Proyecto

"APIs RESTful"

Aplicaciones Web

Albán Lucas Dorian Joel

Escuela Politécnica Nacional

11 de noviembre de 2025

Tabla de contenido

1. Objetivos del Proyecto

El desarrollo de esta API RESTful cumplió con los siguientes propósitos principales:

1. **Diseñar Endpoints RESTful:** Diseñar rutas y métodos que adhieran estrictamente a los estándares y convenciones REST de la industria.
2. **Modelar Relaciones:** Demostrar la implementación de una relación **Uno a Muchos (1:N)** entre entidades (Library y Book) mediante una Foreign Key (libraryId).
3. **Documentación Estándar:** Documentar la API por completo utilizando el estándar **OpenAPI 3.0 (Swagger)**.
4. **Creación de Colección de Pruebas:** Establecer una colección de pruebas verificables utilizando la herramienta **Bruno** para garantizar la correcta implementación de todas las operaciones CRUD.

2. Fundamentos Teóricos y Diseño

2.1. El Estándar Arquitectónico RESTful

REST (Representational State Transfer) es un estilo arquitectónico que guía el diseño de servicios web. Se basa en el concepto de **recursos** (identificados por URLs únicas) y la manipulación de dichos recursos mediante los métodos estándar de HTTP.

Principios Clave Aplicados:

- **Uso de Verbos HTTP:** Utilizar GET, POST, PUT, DELETE para las operaciones CRUD.
- **Recursos Sin Estado (Stateless):** Cada solicitud contiene toda la información necesaria para ser procesada.
- **Identificación Uniforme de Recursos:** URLs claras, predictibles y que utilizan sustantivos en plural (ej. /libraries).

2.2. Modelado de Datos: Relación Uno a Muchos (1:N)

El proyecto modela la relación donde **una Biblioteca puede tener múltiples Libros**, y **cada Libro pertenece a una única Biblioteca**.

- **Entidad Uno (1):** Biblioteca (/libraries)
- **Entidad Muchos (N):** Libro (/books)
- **Vínculo:** El campo libraryId en la entidad Book actúa como Foreign Key, referenciando a la Biblioteca a la que pertenece.

2.3. Endpoints Diseñados

Los *endpoints* se diseñaron para ofrecer operaciones CRUD completas, incluyendo una ruta anidada para explorar la relación 1:N.

Entidad	Método	Endpoint	Descripción
Biblioteca	GET	/libraries	Obtiene la lista de todas las bibliotecas.
Biblioteca	POST	/libraries	Crea una nueva biblioteca (respuesta 201 Created).
Libro	GET	/books	Obtiene la lista de todos los libros.
Libro	POST	/books	Crea un nuevo libro, requiere libraryId en el cuerpo.
Relación	GET	/libraries/{id}/books	**Endpoint Anidado:** Obtiene todos los libros asociados a una biblioteca específica.
CRUD Genérico	PUT/DELETE	/libraries/{id}	Actualización/Eliminación de bibliotecas.
CRUD Genérico	PUT/DELETE	/books/{id}	Actualización/Eliminación de libros.

