|  |
| --- |
|  |
| **EVALUACIÓN 2 - 20%.**  UNIDAD I:  Sistema de prestamo de libros (WEB API) Puntaje Obtenido: Nota: |
| FECHA: El 03-06-2025 |

# NOMBRE ALUMNO:

**CARRERA**: Ingeniería en Ejecución Informática

**CÓDIGO**: IEI- 089

**ASIGNATURA:** Programación .NET **DOCENTE:** Maximiliano Moraga

**MODALIDAD:** Presencial Diurna

Lea atentamente las instrucciones y cada pregunta antes de responder.

**Estructura:** La evaluación consta de 85 puntos. Requiere 51 puntos para aprobar.

**Entregables:** El estudiante luego del tiempo establecido entregara el proyecto con la solución.

**Tiempo asignado:** Usted dispone de 7 días, para desarrollar esta evaluación.

**Modalidad: Grupal (Máximo 3 integrantes).**

**Entregables**

* Código fuente en GitHub (con README) o comprimido en zip por el aula virtual.
* Capturas de pruebas de cada funcionalidad requerida en este documento o video demostrativo.
* Nombre del comprimido: NombreApellido\_Eva\_2.zip o repositorio de git declarado en un .txt

# Caso de Estudio: Desarrollo de una Web API RESTful en ASP.NET Core

La Biblioteca Municipal “Letras Libres” desea modernizar su sistema de préstamos y gestión de libros. Actualmente usan hojas de cálculo y registros manuales. Tu equipo de desarrollo ha sido contratado para construir una **API RESTful** que permita digitalizar este proceso y sentar las bases para futuras aplicaciones móviles y web.

Desarrollar una **Web API** utilizando **ASP.NET Core** y **Entity Framework Core**, que permita gestionar libros, usuarios y préstamos de forma eficiente, segura y escalable.

# Requisitos Funcionales

**Libros**

* GET /api/libros – Lista todos los libros.
* GET /api/libros/{id} – Muestra los detalles de un libro.
* POST /api/libros – Crea un nuevo libro.
* PUT /api/libros/{id} – Edita la información de un libro.
* DELETE /api/libros/{id} – Elimina un libro (solo si no está prestado).

**Usuarios**

* GET /api/usuarios – Lista de usuarios registrados.
* POST /api/usuarios – Registrar un nuevo usuario.
* GET /api/usuarios/{id}/prestamos – Ver el historial de préstamos del usuario.

**Préstamos**

* POST /api/prestamos – Registrar un nuevo préstamo (libro → usuario).
* POST /api/devoluciones – Registrar la devolución de un libro.

# Entity Framework Core (OBLIGATORIO)

* Uso de **EF Core como ORM** con SQL Server (LocalDB, SQLite o InMemory para pruebas).
* Configuración de DbContext e inyección en Program.cs.
* Uso de **migraciones** para crear y actualizar la base de datos.
* Relación entre entidades bien definida (uno-a-muchos, etc.).
* Validaciones en modelos con **Data Annotations**. (Opcional)
* Uso de Include() para obtener datos relacionados en consultas (como historial de préstamos).

**Documentación**

* Configurar Swagger para probar y documentar los endpoints.

# Validaciones

Las pruebas de las validaciones son con respecto a funcionalidad, entregar captura del programa funcionando mal con su código y captura del sistema funcionando bien con su código corregido.

# Criterios de evaluación

|  |  |
| --- | --- |
| Criterio | Puntaje |
| Configuración de proyecto, limpieza y estructura | 10 pts |
| Implementación de CRUD para libros, usuarios y préstamos | 20 pts |
| Uso correcto de Entity Framework Core | 20 pts |
| Relaciones entre entidades correctamente modeladas | 10 pts |
| Validaciones y control de errores | 10 pts |
| Pruebas funcionales (Swagger/Postman) | 10 pts |
| Código limpio y documentado | 5 pts |

# Rubrica de evaluación

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criterio** | **Excelente (100%))** | **Bueno (80%)** | **Aceptable (60%)** | **Insuficiente (40%)** | **Deficiente (20% o menos)** | **Puntaje Máx.** |
| **Configuración de proyecto, limpieza y estructura** | Estructura de carpetas correcta, sin archivos innecesarios, Program.cs y Startup.cs/builder bien organizados. | Estructura mayormente correcta, 1-2 archivos innecesarios. | Estructura con errores o carpetas mal ubicadas. | Código confuso, archivos mezclados, sin organización clara. | Proyecto desordenado, difícil de ejecutar o entender. | 10 pts |
| **CRUD para libros, usuarios y préstamos** | CRUD completo (Create, Read, Update, Delete) para las 3 entidades. Total: 12 endpoints funcionales. | CRUD completo para 2 entidades y parcial para 1. (8–11 endpoints funcionales) | Solo operaciones de lectura o escritura. (4–7 endpoints) | CRUD incompleto para todas las entidades. (1–3 endpoints) | CRUD inexistente o no funcional. | 20 pts |
| **Uso correcto de Entity Framework Core** | Usa DbContext, DbSet, migraciones, LINQ y async/await correctamente en 90%+ de los casos. | Uso correcto en 70–89% de los casos. Migraciones usadas. | EF usado pero con errores de sintaxis o mal manejo de relaciones. (50–69%) | Uso incorrecto en mayoría de los casos. No aplica buenas prácticas de EF. | No usa EF Core o lo usa erróneamente. | 20 pts |
| **Relaciones entre entidades correctamente modeladas** | Relaciones 1:N y N:M definidas en modelos y en DbContext. Navegación funcional. | Relaciones definidas pero sin navegación clara o con omisiones menores. | Relaciones presentes pero con errores de clave foránea o navegación. | Relaciones ausentes o mal implementadas. | No hay relaciones o están rotas. | 10 pts |
| **Validaciones y control de errores** | Validaciones en DTOs/Controllers. Manejo de excepciones y mensajes claros. (90%+ cubierto) | Validaciones presentes pero sin manejo completo de errores. (70–89%) | Validaciones en algunos campos, control limitado de errores. (50–69%) | Muy pocas validaciones. Errores no controlados. | No hay validaciones ni manejo de errores. | 10 pts |
| **Pruebas funcionales (Swagger/Postman)** | Documentación completa en Swagger. Pruebas en Postman evidencian 90%+ de endpoints. | Swagger funcional, 70–89% probado. Capturas/Postman entregadas. | Swagger básico, Postman con 50–69% de pruebas. | Swagger con errores o rutas incompletas. Postman deficiente. | No hay Swagger ni evidencia de pruebas. | 10 pts |
| **Código limpio y documentado** | Código indentado, sin redundancias, con comentarios explicativos en métodos y clases. | Código claro, pocos comentarios, pero entendible. | Código funcional, con comentarios mínimos o desorganizados. | Código difícil de seguir, muchas líneas redundantes. | Código sucio, sin comentarios, sin seguir convenciones. | 5 pts |

