

Clonamos el repo en:

/home/vagrant

git clone <https://github.com/omondragon/APIRestFlask>

cd APIRestFlask

ls

```
[[vagrant@servidor2 ~]$ ls
APIRestFlask  pythonFlaskExample1  pythonFlaskMysql  swagger-example  testFlask
[[vagrant@servidor2 ~]$ cd APIRestFlask
[[vagrant@servidor2 APIRestFlask]$ ls
apiREST.py
[[vagrant@servidor2 APIRestFlask]$
```

vim apiREST.py

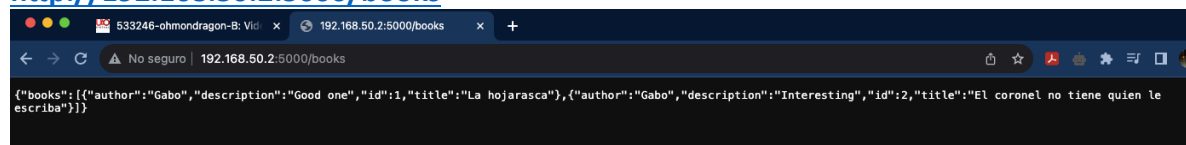
EJECUTAR EL CODIGO

```
export FLASK_APP=apiREST.py
export FLASK_ENV=development
python3 -m flask run --host=0.0.0.0
```

```
[[vagrant@servidor2 APIRestFlask]$ vim apiREST.py
[[vagrant@servidor2 APIRestFlask]$ export FLASK_APP=apiREST.py
[[vagrant@servidor2 APIRestFlask]$ export FLASK_ENV=development
[[vagrant@servidor2 APIRestFlask]$ python3 -m flask run --host=0.0.0.0
* Serving Flask app 'apiREST.py'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.0.2.15:5000
Press CTRL+C to quit
192.168.50.1 - - [24/Oct/2023 16:55:41] "GET / HTTP/1.1" 404 -
192.168.50.1 - - [24/Oct/2023 16:55:58] "GET /books HTTP/1.1" 200 -
192.168.50.1 - - [24/Oct/2023 16:55:59] "GET /favicon.ico HTTP/1.1" 404 -
```

Hacemos prueba con:

<http://192.168.50.2:5000/books>



```
{
  "books": [
    {
      "author": "Gabo",
      "description": "Good one",
      "id": 1,
      "title": "La hojarasca"
    },
    {
      "author": "Gabo",
      "description": "Interesting",
      "id": 2,
      "title": "El coronel no tiene quien le escriba"
    }
  ]
}
```

Hacemos prueba con:

curl -i http://192.168.50.2:5000/books

```
[juanmartinvasquezcaicedo@JUANS-MacBook-Air-5 vagrant % vagrant ssh servidor1
Last login: Tue Oct 24 16:19:01 2023 from 10.0.2.2
[[vagrant@servidor1 ~]$ curl -i http://192.168.50.2:5000/books
HTTP/1.1 200 OK
Server: Werkzeug/3.0.0 Python/3.9.17
Date: Tue, 24 Oct 2023 16:58:07 GMT
Content-Type: application/json
Content-Length: 185
Connection: close

{"books":[{"author":"Gabo","description":"Good one","id":1,"title":"La hojarasca"}, {"author":"Gabo","description":"Interesting","id":2,"title":"El coronel no tiene quien le escriba"}]}
[vagrant@servidor1 ~]$
```

curl -i <http://192.168.50.2:5000/books/2>

```
[[vagrant@servidor1 ~]$ curl -i http://192.168.50.2:5000/books/2
HTTP/1.1 200 OK
Server: Werkzeug/3.0.0 Python/3.9.17
Date: Tue, 24 Oct 2023 19:04:36 GMT
Content-Type: application/json
Content-Length: 109
Connection: close

{"book":{"author":"Gabo","description":"Interesting","id":2,"title":"El coronel no tiene quien le escriba"}}
[vagrant@servidor1 ~]$
```

PROBAR LIBRO NO EXISTENTE

curl -i <http://192.168.50.2:5000/books/3>

```
[[vagrant@servidor1 ~]$ curl -i http://192.168.50.2:5000/books/3
HTTP/1.1 404 NOT FOUND
Server: Werkzeug/3.0.0 Python/3.9.17
Date: Tue, 24 Oct 2023 19:06:39 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Connection: close

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>
[vagrant@servidor1 ~]$
```