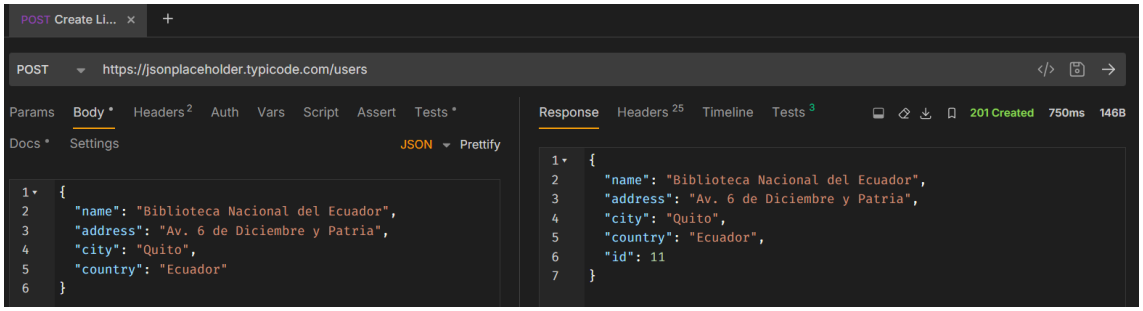
3. Evidencia Práctica de Métodos CRUD

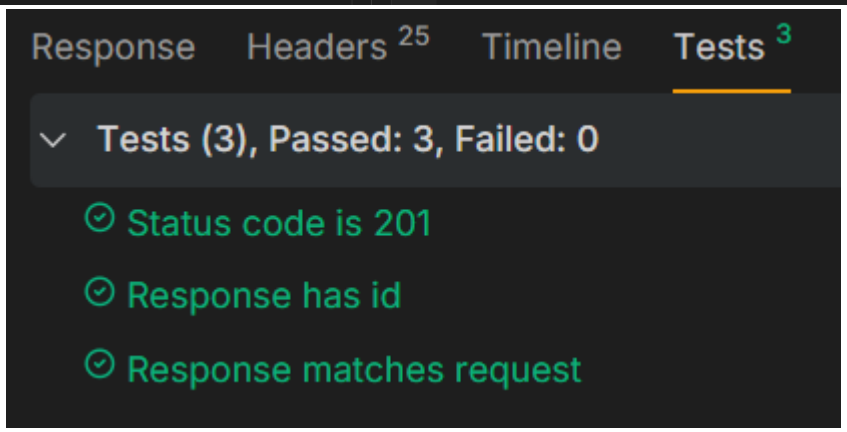
Esta sección contiene las evidencias visuales de la ejecución de los métodos HTTP contra la API, verificando los códigos de estado y la estructura de los datos (schemas) documentados en **swagger.yaml** y probados en la colección **bruno/**.

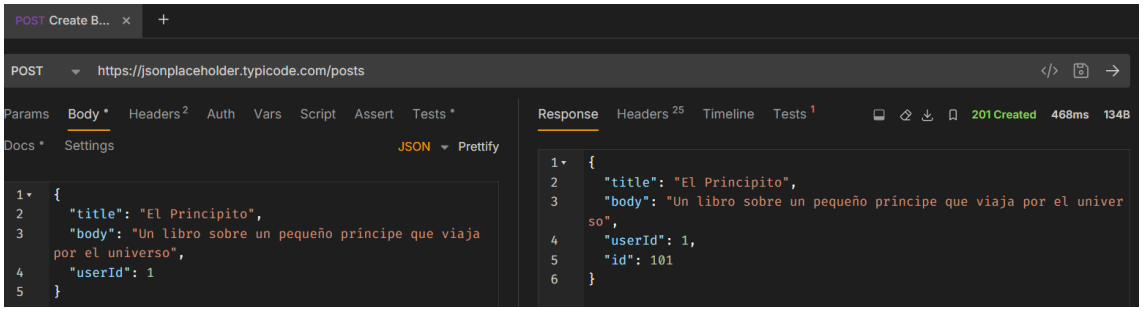
3.1. Creación de Recursos (POST)

Se verifica que la creación de un recurso resulte en un código de estado 201 Created y devuelva el objeto con el ID generado.

Descripción
POST /libraries (Creación de Biblioteca)
POST /books (Creación de Libro)