# Escala de notas:

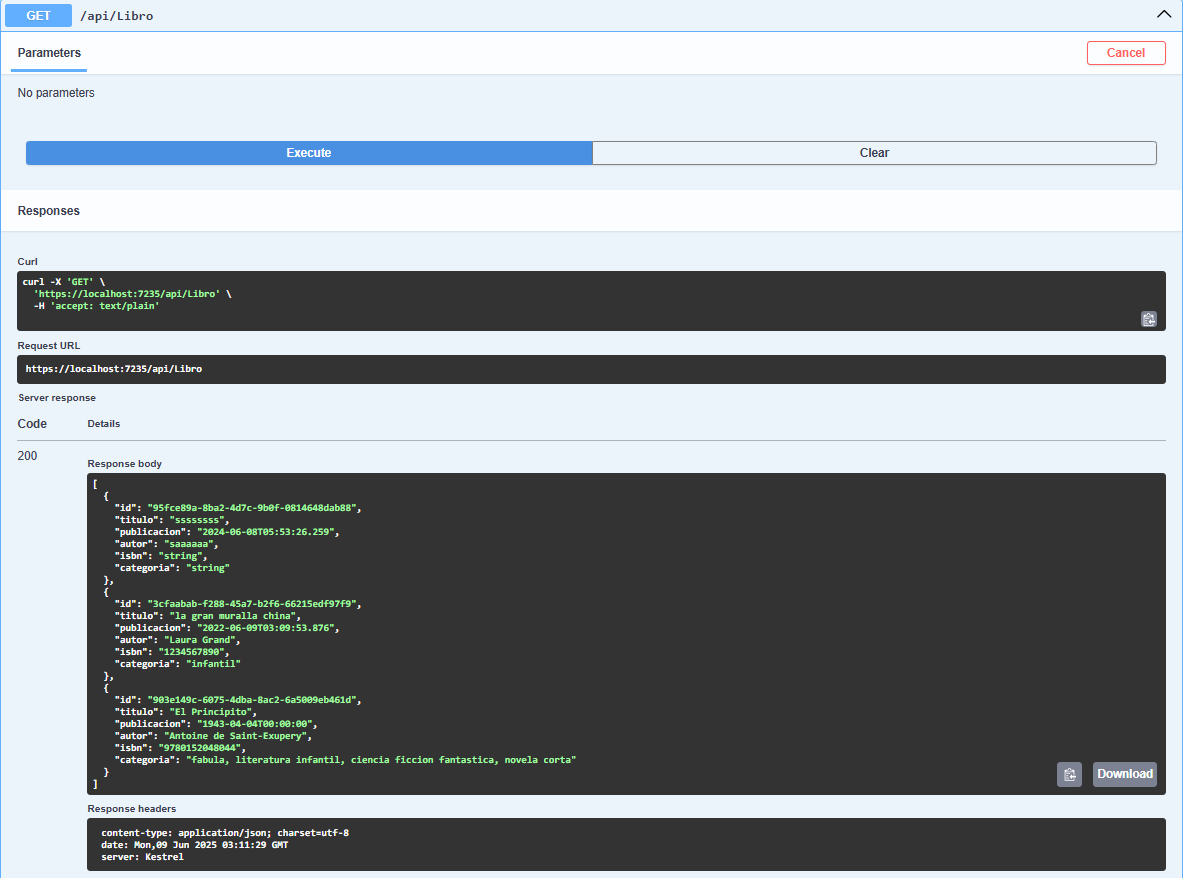
Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

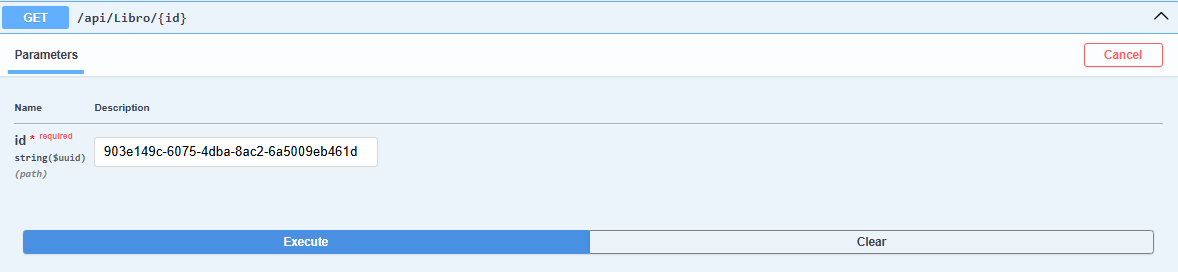
# Capturas de pantalla

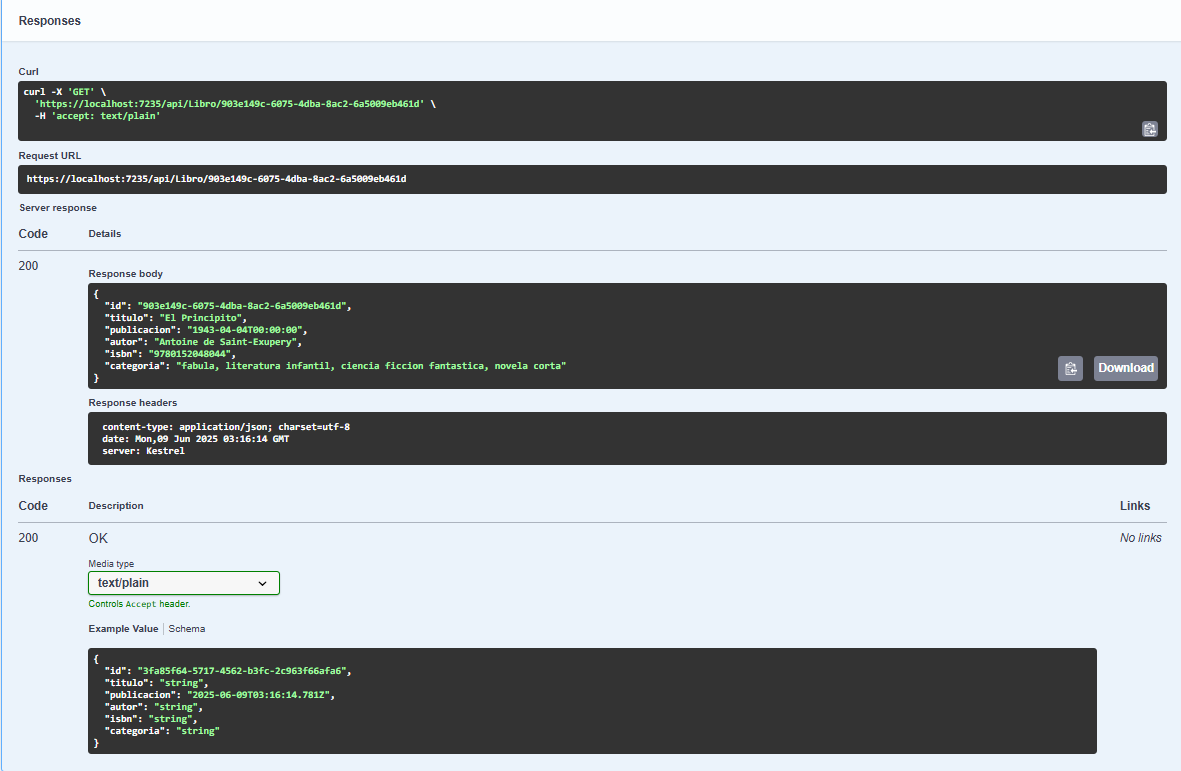
Funcionalidad de Libro

N°1– Lista todos los libros.

