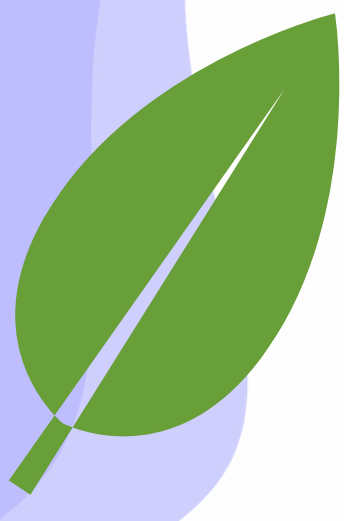
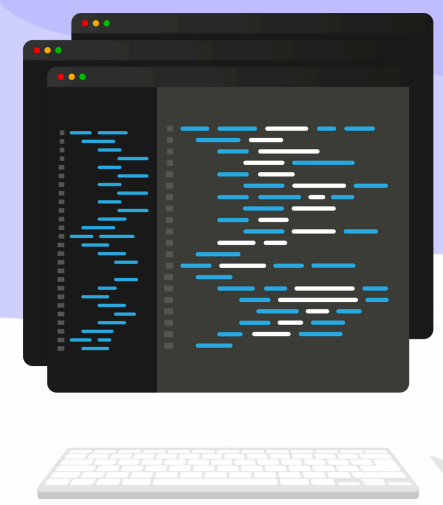




ACADEMIA XIDERAL

CRUD Mongo DB

PINTORES FAMOSOS

LISSET SÁNCHEZ

Antes de empezar quisiera dar una pequeña introducción sobre qué es el CRUD, como bien nos han enseñado esta semana en la academia es importante conocer de donde surge todo. Recordando, CRUD es un acrónimo de Create, Read, Update, Delete; que son las operaciones básicas para manipular datos en una base de datos.

Este proyecto en específico consiste en un CRUD para almacenar información de pintores famosos, incluyendo nombre, nacionalidad, estilo, fechas de nacimiento y fallecimiento, obras famosas y enlaces a imágenes de sus obras.

Las herramientas utilizadas fueron las siguientes:

- Java 17
- Spring Boot: framework que nos ayuda a construir aplicaciones Java rápidas y con buenas prácticas.
- Spring Data MongoDB: quien nos facilita la conexión y manipulación de documentos en MongoDB.
- MongoDB Compass: nuestra interfaz gráfica para gestionar la base de datos.
- Postman: herramienta que utilizamos para probar los endpoints del CRUD.

Pasaré ahora a la estructura del proyecto, la arquitectura que utilicé fue la convencional por capas. Creándolos empezando por el modelo y siguiendo el orden conforme a los menciono.

Modelo (Entity)

Mi modelo Representa un pintor dentro de la base de datos MongoDB, definí mi Clase Pintor con sus atributos de nombre, nacionalidad, estilo, fechaNacimiento, obrasFamosas e imagenes. Nuestro modelo nos sirve como plantilla para los documentos que se guardan en MongoDB.

Le anotamos con @Document(collection = "pintores") para indicar que los objetos de esta clase se guardan en la colección pintores.

Los atributos que utilicé fueron:

private String id;
private String nombre;
private String nacionalidad;
private String estilo;
private String fechaNacimiento;
private String fechaFallecimiento;
private List<String> obrasFamosas;
private List<String> imagenes;

Repositorio (Repository)

En la capa de Repositorio realicé la interfaz que extiende MongoRepository, la cual nos proporciona métodos ya predefinidos para interactuar con MongoDB, como guardar, consultar, actualizar y eliminar documentos. Esto nos permite manipular los datos de los pintores sin escribir consultas directamente, y Spring Boot se encarga de traducir estas llamadas a operaciones sobre la base de datos.

Servicio (Service + ServiceImpl)

En la capa de Service realicé la lógica de negocio como es lo convencional, es decir, cómo se crean, leen, actualizan y eliminan los pintores. La interfaz define qué métodos existen y la implementación contiene la lógica real usando el repositorio.

Controlador (Controller)

En la capa de Controller se exponen las rutas o endpoints de la API REST. En este proyecto, el recurso principal es /pintores. Desde ahí, podemos realizar las operaciones básicas: crear (POST), consultar todos (GET), consultar por id (GET), actualizar (PUT) y eliminar (DELETE). El controlador recibe la petición, la envía al servicio, y finalmente devuelve una respuesta al cliente.