POSTMAN

Instalamos postman

Creamos una coleccion:

API REST Flask

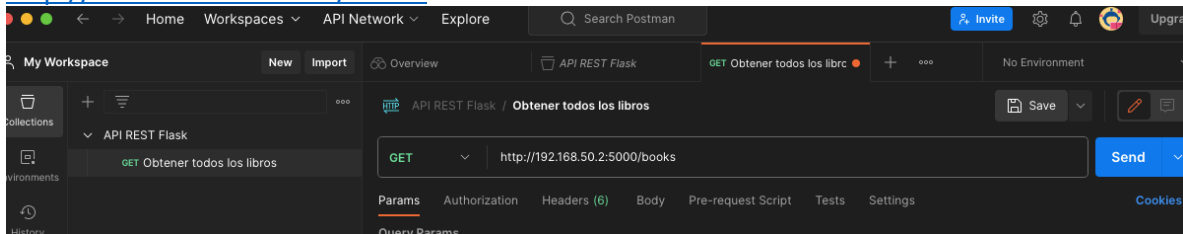
Agregamos solicitudes a la coleccion:

Obtener todos los libros

Nos aseguramos que el método sea GET

Colocamos la URL de la API

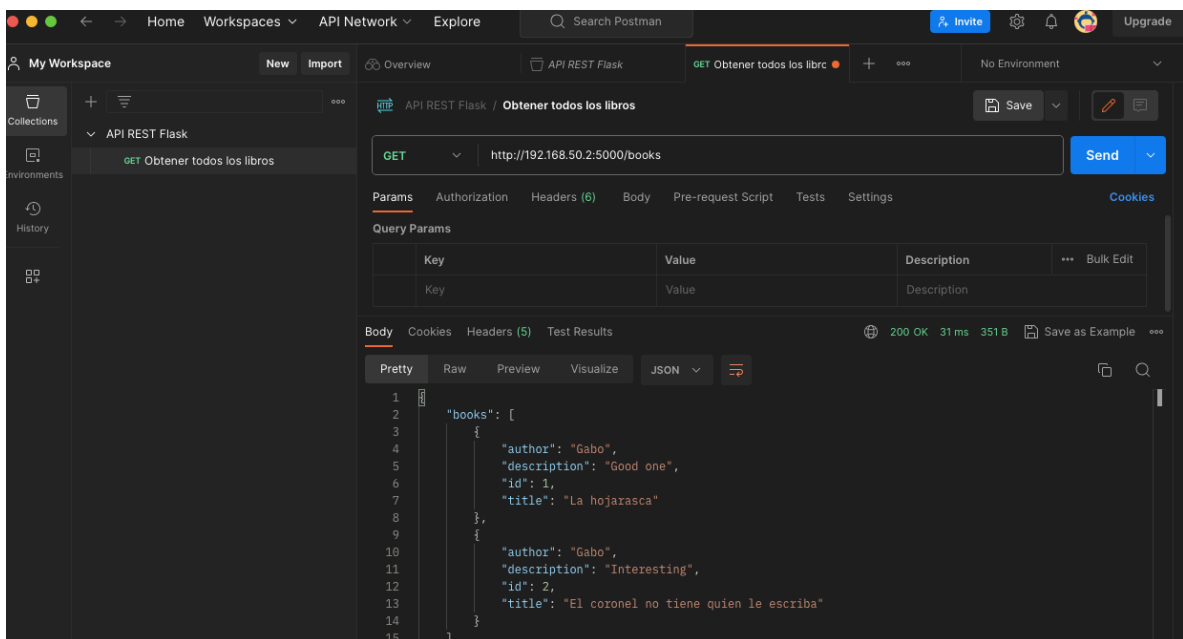
<http://192.168.50.2:5000/books>



Enviar la Solicitud

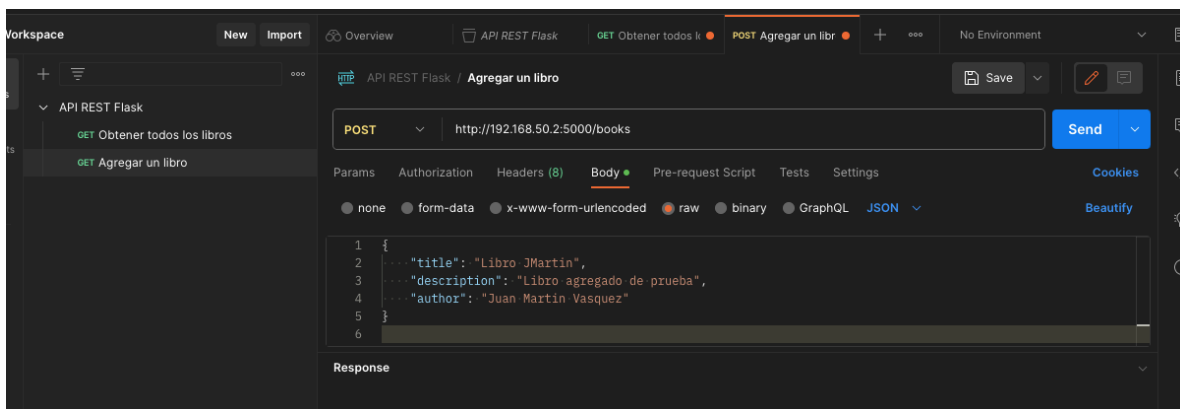
Hacemos clic en el botón "Send" (Enviar) para realizar la solicitud GET.

Vemos la respuesta de la API en la parte inferior de la ventana de Postman. Esta respuesta mostrará los libros disponibles en la API.