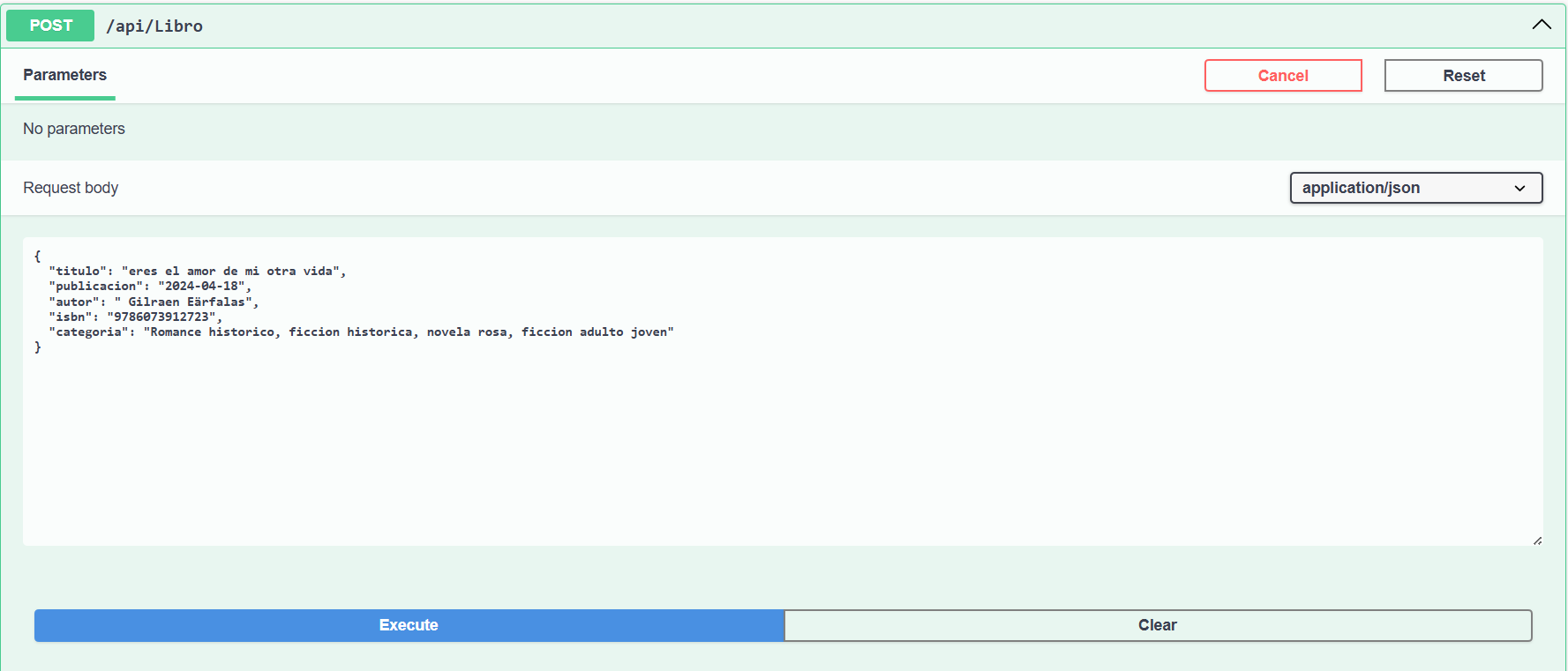


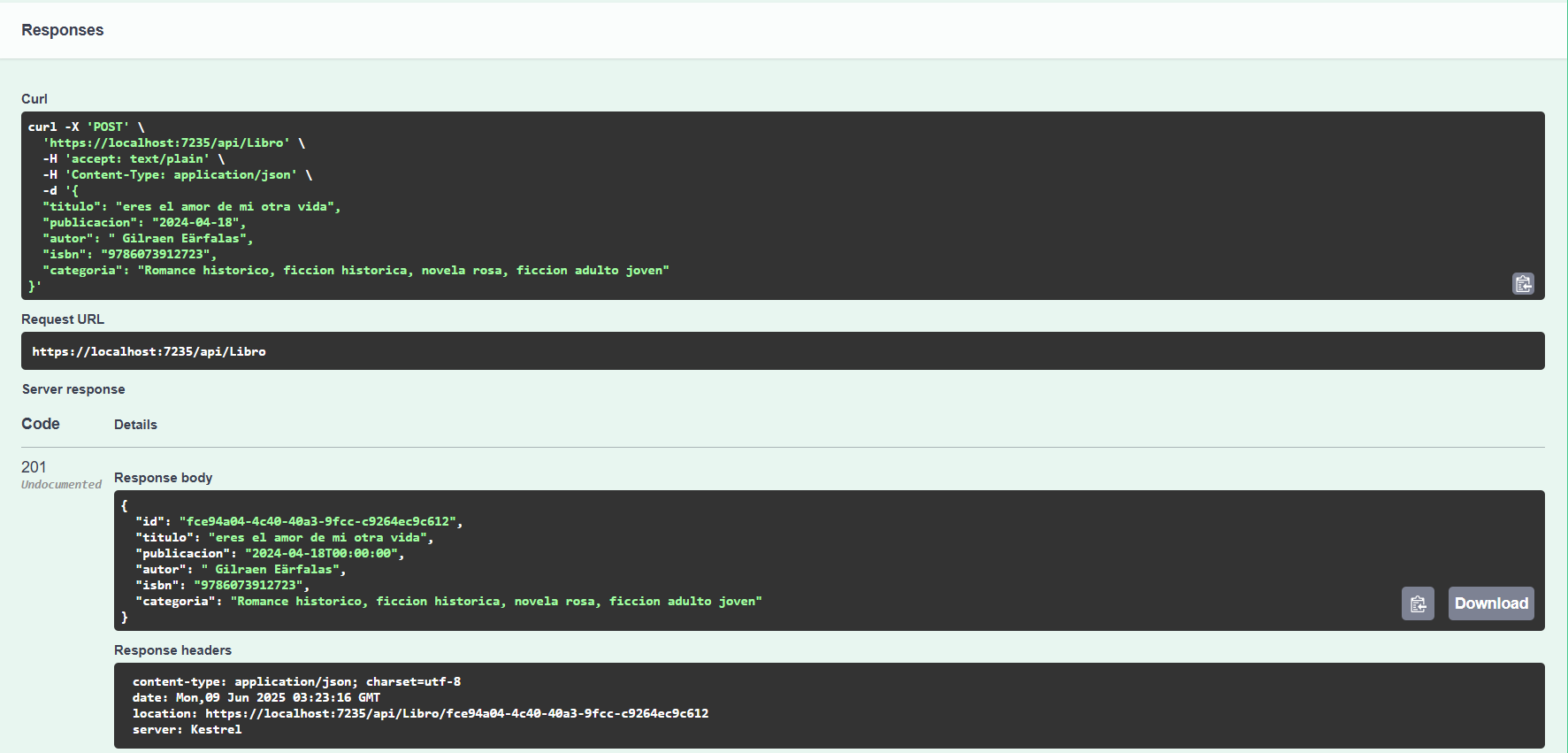
N°2- Muestra los detalles de un libro.



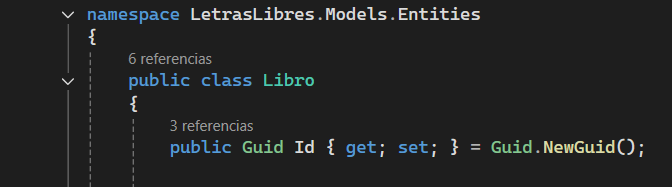


N°3- Crea un nuevo libro.



