Integrantes:

Douglas Alfredo Lino Henriquez

Rodrigo Aldair Lopez Pereira

César Elmer Vanegas Mejía

Indice

[Introduccion 2](#_Toc187012939)

[Descripcion del diseño 2](#_Toc187012940)

[Diagrama de arquitectura 2](#_Toc187012941)

[Diagrama de aplicación 3](#_Toc187012942)

[Modelo de datos 3](#_Toc187012943)

[Documentacion de Endpoints 3](#_Toc187012944)

[Endpoint 1 GET: Obtener todos los usuarios 4](#_Toc187012945)

[Endpoint 2 POST: Crear un nuevo usuario 4](#_Toc187012946)

[Endpoint 3 PUT: Actualizar un usuario existente 5](#_Toc187012947)

[Endpoint 4 DELETE: Eliminar un usuario existente 6](#_Toc187012948)

[Instrucciones de Configuración y Ejecución 8](#_Toc187012949)

[Requisitos Previos 8](#_Toc187012950)

[Pasos para Compilar y Ejecutar el Proyecto 8](#_Toc187012951)

[Configuración de la Base de Datos 8](#_Toc187012952)

[Swagger UI 9](#_Toc187012953)

# Introduccion

Esta api es un sistema mini CRUD (Create, Read, update, Delete) para la gestión de usuarios (id, nombre, email), permitiendo registrar, consultar, actualizar y eliminar usuarios desde una base de datos.

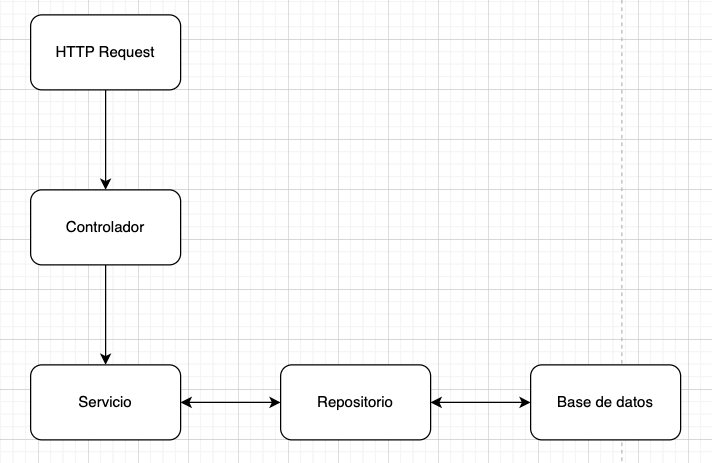
Es un ejemplo sencillo de implementación de un CRUD, fácilmente se puede implementar para plataformas tipo educativas, sistemas de gestión o empreas.

# Descripcion del diseño

## Diagrama de arquitectura

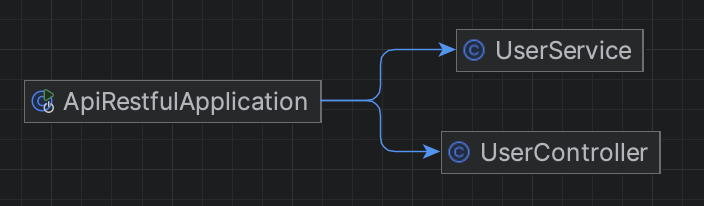
Implementa el patron de diseño MVC (Modelo-Vista-Controlador)

* Modelo (M): Representa las entidades de la base de datos (por ejemplo, User).
* Vista (V): En una API RESTful, está representada por las respuestas JSON enviadas al cliente.
* Controlador (C): Maneja las solicitudes HTTP y llama a los servicios y repositorios necesarios.



## Diagrama de aplicación

Este representa como interactúan las clases y el modelo MVC



## Modelo de datos

La entidad principal es User, que tiene los siguientes atributos:

|  |  |  |
| --- | --- | --- |
| Atributo | Tipo | Descripción |
| id | BIGINT (PK) | Identificador único del usuario. |
| name | VARCHAR(255) | Nombre del usuario. |
| email | VARCHAR(255) | Correo electrónico del usuario. |

# 

# Documentacion de Endpoints

## Endpoint 1 GET: Obtener todos los usuarios

* Endpoint: /api/users
* Método HTTP: GET
* Descripción: Devuelve una lista de todos los usuarios registrados.
* Parámetros: Ninguno.
* Códigos de Respuesta:
* 200 OK: Lista de usuarios devuelta correctamente.
* Ejemplo de Respuesta:

[

{

"id": 1,

"name": "John Doe",

"email": "johndoe@example.com"

},

{

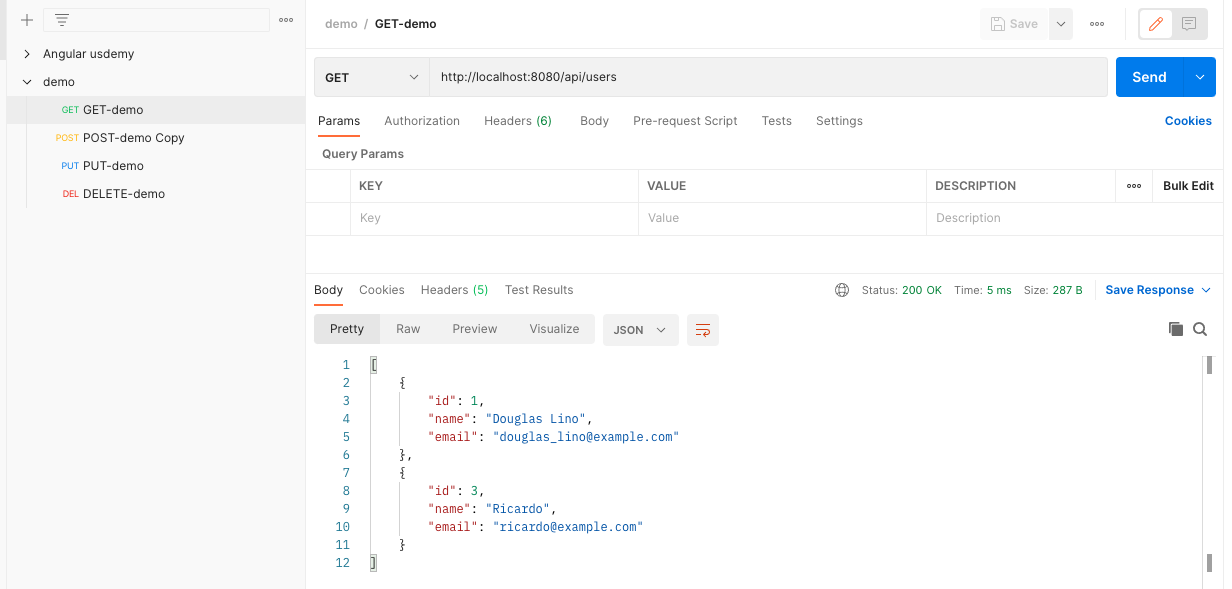
"id": 2,

"name": "Jane Doe",

"email": "janedoe@example.com"

}

]



## Endpoint 2 POST: Crear un nuevo usuario

* Endpoint: /api/users
* Método HTTP: POST
* Descripción: Crea un nuevo usuario en la base de datos.
* Parámetros:
  + Cuerpo de la Solicitud (JSON):

