Escuela Politécnica Nacional
Facultad De Ingeniería De Sistemas
Aplicaciones Web
Periodo Académico 2025B

Proyecto
APIs RESTful

Aplicaciones Web

Christian Aragón

12/12/2025

Contenidos

1.OBJETIVOS DEL PROYECTO

2.FUNDAMENTOS TEÓRICOS Y DISEÑO

2.1. El Estándar Arquitectónico RESTful

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

2.3. Endpoints Diseñados

3.IMPLEMENTACIÓN Y DOCUMENTACIÓN

4.EVIDENCIA PRÁCTICA DE MÉTODOS CRUD

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, /books).
- Representación JSON: Todas las respuestas y peticiones utilizan el formato JSON para el intercambio de datos.
- Códigos de Estado HTTP Semánticos: Uso correcto de códigos como 200 OK, 201 Created, 204 No Content, 404 Not Found, etc.

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.

Esquema de Datos

Biblioteca (Library)

json

```
{
 "id": 1,
 "name": "Biblioteca Nacional",
 "address": "Av. Principal 123",
 "city": "Quito",
 "phone": "+593-2-1234567"
}
```

Libro (Book)

json

```
{
 "id": 1,
 "title": "Cien años de soledad",
 "author": "Gabriel García Márquez",
 "isbn": "978-0307474728",
 "genre": "Ficción",
 "publishYear": 1967,
 "libraryId": 1
}
```

El campo `libraryId` establece la relación de pertenencia, asegurando la integridad referencial entre ambas entidades.

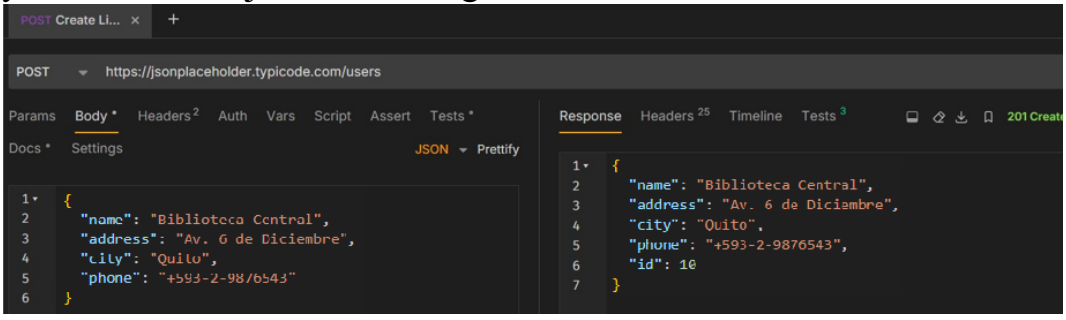
2.3. Endpoints Diseñados

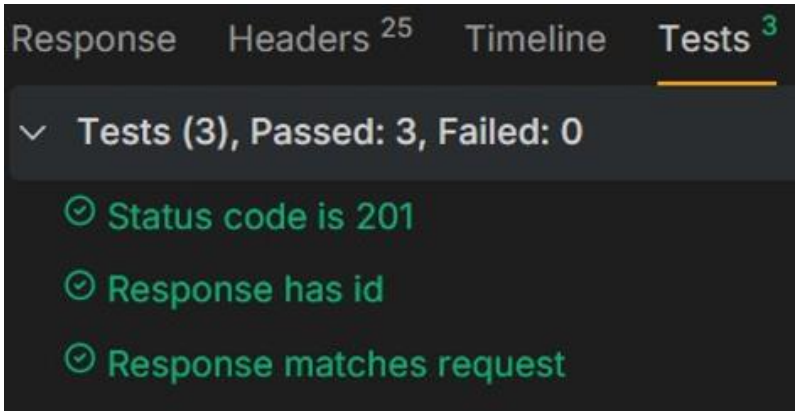
Entidad	Método	Endpoint	Descripción
Biblioteca	GET	/libraries	Obtiene la lista de todas las bibliotecas.
Biblioteca	GET	/libraries/{id}	Obtiene una biblioteca específica por ID.
Biblioteca	POST	/libraries	Crea una nueva biblioteca (respuesta 201 Created).
Biblioteca	PUT	/libraries/{id}	Actualiza completamente una biblioteca existente.
Biblioteca	DELETE	/libraries/{id}	Elimina una biblioteca por ID (respuesta 204).
Libro	GET	/books	Obtiene la lista de todos los libros.
Libro	GET	/books/{id}	Obtiene un libro específico por ID.
Libro	POST	/books	Crea un nuevo libro, requiere libraryId en el cuerpo.
Libro	PUT	/books/{id}	Actualiza completamente un libro existente.
Libro	DELETE	/books/{id}	Elimina un libro por ID (respuesta 204).
Relación	**GET**	**/libraries/{id}/books**	**Endpoint Anidado: Obtiene todos los libros asociados a una biblioteca específica.**