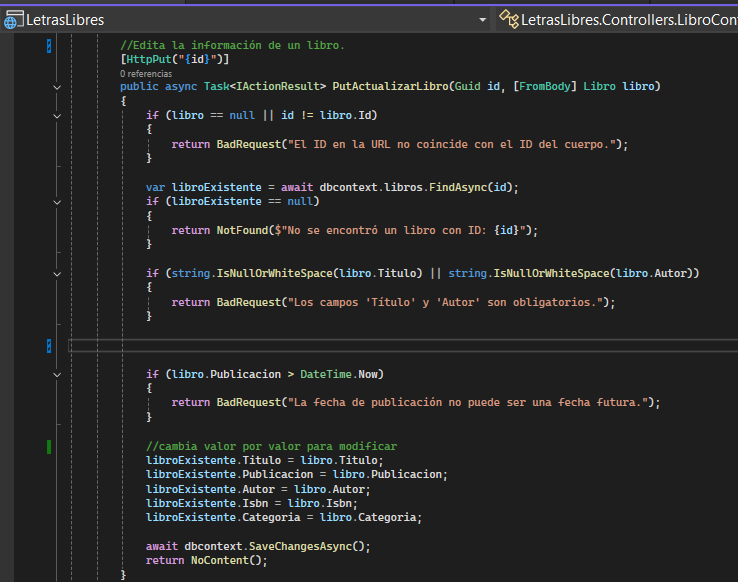


El campo id del JSON no es necesario de llenar, ya que se genera automáticamente

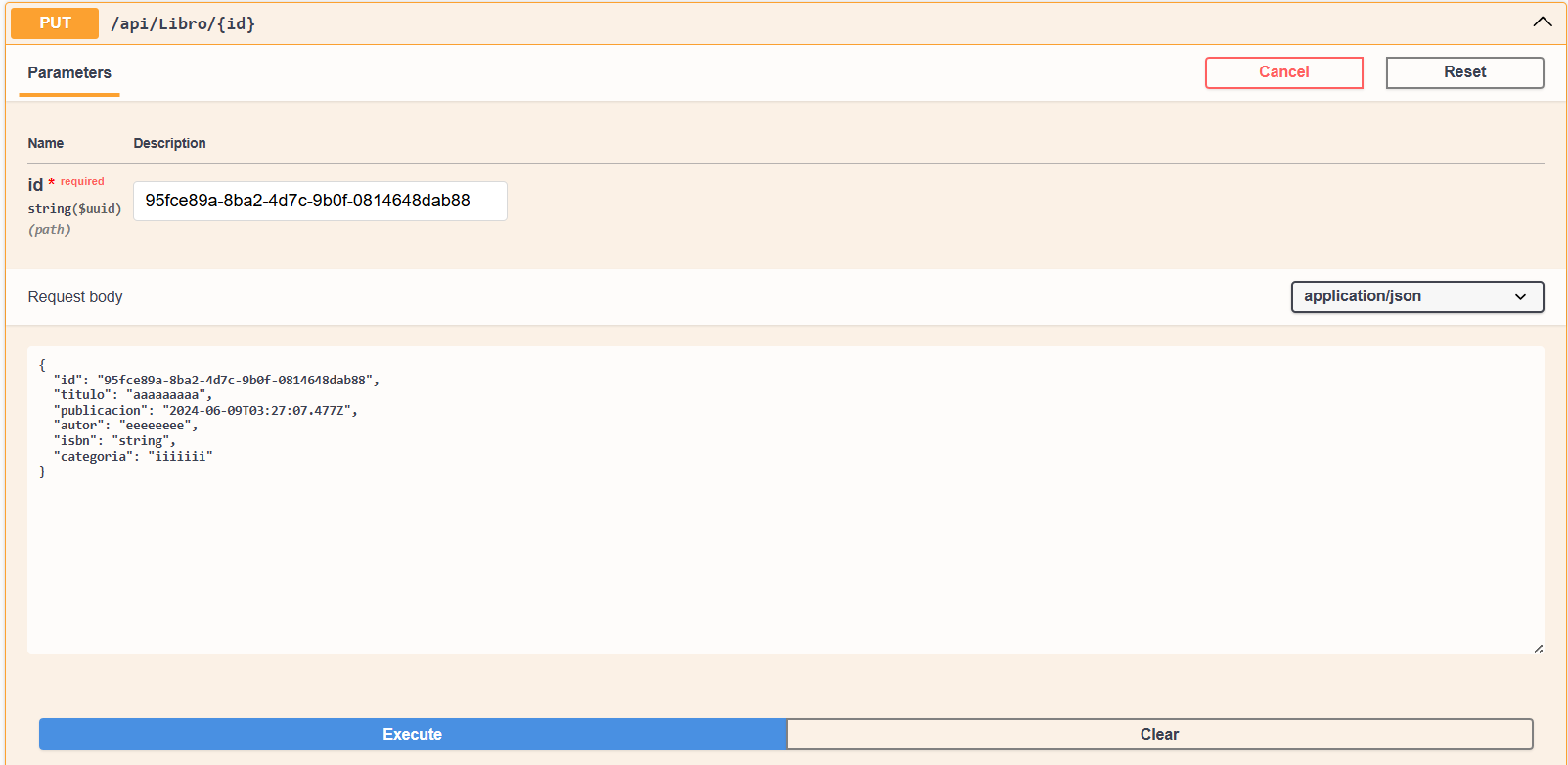


N°4 y Validación del método PUT

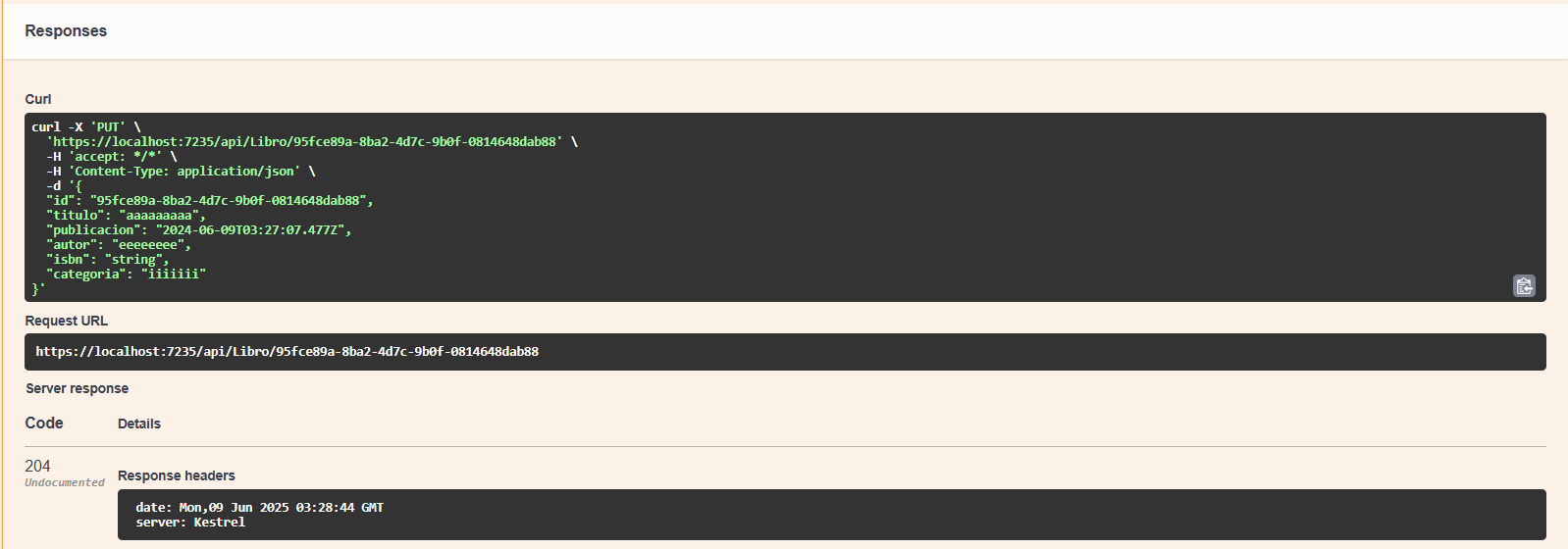
Con error