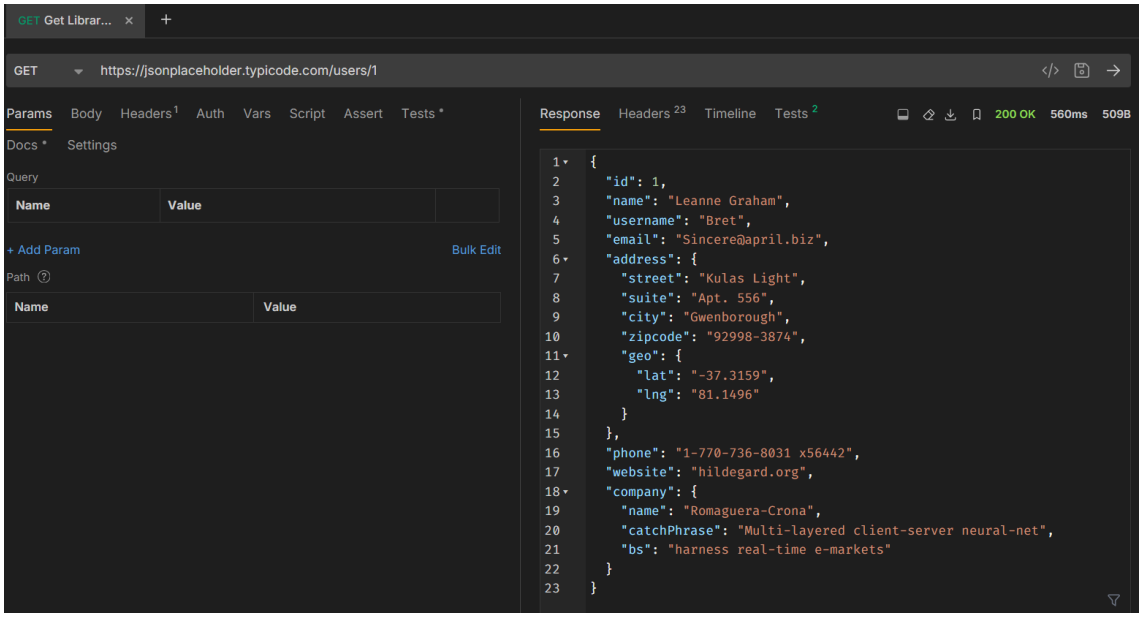


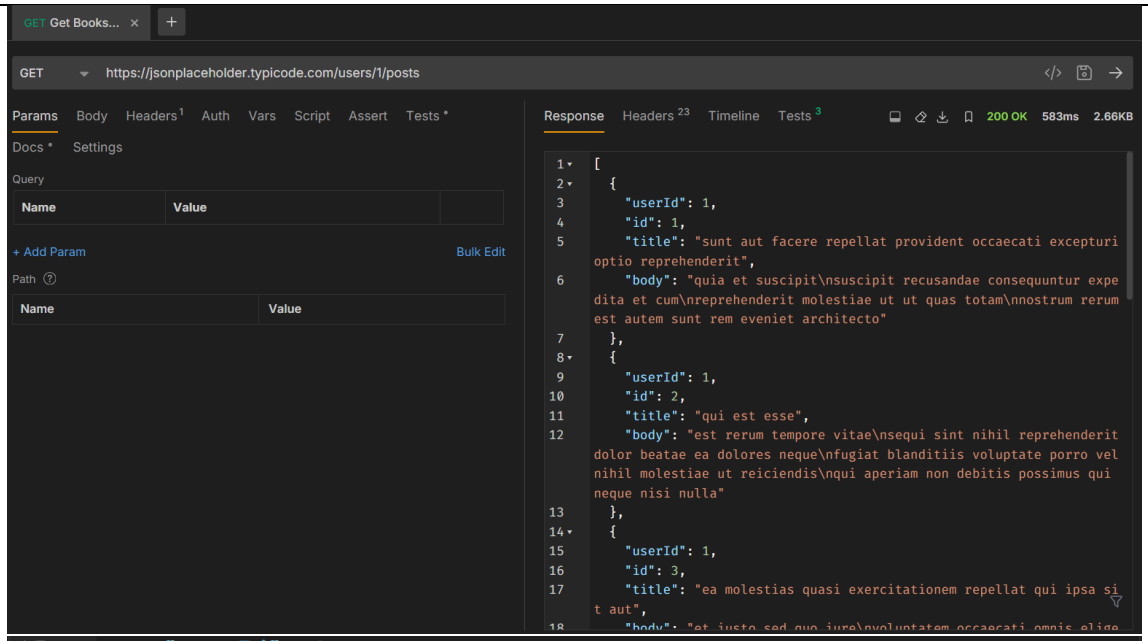


3.2. Lectura y Relación (GET)

Se comprueba la obtención de recursos individuales y la funcionalidad de la relación 1:N.

Descripción
GET /libraries/{id} (Obtención por ID)
GET /libraries/{id}/books (Relación 1:N)