3. Implementación y Documentación

Creación de recursos (POST)

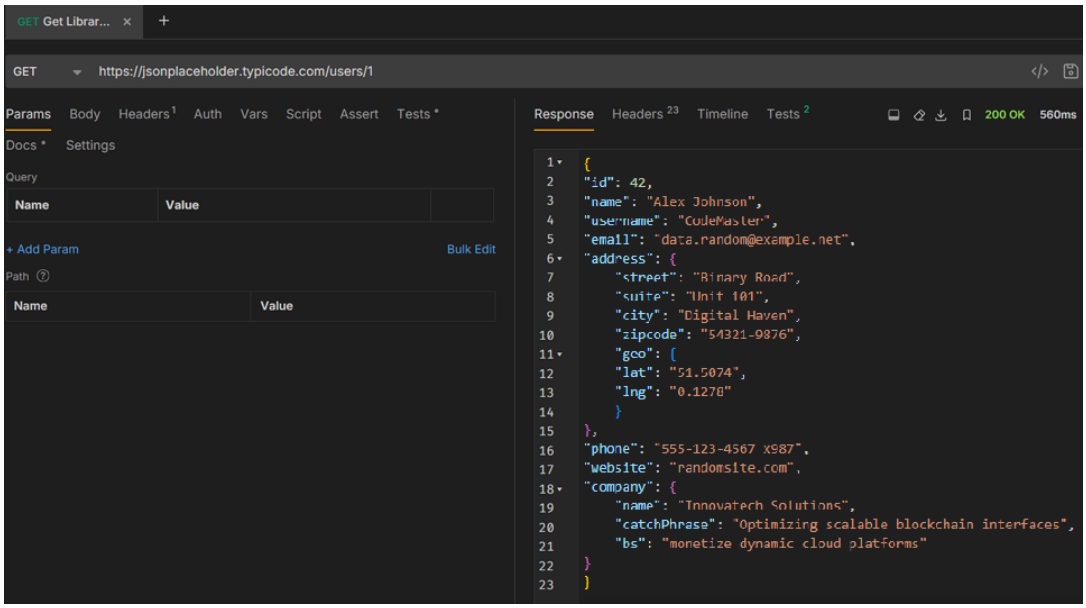
Se verifica que la creación de un recurso resulte en un código de estado 201 Created y devuelva el objeto con el ID generado.





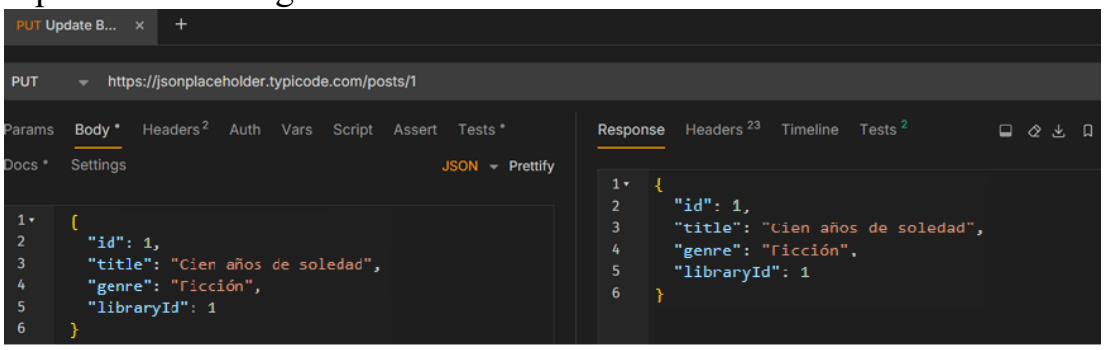
Lectura y relación (GET)