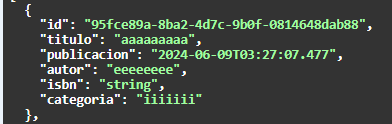
Este es el método PUT funcionalidad



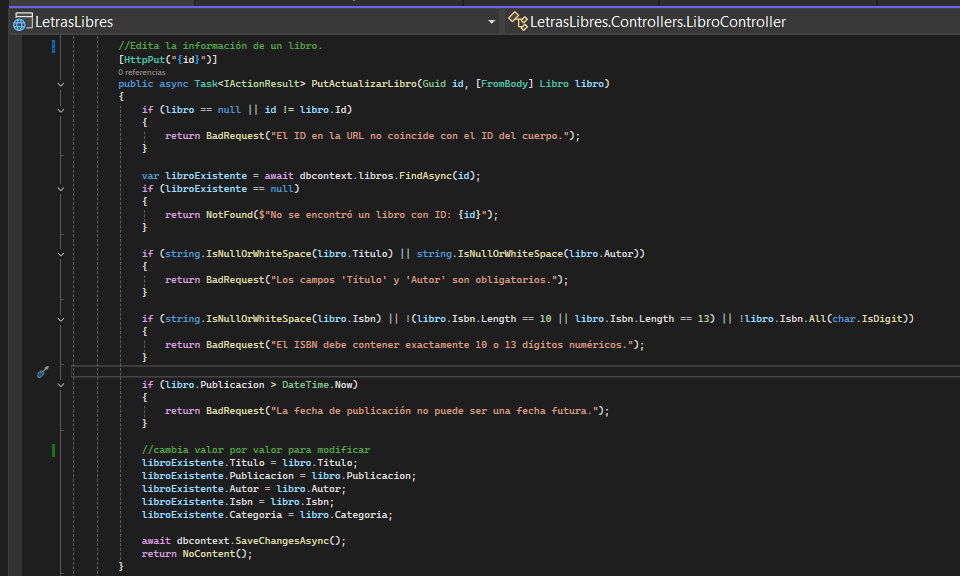
Respuesta



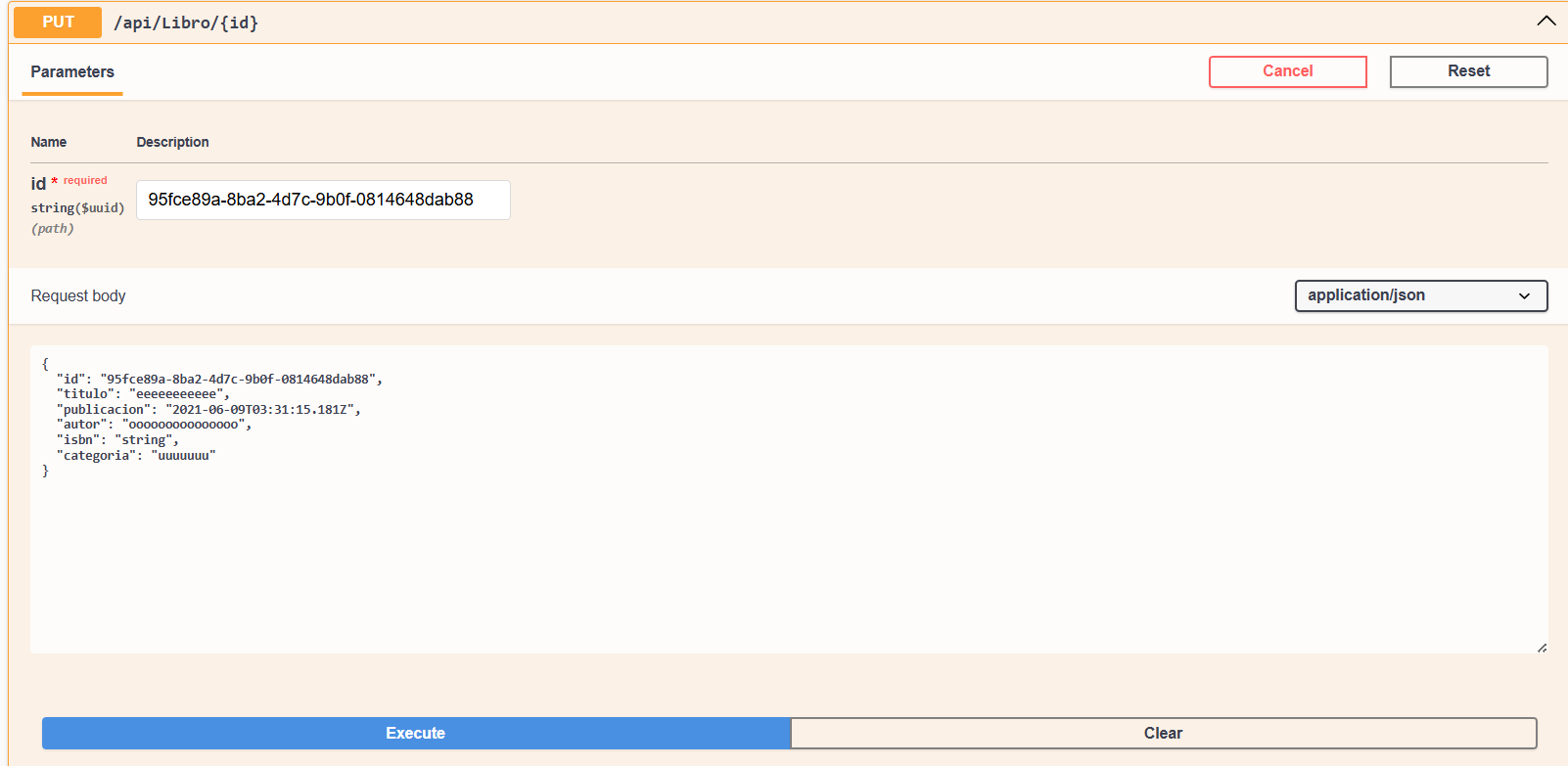
Esto es lo que sigue devolviendo el método PUT por que no remplaza los valores