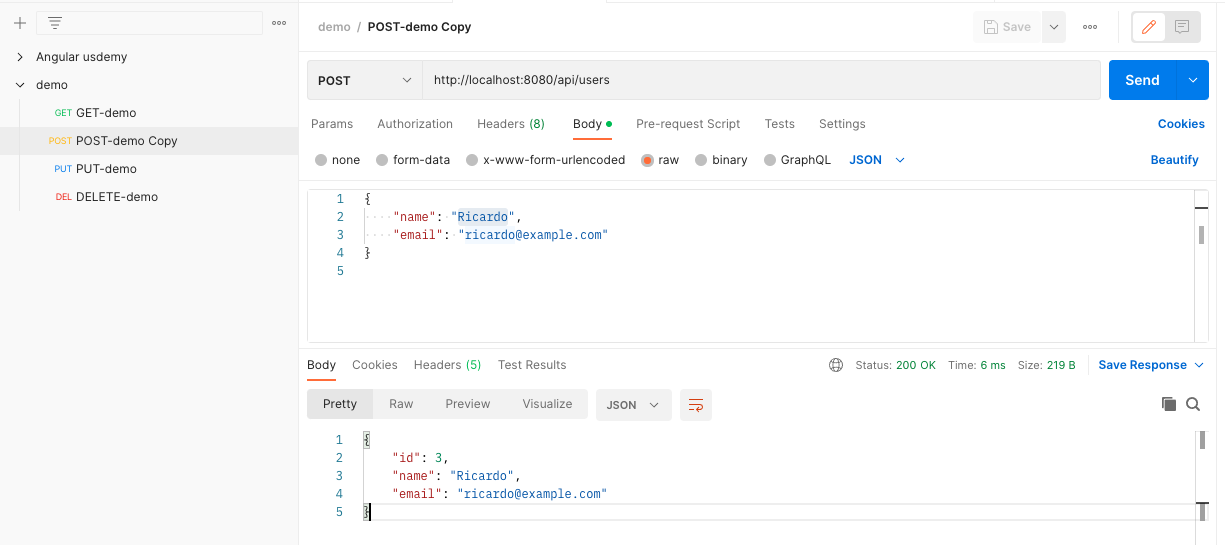
{

"name": "Ricardo",

"email": "ricardo@example.com"

}

* Códigos de Respuesta:
  + 201 Created: Usuario creado exitosamente.
* Ejemplo de Respuesta:



## Endpoint 3 PUT: Actualizar un usuario existente

* Endpoint: /api/users/{id}
* Método HTTP: PUT
* Descripción: Actualiza los datos de un usuario existente.
* Parámetros:
  + Parámetros de Ruta:
    - id (Requerido): ID del usuario a actualizar.
  + Cuerpo de la Solicitud (JSON):

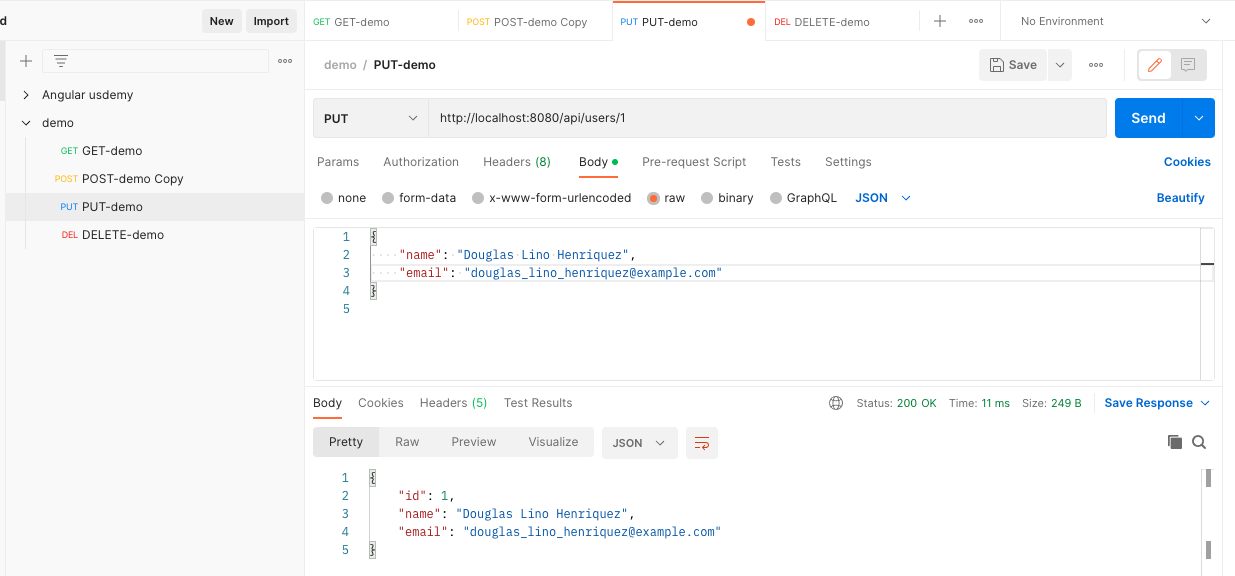
{

"name": "Douglas Lino Henriquez",

"email": "douglas\_lino\_henriquez@example.com"

