (req, res) => {
  res.json(books);
});

app.get('/api/books/:id', (req, res) => {
  const id = Number(req.params.id);
  const book = books.find(book => book.id === id);
  if (book) {
    res.json(book);
  } else {
    res.status(404).end();
  }
});

app.post('/api/books', (req, res) => {
  const book = req.body;

  if (!book.title || !book.author) {
    return res.status(400).json({ error: 'Title or author missing' });
  }

  book.id = books.length + 1;
  books.push(book);
  res.status(201).json(book);
});
```

```
app.put('/api/books/:id', (req, res) => {
  const id = Number(req.params.id);
  const { title, author } = req.body;

  const bookIndex = books.findIndex(book => book.id === id);

  if (bookIndex !== -1) {
    if (title) books[bookIndex].title = title;
    if (author) books[bookIndex].author = author;
    res.json(books[bookIndex]);
  } else {
    res.status(404).json({ error: 'Book not found' });
  }
});

app.delete('/api/books/:id', (req, res) => {
  const id = Number(req.params.id);
  books = books.filter(book => book.id !== id);
  res.status(204).end();
});

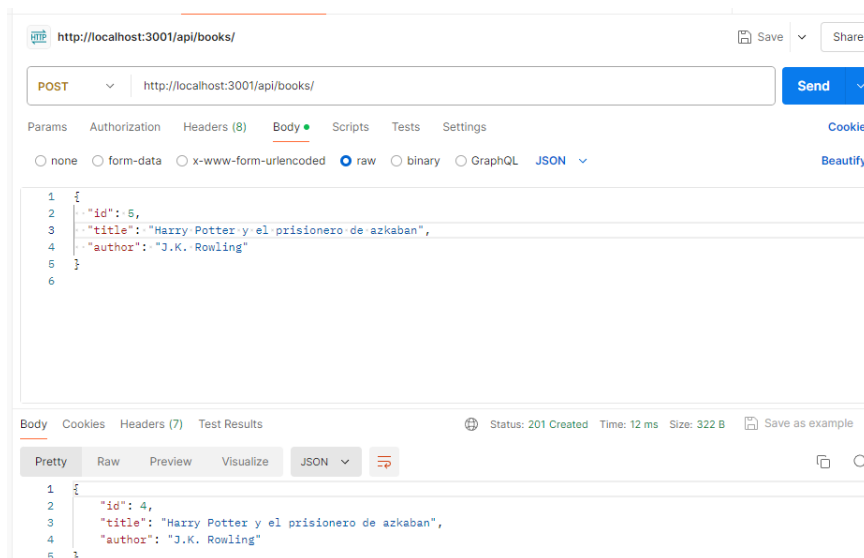
// Middleware para manejar rutas desconocidas
const unknownEndpoint = (req, res) => {
  res.status(404).send({ error: 'Unknown endpoint' });
};

app.use(unknownEndpoint);

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

Una vez que hicimos estos cambios, guardamos el archivo y lo ejecutamos usando node index.js en la terminal.

Luego de integrar Morgan probamos en postman el método get:



## 9. Se agrega usuarios, prestamos y autor.

Agregamos la funcionalidad para manejar autores, usuarios y préstamos teniendo la estructura básica con Express y Node.js que implementamos para los libros. Siguiendo la misma lógica, creamos rutas adicionales para manejar estas entidades.

- Definimos las Nuevas Entidades

Vamos a definir las nuevas entidades authors (autores), users (usuarios) y loans (prestamos) en nuestro archivo. Esto incluye crear arrays de ejemplo y añadir las rutas correspondientes.

- Crear las Rutas CRUD para las Nuevas Entidades

Para cada nueva entidad, agregamos las rutas necesarias para realizar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

### 1. Rutas para Autores:

- **GET /api/authors:** Devuelve todos los autores.
- **GET /api/authors/:id:** Devuelve un autor por ID.
- **POST /api/authors:** Crea un nuevo autor.
- **PUT /api/authors/:id:** Actualiza un autor por ID.
- **DELETE /api/authors/:id:** Elimina un autor por ID.

### 2. Rutas para Usuarios:

- **GET /api/users:** Devuelve todos los usuarios.
- **GET /api/users/:id:** Devuelve un usuario por ID.
- **POST /api/users:** Crea un nuevo usuario.
- **PUT /api/users/:id:** Actualiza un usuario por ID.
- **DELETE /api/users/:id:** Elimina un usuario por ID.

### 3. Rutas para Préstamos:

- **GET /api/loans:** Devuelve todos los préstamos.
- **GET /api/loans/:id:** Devuelve un préstamo por ID.
- **POST /api/loans:** Crea un nuevo préstamo.
- **PUT /api/loans/:id:** Actualiza un préstamo por ID.
- **DELETE /api/loans/:id:** Elimina un préstamo por ID.