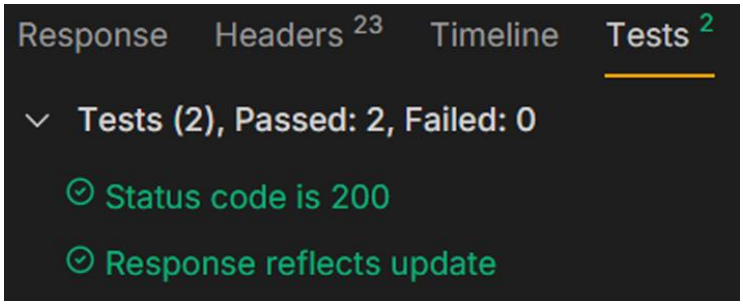
Se comprueba la obtención de recursos individuales



Actualización de recursos (PUT)

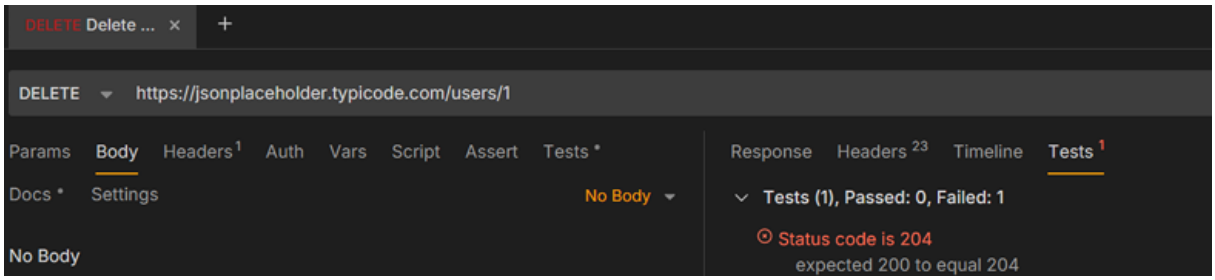
Se utiliza el método PUT para demostrar la actualización de un recurso por ID, esperando un código 200 OK.





Eliminación de recursos (Delete)

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



4. Evidencia Práctica de Métodos CRUD

- El proyecto confirmó la capacidad de diseñar una API alineada con la arquitectura REST, utilizando sustantivos en plural para los recursos y métodos HTTP (GET, POST, PUT, DELETE) para representar operaciones CRUD de manera semánticamente correcta.
- Se implementó un diseño efectivo para manejar la relación 1:N (Biblioteca a Libros). Esto se logró mediante la clave foránea y rutas anidadas, asegurando la integridad referencial y permitiendo la navegación intuitiva entre los recursos.
- La generación del archivo de especificación OpenAPI (Swagger) es fundamental. Este documento sirve como un contrato formal que mejora la comunicación entre equipos, facilita la generación automática de clientes y acelera el proceso de *onboarding* de nuevos desarrolladores.
- El proyecto valida un flujo de trabajo profesional donde la documentación, el diseño y las pruebas se mantienen sincronizados. Esta práctica garantiza la calidad, consistencia y mantenibilidad del servicio a lo largo del tiempo.
- Se reafirmó la importancia de utilizar URLs claras y códigos de estado HTTP correctos para mejorar la experiencia del desarrollador, y la necesidad de una planificación cuidadosa al modelar las relaciones entre recursos.