}

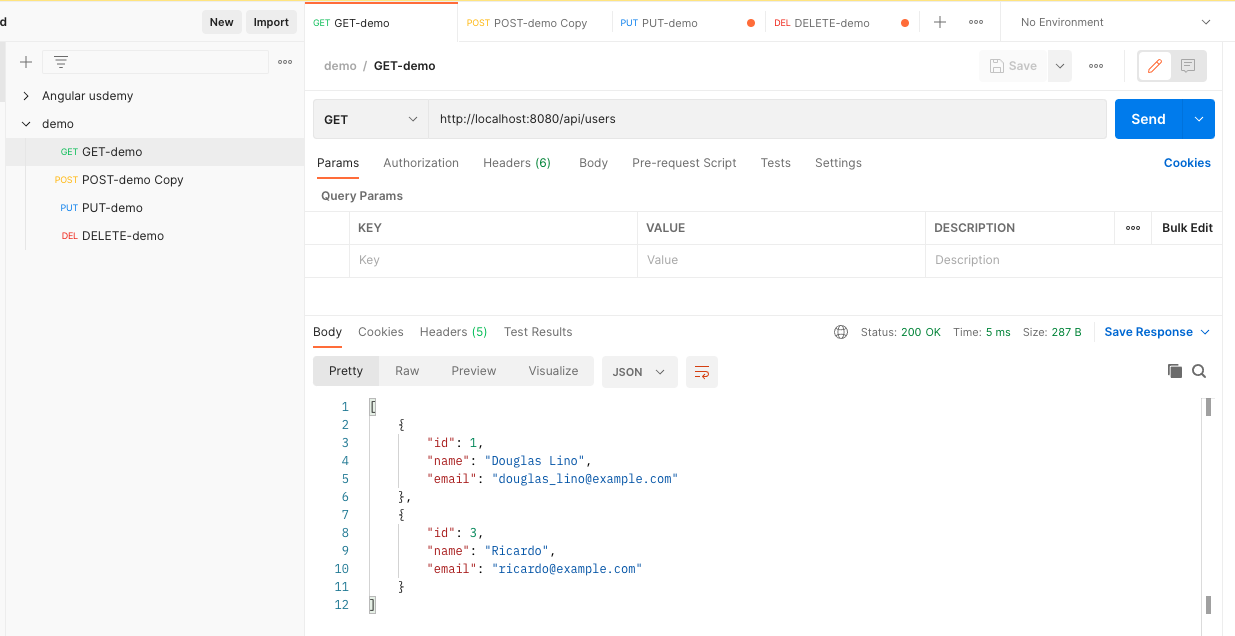
* Códigos de Respuesta:
  + 200 OK: Usuario actualizado exitosamente.
* Ejemplo de Respuesta:



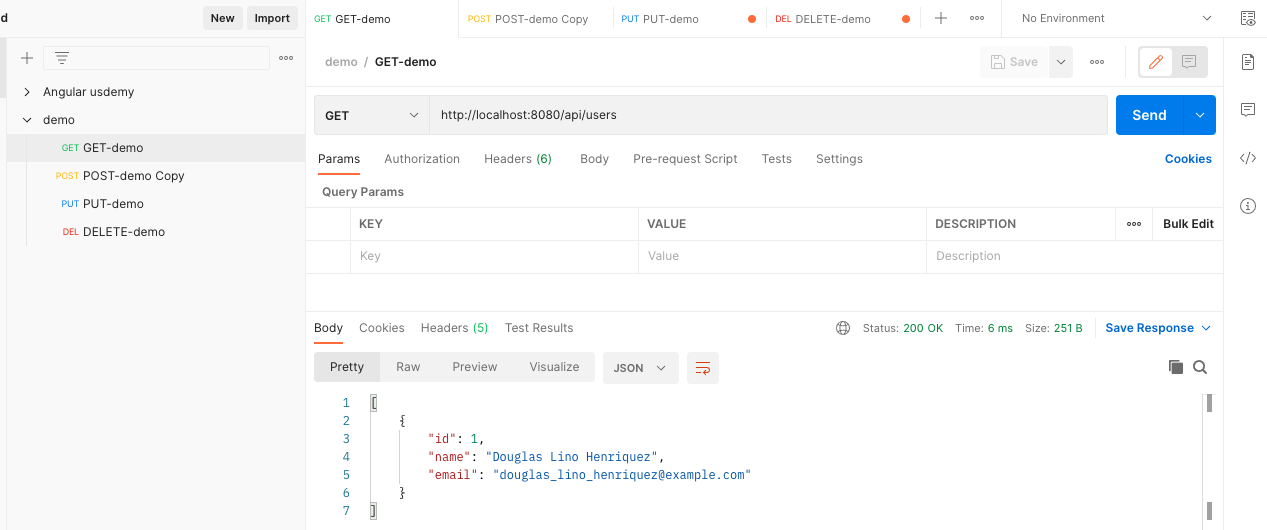
## Endpoint 4 DELETE: Eliminar un usuario existente

* Endpoint: /api/users/{id}
* Método HTTP: DELETE
* Descripción: Elimina un usuario existente por su ID.
* Parámetros:
  + Parámetros de Ruta:
    - id (Requerido): ID del usuario a eliminar.
* Códigos de Respuesta:
  + 204 No Content: Usuario eliminado exitosamente.