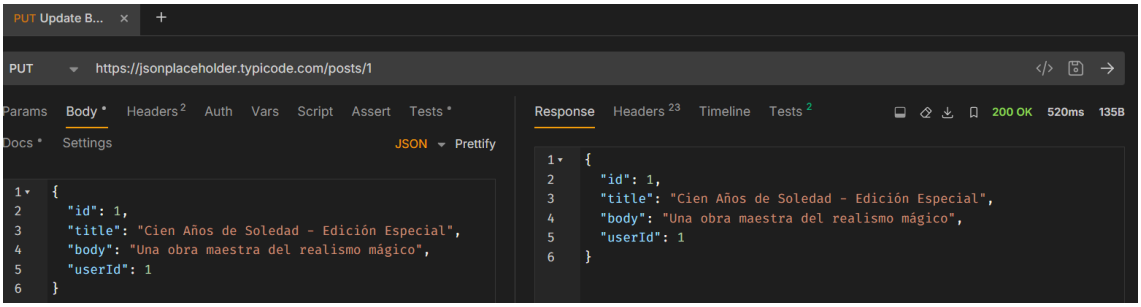

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
    "body": "et iusto sed quo iure\nvoluptatem occaecati omnis elige
```

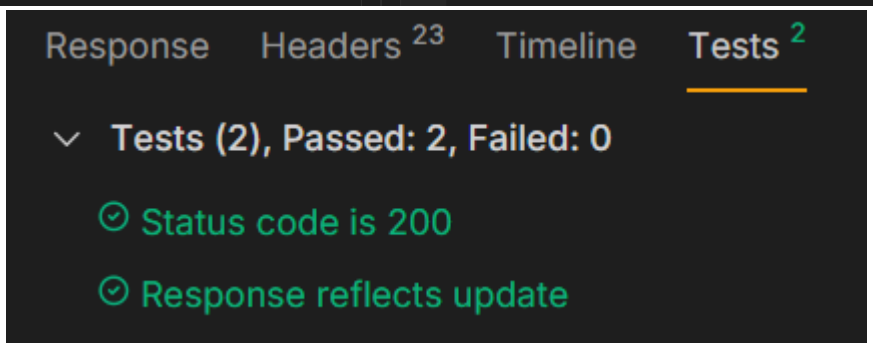
```
    "userId": 1,
    "id": 8,
    "title": "dolorem dolore est ipsam",
    "body": "dignissimos aperiam dolorem qui eum\nfacilis quibusdam animi sint suscipit qui sint possimus cum\nquaerat magni maiores excepturi\nipsam ut commodi dolor voluptatum modi aut vitae"
  },
  {
    "userId": 1,
    "id": 9,
    "title": "nesciunt iure omnis dolorem tempora et accusantium",
    "body": "consectetur animi nesciunt iure dolore\nenim quia ad\nveniam autem ut quam aut nobis\net est aut quod aut provident voluptas autem voluptas"
  },
  {
    "userId": 1,
    "id": 10,
    "title": "optio molestias id quia eum",
    "body": "quo et expedita modi cum officia vel magni\ndoloribus qui repudiandae\nvero nisi sit\nquos veniam quod sed accusamus veritatis error"
  }
```

3.3. Actualización de Recursos (PUT)

Se utiliza el método PUT para demostrar la actualización de un recurso por ID, esperando un código 200 OK.

Descripción
PUT /books/{id}

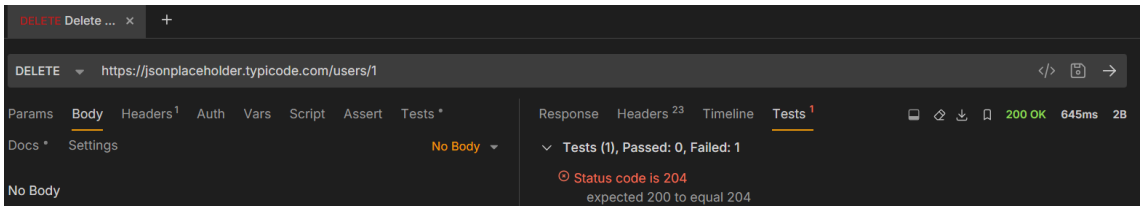




3.4. Eliminación de Recursos (DELETE)

Se verifica la eliminación exitosa de un recurso, esperando una respuesta sin contenido (204 No Content)

Descripción
DELETE /libraries/{id}



4. Conclusiones del Proyecto RESTful

El desarrollo de esta API de Biblioteca y Libros permitió aplicar y consolidar los principios del diseño REST y la documentación estándar de la industria.

1. **Adherencia RESTful:** Se demostró la capacidad de diseñar *endpoints* utilizando los sustantivos en plural para los recursos y los métodos HTTP de manera semánticamente correcta (ej. /books para colecciones, POST para crear, DELETE para eliminar).
2. **Modelado de la Relación 1:N:** La implementación del *Foreign Key* (libraryId) en el recurso Book y la inclusión de la ruta anidada (/libraries/{id}/books) confirman un diseño adecuado para modelar relaciones uno a muchos en un entorno REST.
3. **Valor de la Documentación (Swagger):** La generación del archivo **swagger.yaml** asegura que la API tenga un contrato formal y verificable. Esta documentación es esencial para el equipo de desarrollo, reduciendo la ambigüedad y facilitando la integración.
4. **Flujo de Trabajo Integrado:** El proyecto valida un flujo de trabajo profesional donde el diseño (README.md), la documentación (swagger.yaml) y el *testing* (bruno/) se mantienen sincronizados, garantizando la calidad y consistencia del servicio.