Esto permite que la aplicación sea consumida desde Postman o un frontend.

Base de Datos (MongoDB con Docker)

La base de datos utilizada fue MongoDB, corriendo en un contenedor Docker. Se configuró en el archivo application.properties para que la aplicación pudiera conectarse y almacenar a los pintores.

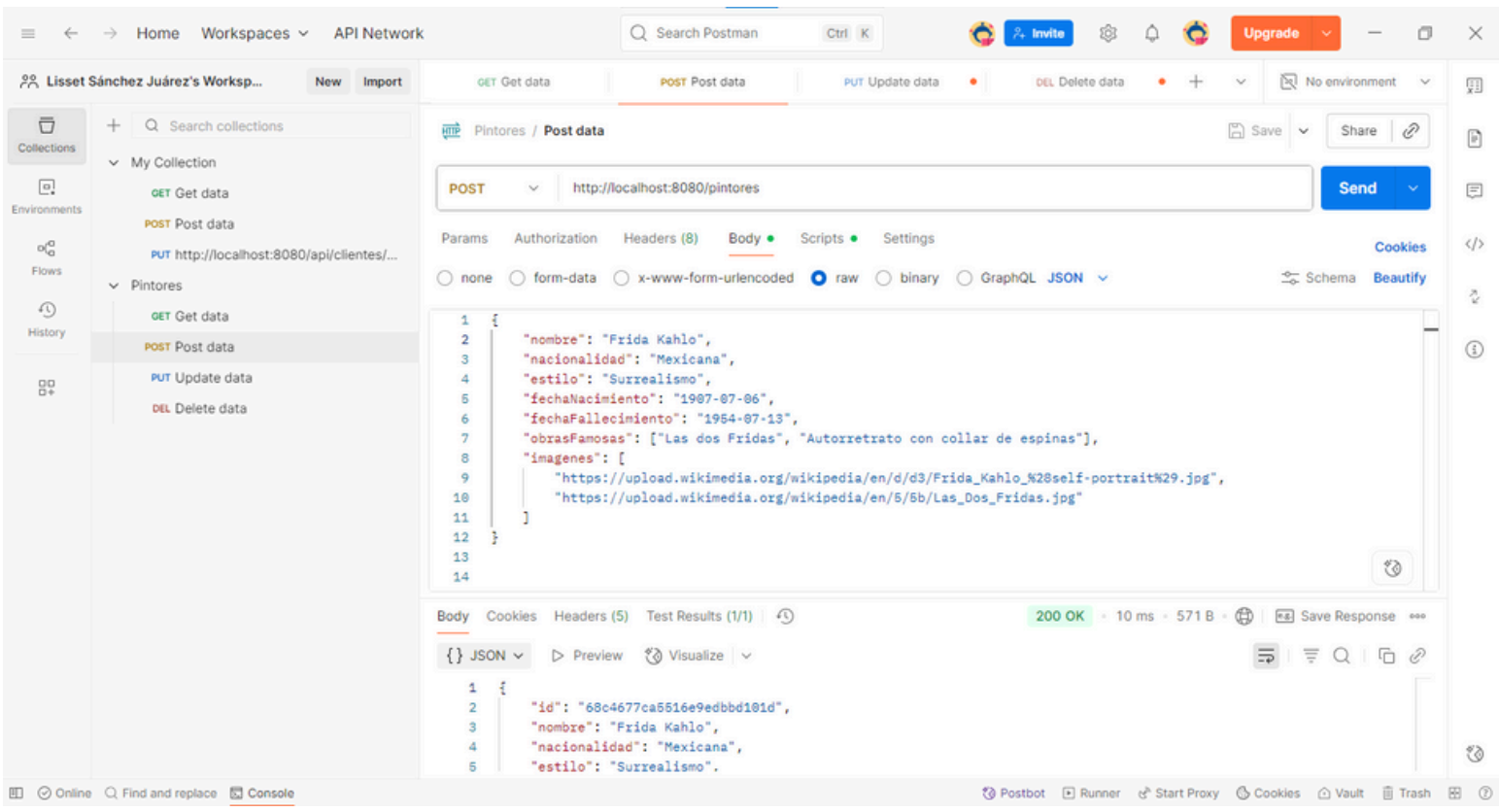
En el archivo application.properties configuré:

- spring.data.mongodb.host=localhost
- spring.data.mongodb.port=27017
- spring.data.mongodb.database=pintoresdb
-