Ejemplos de GET de todos los usuarios:



Ahora borramos el usuario con ID “3”



# Instrucciones de Configuración y Ejecución

## Requisitos Previos

1. Instalar JDK (versión mínima Java 17).
2. Instalar Maven para gestionar las dependencias.
3. Tener instalado un IDE como IntelliJ IDEA o Eclipse.

## Pasos para Compilar y Ejecutar el Proyecto

1. Clona el repositorio o descarga el .zip .
2. Abre el proyecto en el IDE.
3. Ejecuta los siguientes comandos en la terminal:

Para compilar el proyecto:

Ejecutar:

mvn clean install

Para ejecutar la aplicación:

Ejecutar

mvn spring-boot:run

La aplicación estará disponible en http://localhost:8080.

## Configuración de la Base de Datos

El archivo application.properties ya está configurado para usar una base de datos H2 en memoria:

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

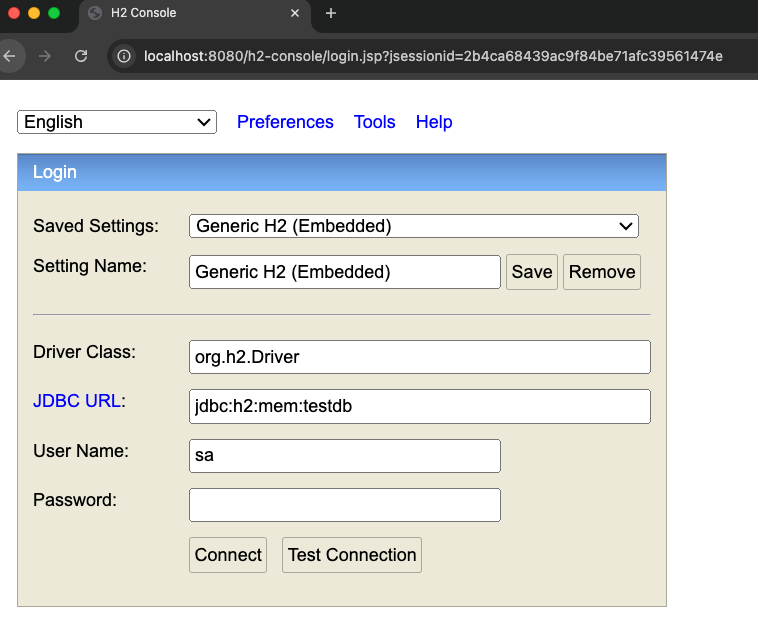
spring.jpa.hibernate.ddl-auto=update

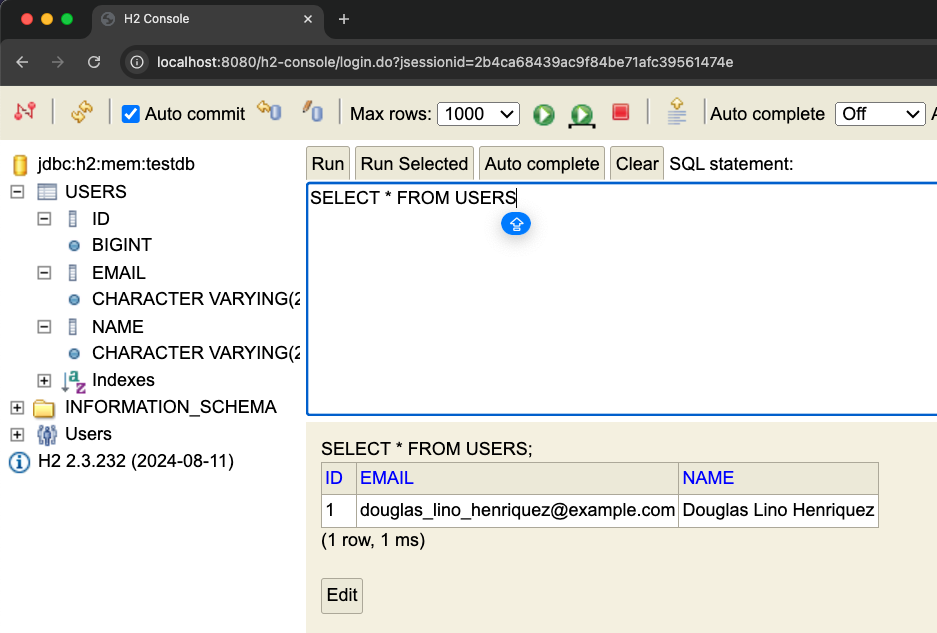
spring.h2.console.enabled=true

Para acceder a la consola H2:

1. Ir a http://localhost:8080/h2-console.
2. Ingresa los siguientes valores:

* JDBC URL: jdbc:h2:mem:testdb
* User Name: sa
* Password: (dejar vacío)

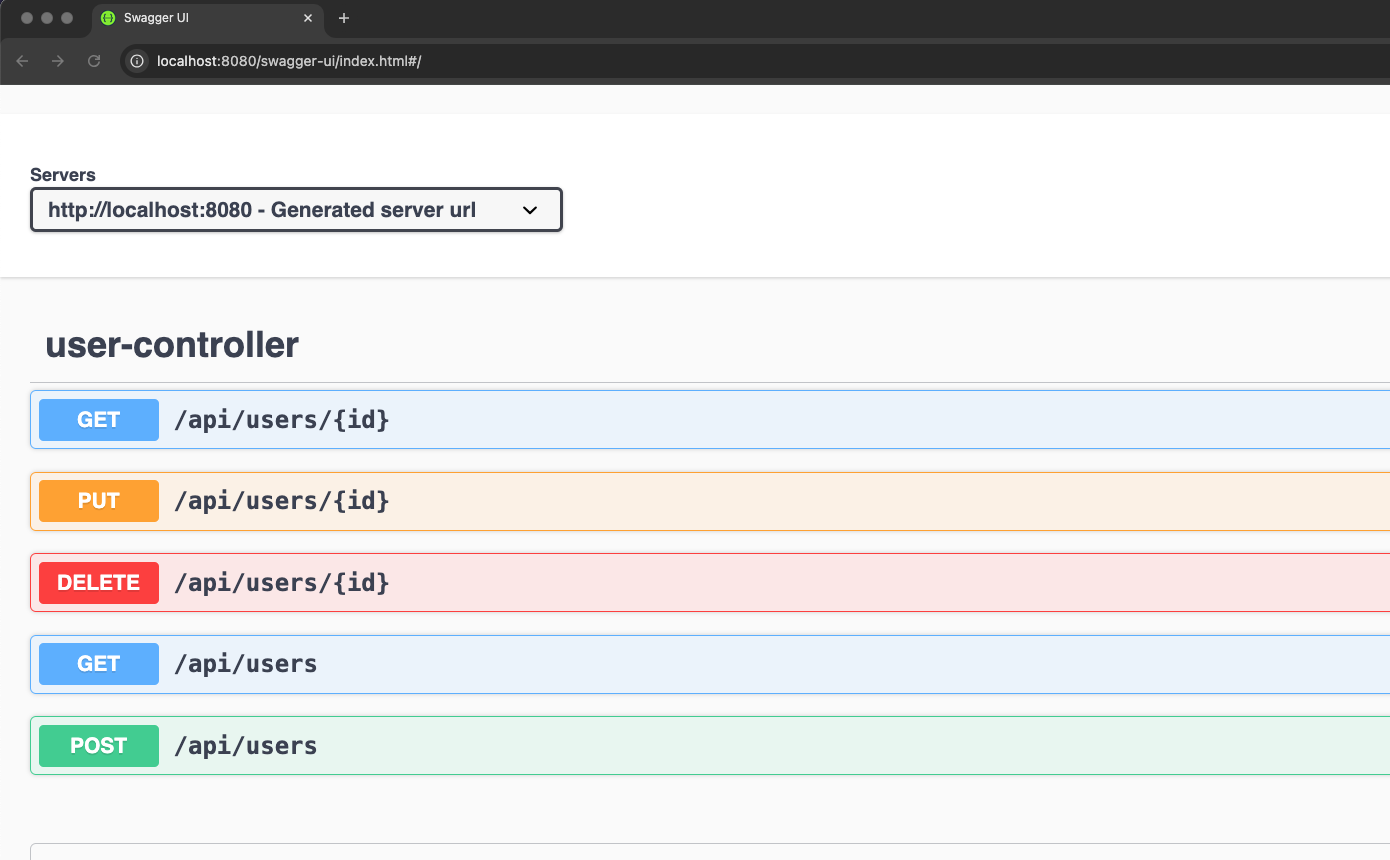




## Swagger UI

Tambein se instalo Swagger

Accediendo en el siguiente enlace: http://localhost:8080/swagger-ui/index.html#/



Por ejemplo con la petición GET all