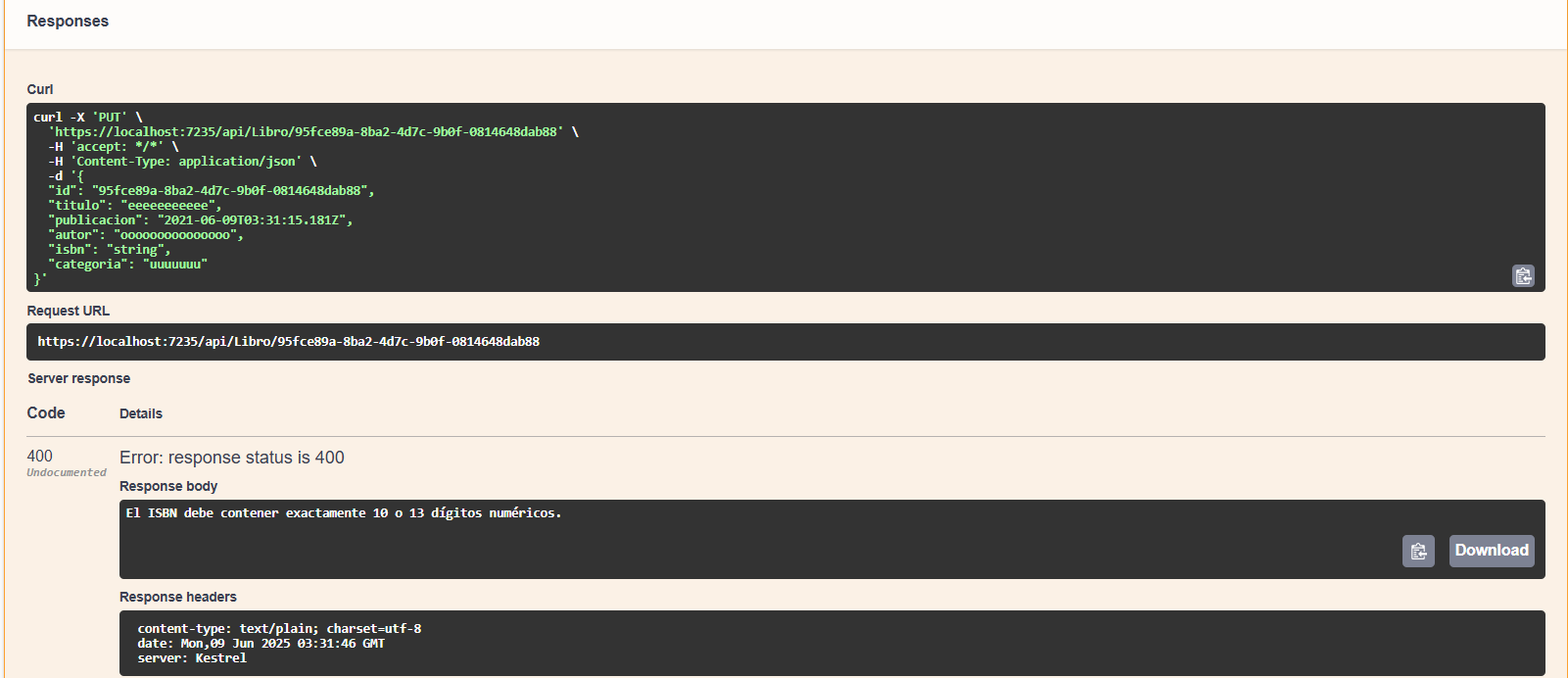
Sin error



Ejecución del método PUT

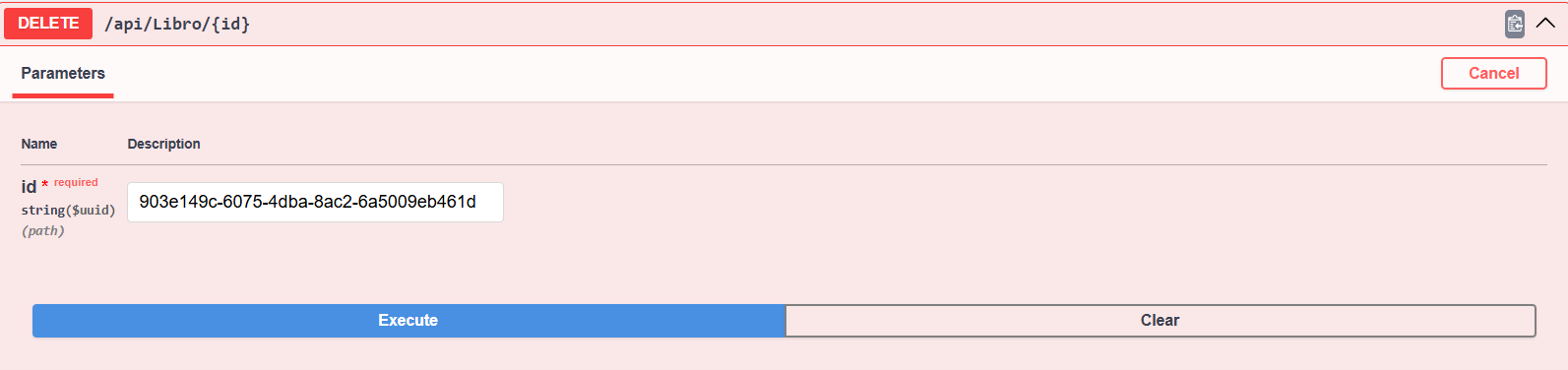


Respuesta



Ahora corregido el método PUT pide que se le asigne cierta cantidad de valores en el campo Isbn

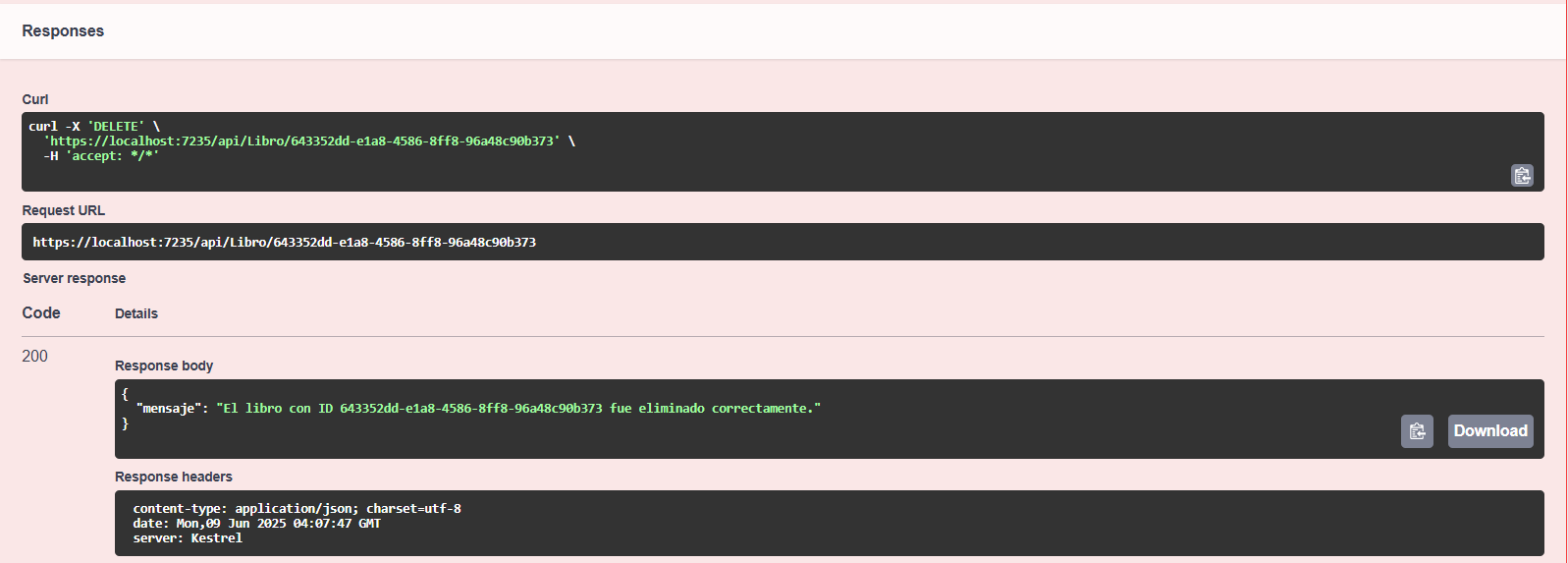
N°5- Elimina un libro (solo si no está prestado).



El método Delete da error al intentar eliminar un libro prestado



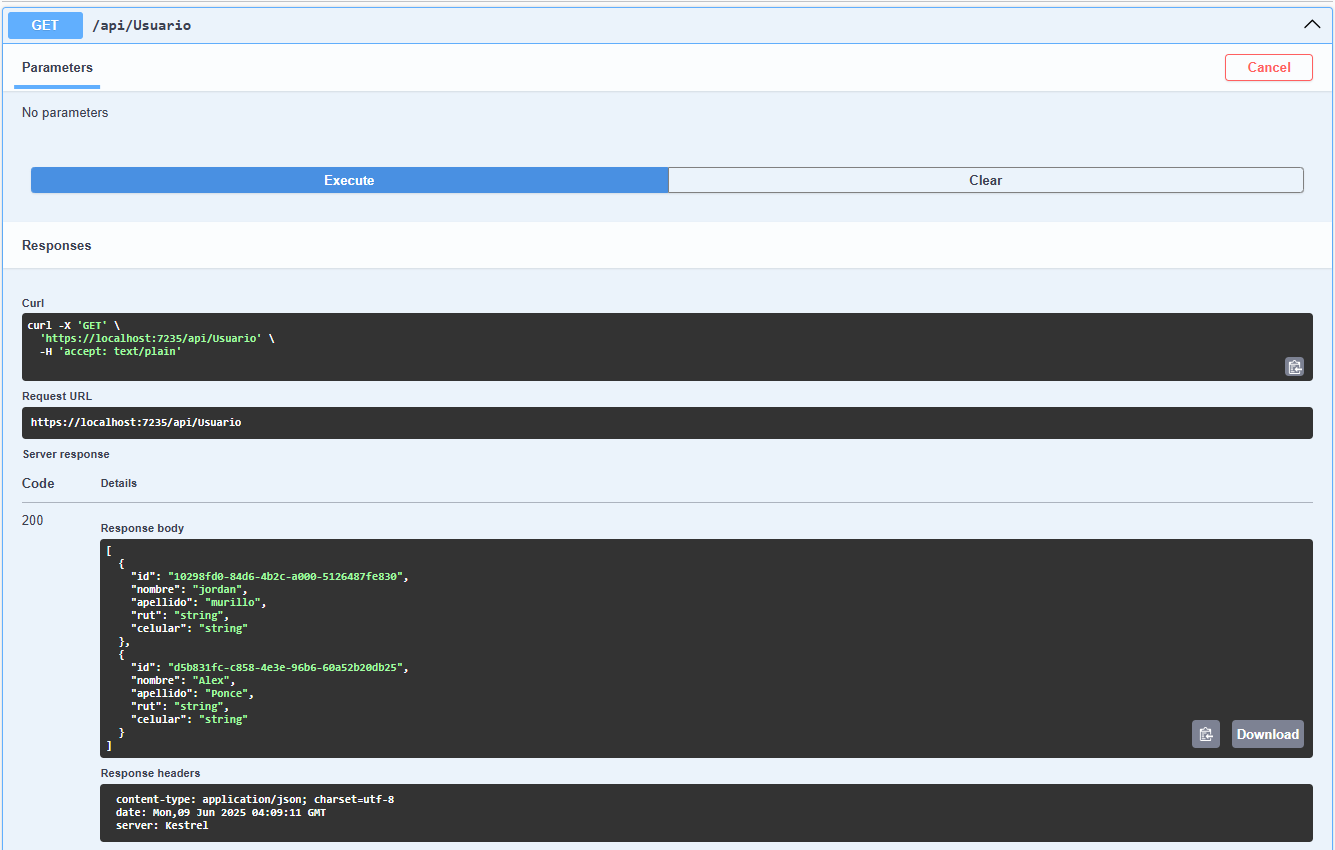
Una vez quitado del préstamo ya se puede eliminar el libro