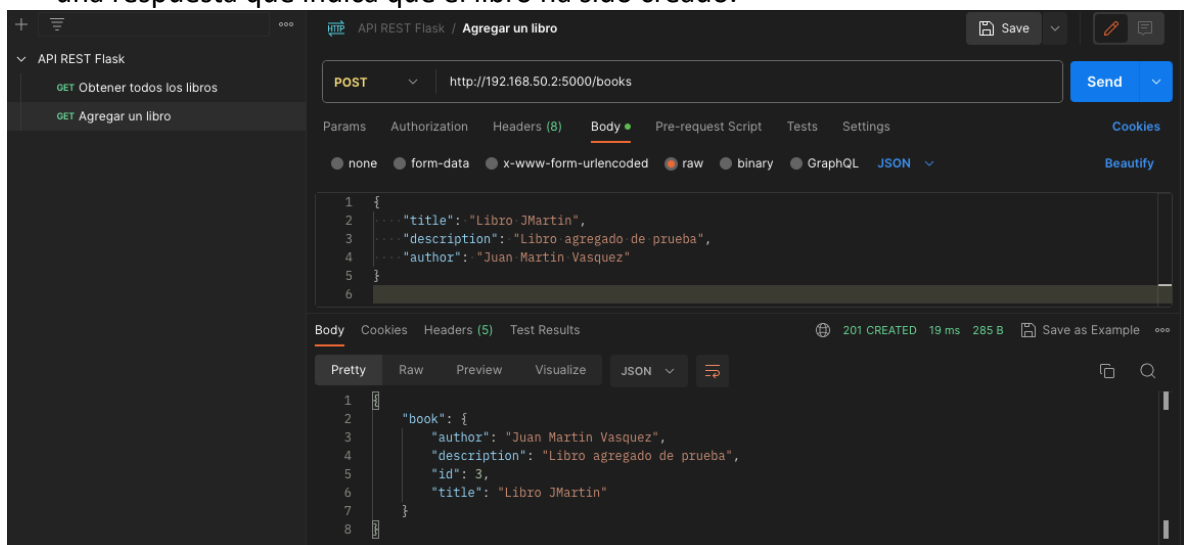


Crear una solicitud POST para agregar un libro:

- Nos ubicamos en la colección "API REST Flask" en Postman.
- Hacemos clic en "Add a request" y asignamos un nombre a la nueva solicitud, por ejemplo, "Agregar un libro".
- En el panel izquierdo de la solicitud, seleccionamos el método "POST".
- En la barra de URL, ingresa la URL de la API para agregar un libro:
<http://192.168.50.2:5000/books>
- En la sección "Body" (Cuerpo), seleccionamos "raw" (en crudo) y elegimos "JSON (application/json)" en el menú desplegable.
- Ingresamos los datos del libro en formato JSON. Por ejemplo:

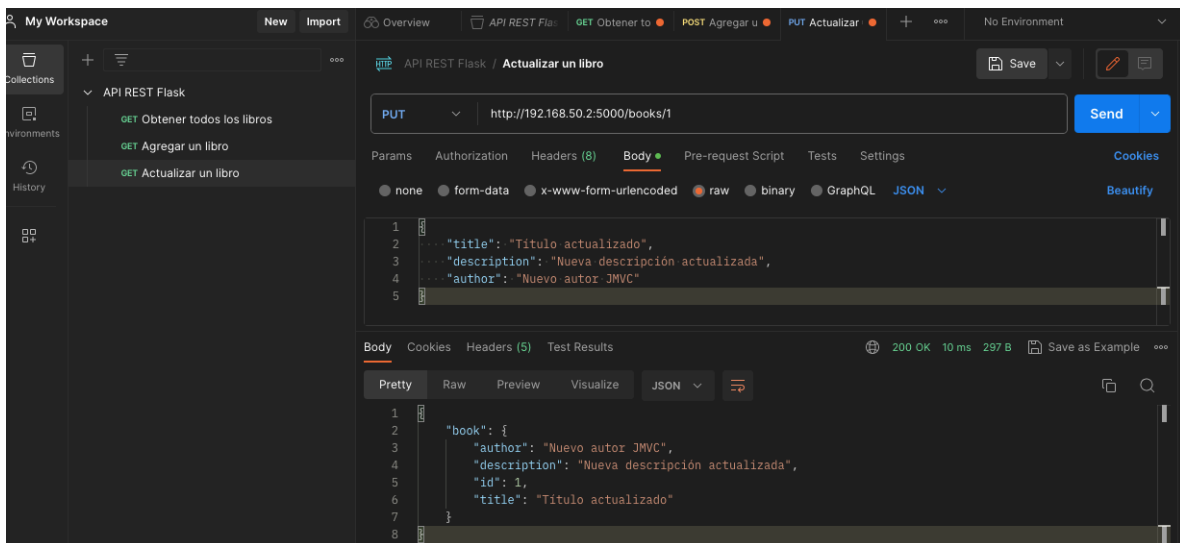


- Luego, hacemos clic en "Send" (Enviar) para agregar el libro. Deberíamos recibir una respuesta que indica que el libro ha sido creado.



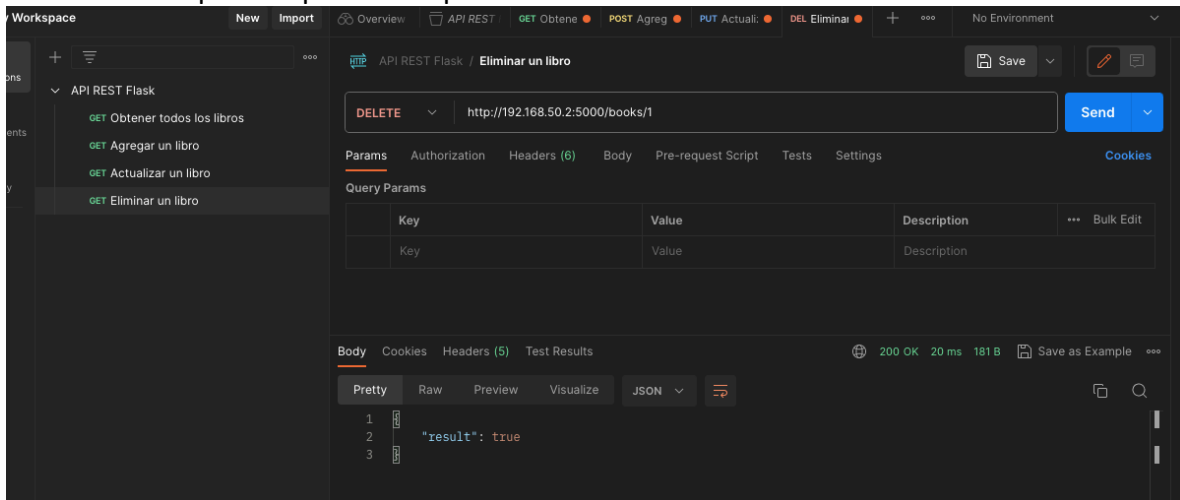
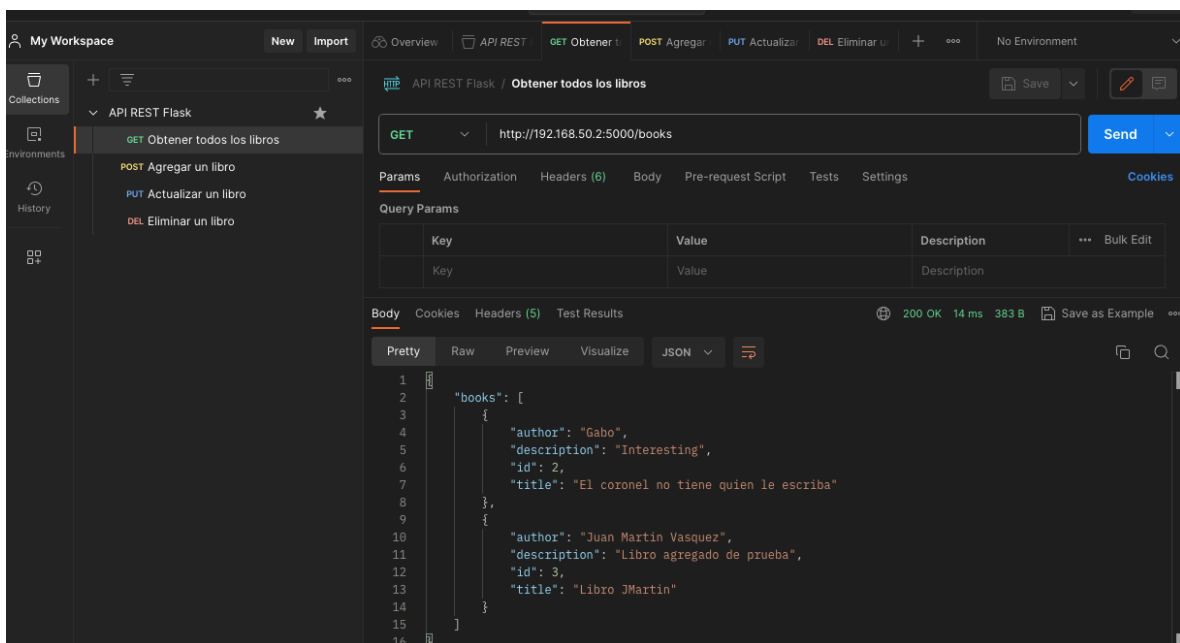
Crear una solicitud PUT para actualizar un libro:

- Creamos una nueva solicitud en la colección "API REST Flask" y la nombramos, por ejemplo, "Actualizar un libro".
- Seleccionamos el método "PUT" en el panel izquierdo de la solicitud.
- En la barra de URL, ingresa la URL de la API para actualizar un libro, por ejemplo: <http://192.168.50.2:5000/books/1> (donde "1" es el ID del libro que deseamos actualizar).
- En la sección "Body" (Cuerpo), seleccionamos "raw" y elegimos "JSON (application/json)".
- Ingresa los datos actualizados del libro en formato JSON. Por ejemplo:



Crear una solicitud DELETE para eliminar un libro:

- Creamos una nueva solicitud en la colección "API REST Flask" y la nombramos, por ejemplo, "Eliminar un libro".
- Seleccionamos el método "DELETE" en el panel izquierdo de la solicitud.
- En la barra de URL, ingresamos la URL de la API para eliminar un libro, por ejemplo: <http://192.168.50.2:5000/books/1> (donde "1" es el ID del libro que deseamos eliminar).
- Luego, hacemos clic en "Send" (Enviar) para eliminar el libro. Deberíamos recibir una respuesta que indica que el libro ha sido eliminado.

**GUARDAMOS LA COLECCIÓN**

AHORA CON Node.js

Instalamos Node.js y npm:

```
sudo dnf install nodejs
```

```
sudo dnf install npm
```

Instalamos Express y otras dependencias necesarias:

```
npm install express --save
```

```
npm install body-parser --save
```

Creamos el directorio del proyecto:

```
mkdir my-rest-api
```

```
cd my-rest-api
```

Creamos el archivo:

```
vim app.js
```

DENTRO DEL ARCHIVO:

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

```
const bodyParser = require('body-parser');
```

```
const app = express();
```

```
const port = 3000; // Puedes cambiar el puerto si lo deseas
```

```
app.use(bodyParser.json());
```

```
// Datos de ejemplo (una colección de libros)
```

```
let books = [
```

```
{
```

```
  id: 1,
```

```
  title: 'Libro 1',
```

```
  author: 'Autor 1',
```

```
},
```

```
{
```

```
  id: 2,
```

```
  title: 'Libro 2',
```

```
  author: 'Autor 2',
```

```
},
```

```
];
```

```
// Ruta para obtener todos los libros
```

```
app.get('/books', (req, res) => {
```

```
  res.json(books);
```

```
});
```

```
// Ruta para obtener un libro por su ID
app.get('/books/:id', (req, res) => {
  const bookId = parseInt(req.params.id);
  const book = books.find((book) => book.id === bookId);
  if (book) {
    res.json(book);
  } else {
    res.status(404).send('Libro no encontrado');
  }
});
```

```
// Ruta para agregar un nuevo libro
app.post('/books', (req, res) => {
  const newBook = req.body;
  newBook.id = books.length + 1;
  books.push(newBook);
  res.status(201).json(newBook);
});
```

```
// Ruta para actualizar un libro por su ID
app.put('/books/:id', (req, res) => {
  const bookId = parseInt(req.params.id);
  const book = books.find((book) => book.id === bookId);
  if (book) {
    Object.assign(book, req.body);
    res.json(book);
  } else {
    res.status(404).send('Libro no encontrado');
  }
});
```

```
// Ruta para eliminar un libro por su ID
app.delete('/books/:id', (req, res) => {
  const bookId = parseInt(req.params.id);
  const index = books.findIndex((book) => book.id === bookId);
  if (index !== -1) {
    books.splice(index, 1);
    res.status(204).send('Libro eliminado');
  } else {
    res.status(404).send('Libro no encontrado');
  }
});
```



```
app.listen(port, () => {  
  console.log(`Servidor escuchando en http://localhost:${port}`);  
});
```

PARA INCIAR EL SERVICIO:

```
node app.js
```

<http://192.168.50.2:3000/books>

POSTMAN**Obtener todos los libros**

The screenshot shows the Postman interface with a GET request to `http://192.168.50.2:3000/books`. The response is a 200 OK status with a body containing a JSON array of two books.

Key	Value	Description
Key	Value	Description

```
1 {  
2   "id": 1,  
3   "title": "Libro 1",  
4   "author": "Autor 1"  
5 },  
6 {  
7   "id": 2,  
8   "title": "Libro 2",  
9   "author": "Autor 2"  
10 }  
11 }  
12 }
```

Obtener un libro por ID

The screenshot shows the Postman interface with a workspace named 'My Workspace'. On the left, a collection 'API de Libros' is expanded, showing a list of requests: 'GET Obtener todos los libros', 'GET Obtener un libro por ID' (selected), 'POST Agregar un libro', 'PUT Actualizar un libro', and 'DEL Eliminar un libro'. The main panel displays the details of the selected 'GET Obtener un libro por ID' request. The URL is 'http://192.168.50.2:3000/books/1'. The response status is '200 OK' with a response time of '19 ms' and a body size of '280 B'. The response body is shown in JSON format:

```
{  "id": 1,  "title": "Libro 1",  "author": "Autor 1"}
```

Agregar un libro

The screenshot shows the Postman interface with the same workspace. The collection 'API de Libros' is expanded, and the 'GET Agregar un libro' request is selected. The main panel displays the details of the selected 'POST Agregar un libro' request. The URL is 'http://192.168.50.2:3000/books'. The request body is shown in JSON format:

```
{  "title": "Nuevo Libro",  "author": "Nuevo Autor"}
```

 The response status is '201 Created' with a response time of '42 ms' and a body size of '293 B'. The response body is shown in JSON format:

```
{  "title": "Nuevo Libro",  "author": "Nuevo Autor",  "id": 3}
```

Actualizar libro

The screenshot shows the Postman interface with a workspace named "My Workspace". The left sidebar displays a collection named "API REST Flask" containing several API endpoints. The selected endpoint is "PUT Actualizar un libro". The main panel shows the request configuration for a PUT request to the URL "http://192.168.50.2:3000/books/1". The request body is set to JSON and contains the following data:

```
{  "title": "Libro Actualizado",  "author": "Autor Actualizado"}
```

The response section shows a status of 200 OK, with a response time of 18 ms and a body size of 300 B. The response body is displayed in JSON format:

```
{  "id": 1,  "title": "Libro Actualizado",  "author": "Autor Actualizado"}
```

Eliminar un libro

The screenshot shows the Postman interface with a workspace named "My Workspace". The left sidebar displays a collection named "API REST Flask" containing several API endpoints. The selected endpoint is "DEL Eliminar un libro". The main panel shows the request configuration for a DELETE request to the URL "http://192.168.50.2:3000/books/1". The request body is empty. The response section shows a status of 204 No Content, with a response time of 16 ms and a body size of 175 B. The response body is displayed in Text format:

```
1
```