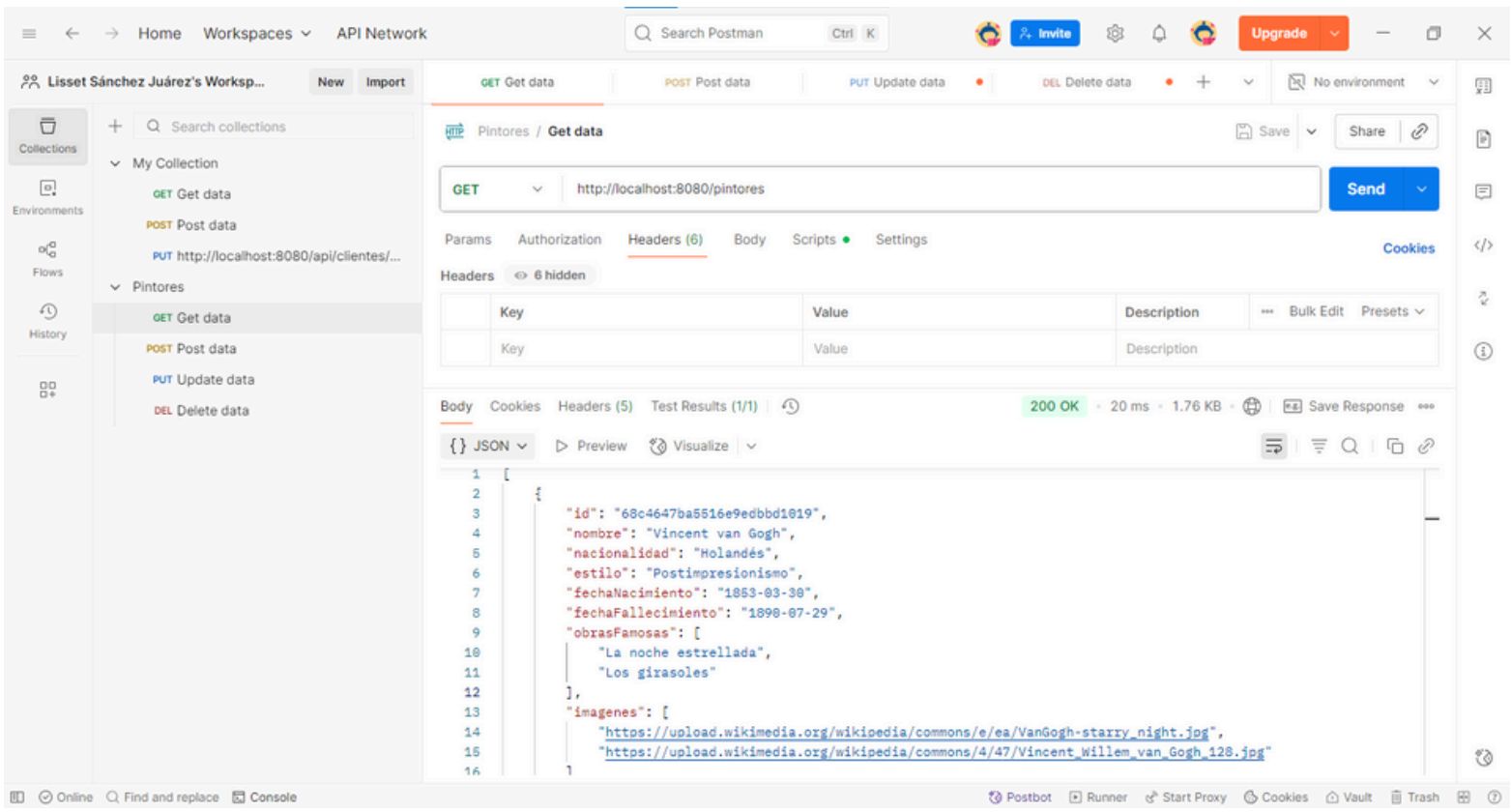
Pruebas realizadas

Utilizando Postman, probé cada una de las operaciones de mi CRUD de pintores:

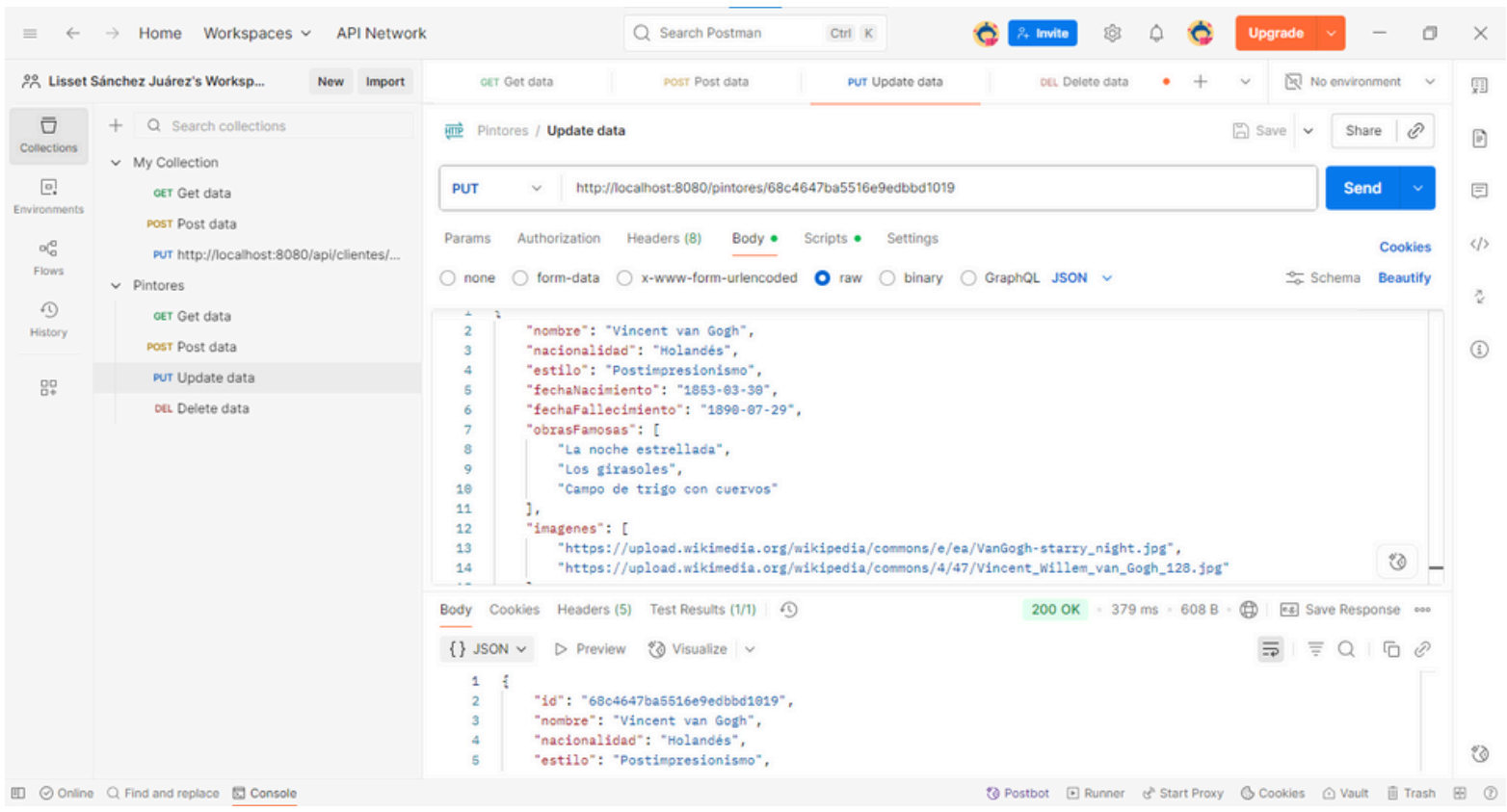
- POST /pintores → Creé varios pintores famosos, por ejemplo Pablo Picasso, Leonardo da Vinci y Frida Kahlo. En el cuerpo de la petición envié sus datos (nombre, país, estilo, fechas e imágenes).



GET /pintores → Consulté la lista completa de pintores registrados y confirmé que todos los que había creado aparecían en la colección.
GET /pintores/{id} → Busqué un pintor específico usando su ID generado automáticamente, y comprobé que la API devolvía únicamente la información de ese pintor.



PUT /pintores/{id} → Actualicé los datos de un pintor (le agregué una pintura Van Gogh) y verifiqué que los cambios se reflejaron correctamente en la base de datos.



DELETE /pintores/{id} → Eliminé un pintor por su ID y luego ejecuté un GET para comprobar que efectivamente había desaparecido de la colección.