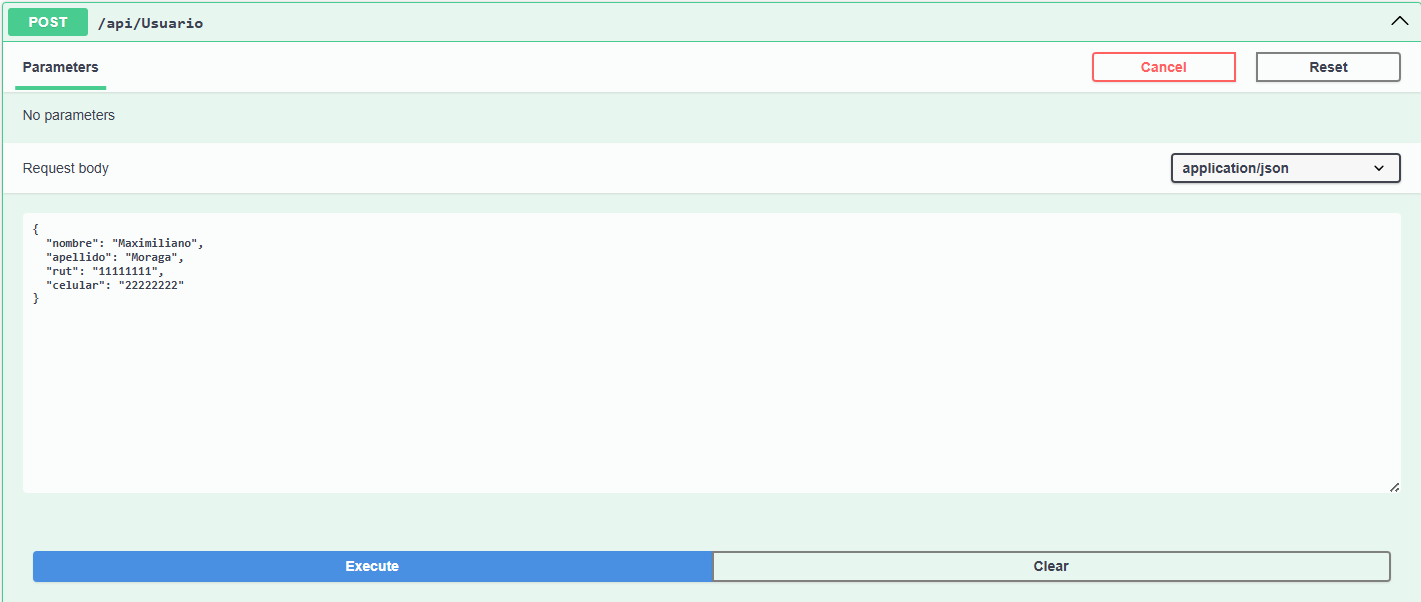
# Capturas de pantalla

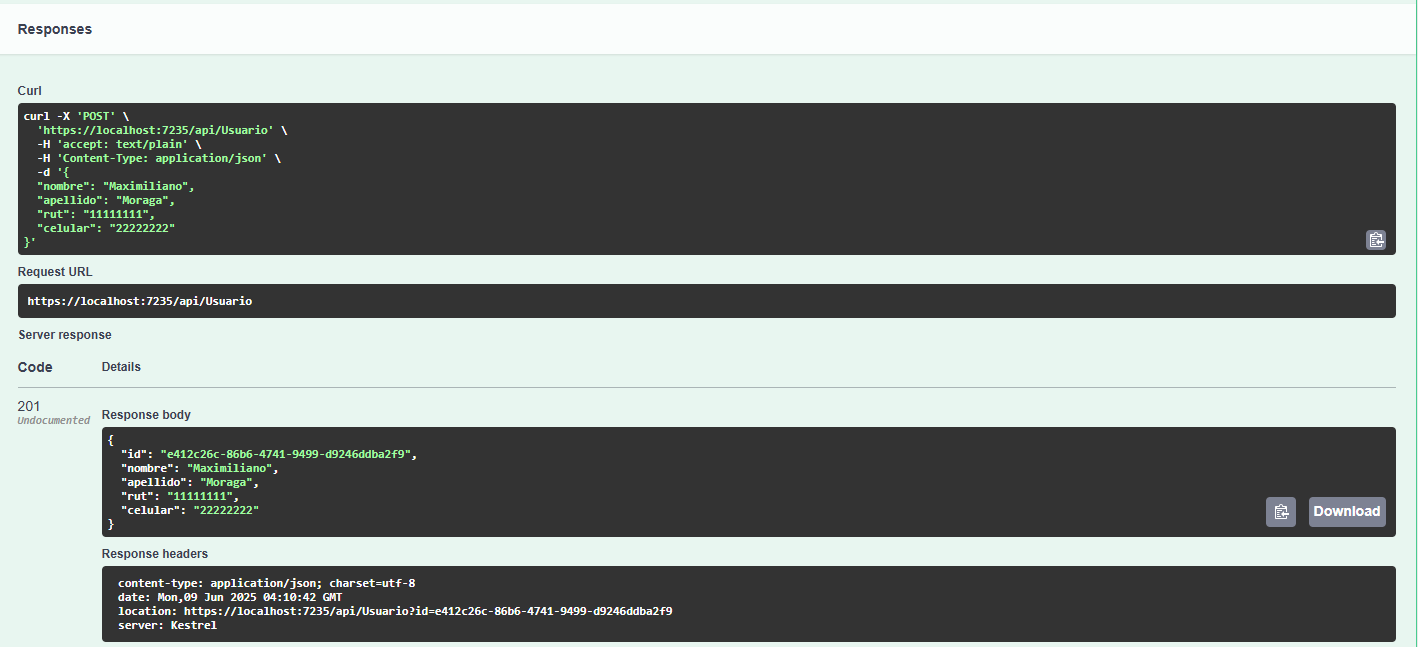
Usuarios

N°1- Lista de usuarios registrados.

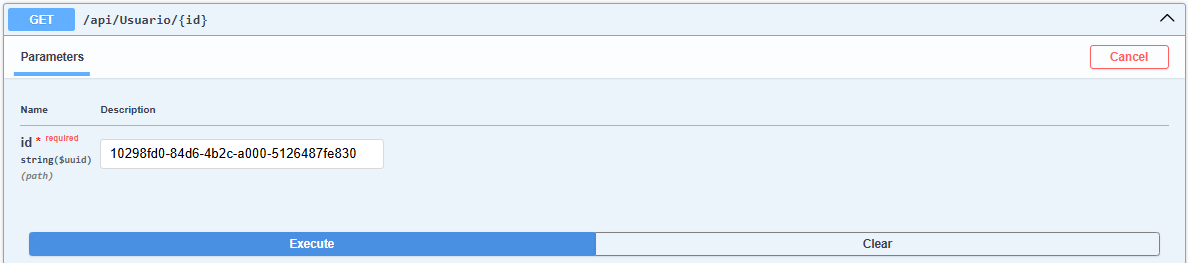


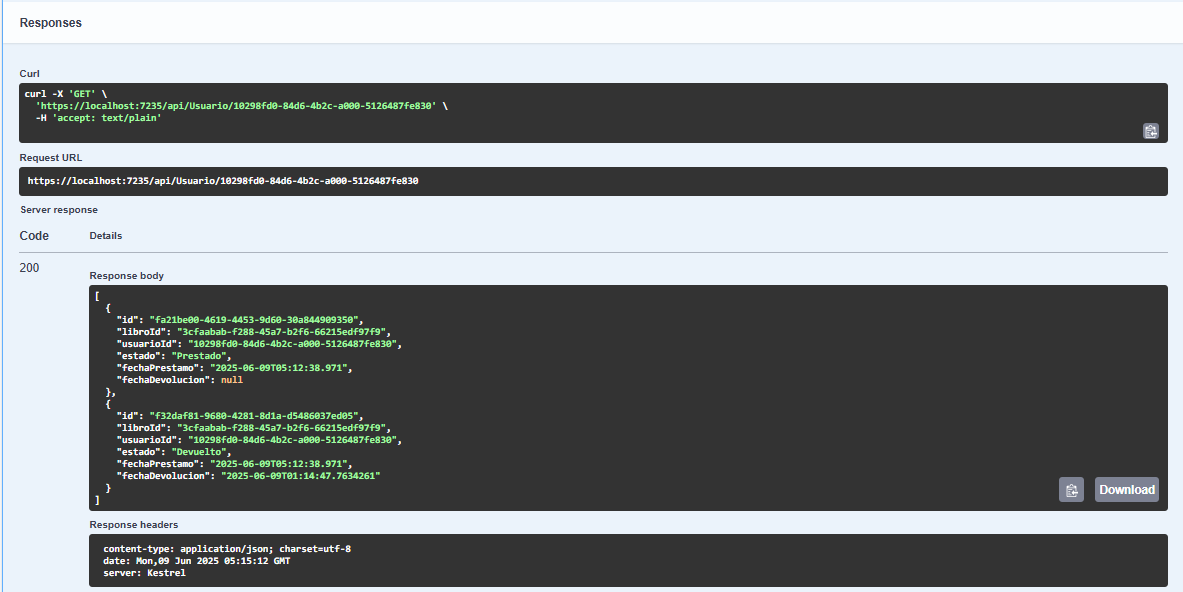
N°2- Registrar un nuevo usuario.





N°3- Ver el historial de préstamos del usuario.

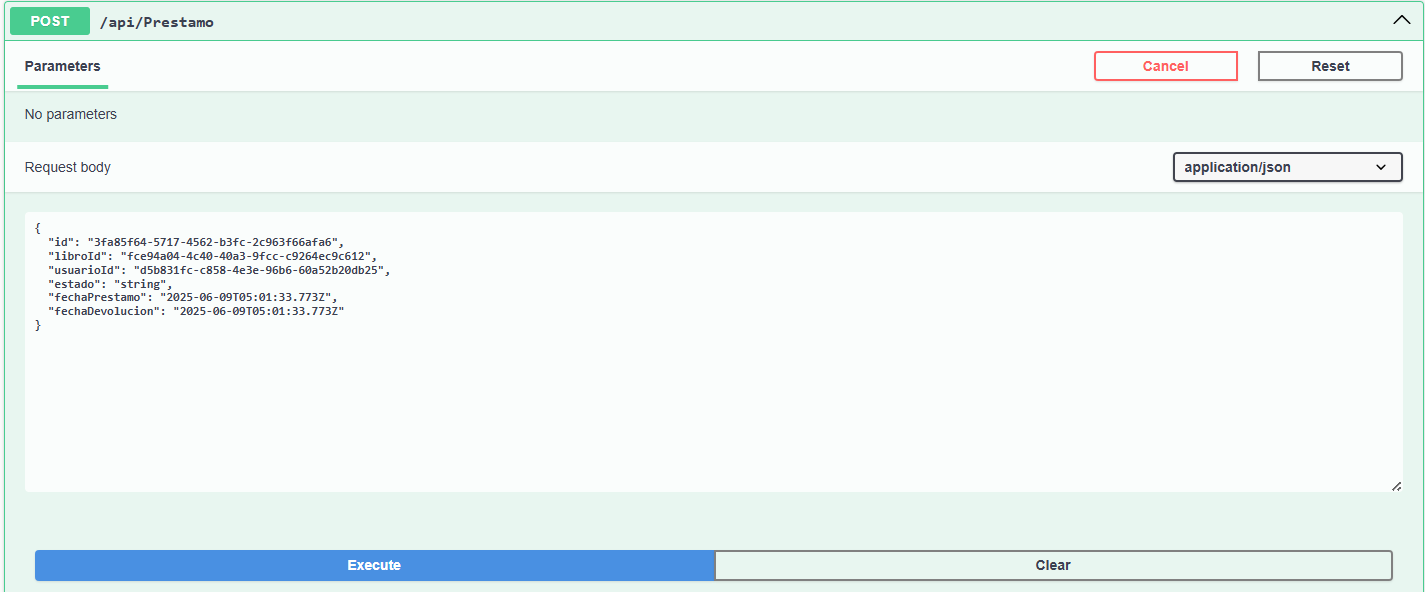


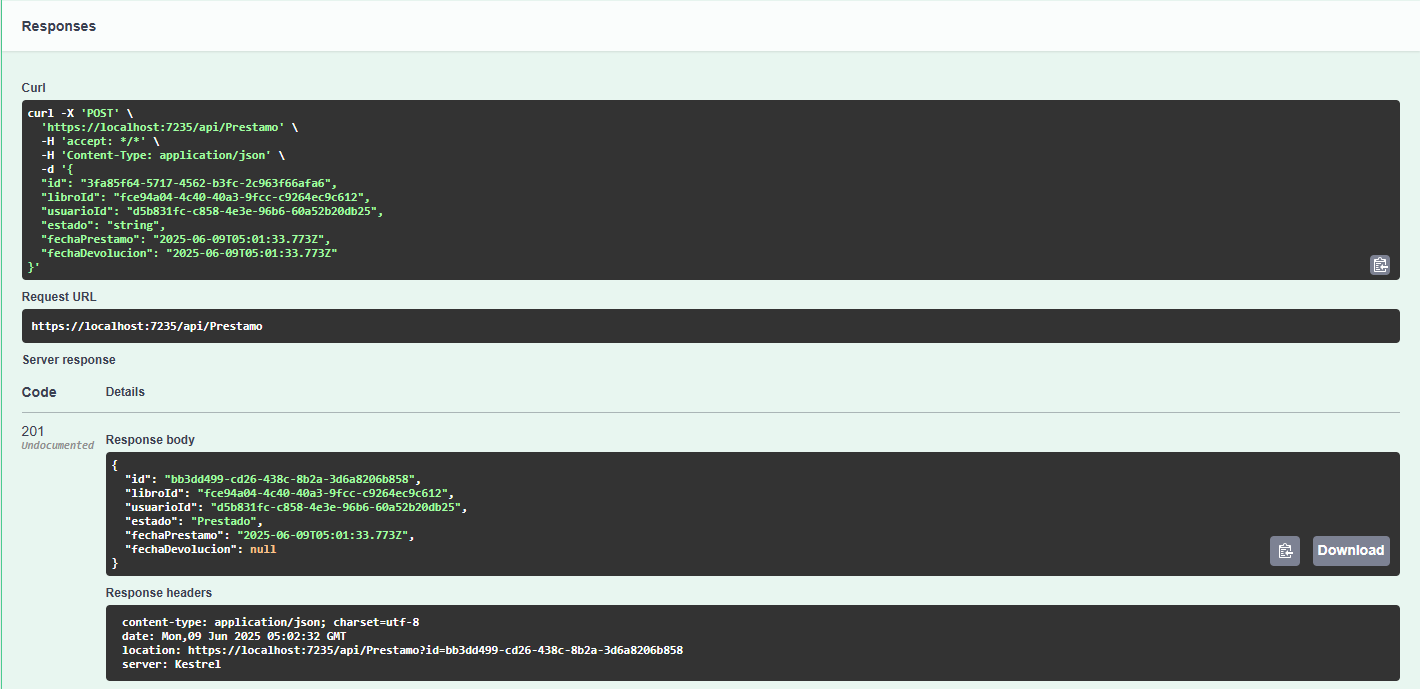


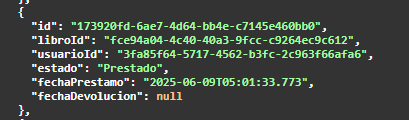
# Capturas de pantalla

Préstamo

N°1- Registrar un nuevo préstamo (libro → usuario).







N°2- Registrar la devolución de un libro.

