1 DE DICIEMBRE DE 2025

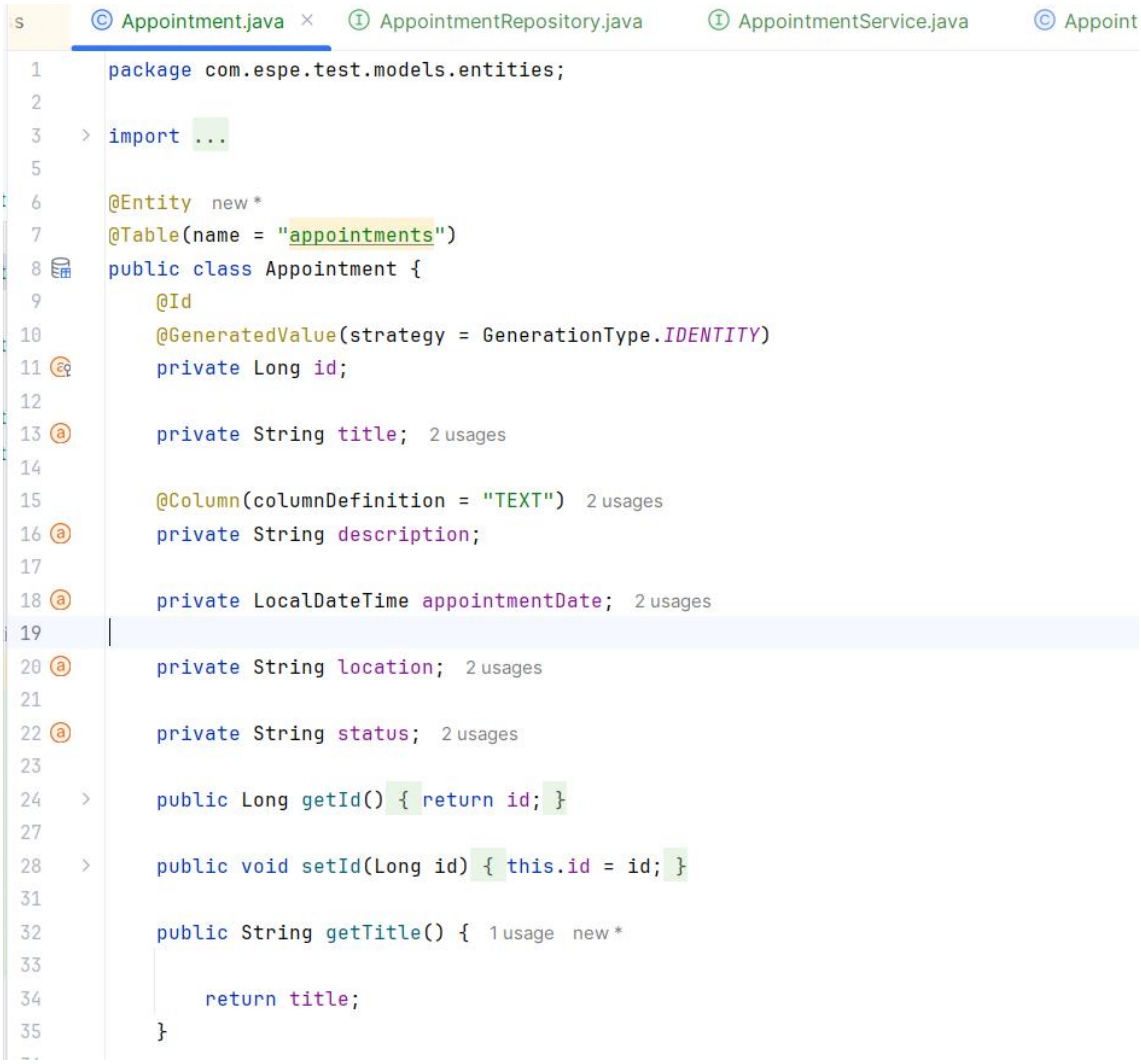
API CON DOCKER

Taller01

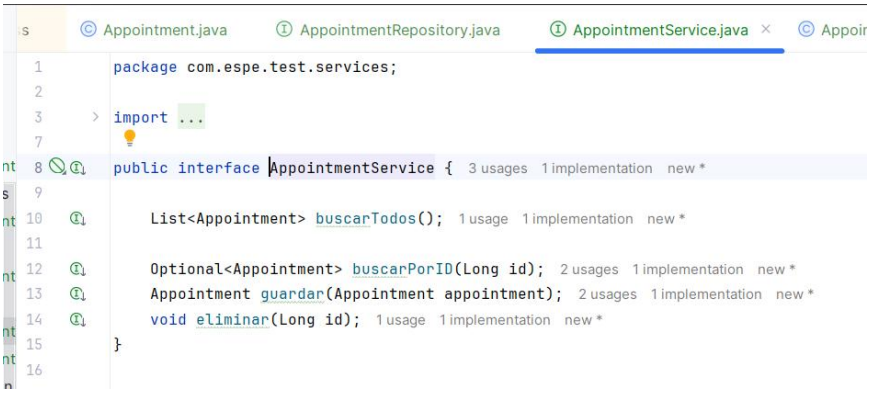
KEVIN SEBASTIAN LECHON CHURUCHUMBI
UNIVERSIDAD DE LAS FUERZAS ARMADAS “ESPE”
Sangolquí - Ecuador

Reporte Ejecutivo Técnico – API Book

Creación del modelo según la entidad escogida



Creación de service



Reglas de negocio según la entidad escogida

```java
package com.espe.test.services;

import ...

@Service  new *
public class AppointmentServiceImpl implements AppointmentService {

    @Autowired
    private AppointmentRepository repository;

    @Override  1 usage  new *
    @Transactional(readOnly = true)
    public List<Appointment> buscarTodos() { return (List<Appointment>) repository.findAll(); }

    @Override  2 usages  new *
    @Transactional(readOnly = true)
    public Optional<Appointment> buscarPorID(Long id) { return repository.findById(id); }

    @Override  2 usages  new *
    @Transactional
    public Appointment guardar(Appointment appointment) { return repository.save(appointment); }

    @Override  1 usage  new *
    @Transactional
    public void eliminar(Long id) { repository.deleteById(id); }
}
```

Creación de un repositorio

```java
package com.espe.test.repositories;

import ...

@Transactional  2 usages  new *
public interface AppointmentRepository extends CrudRepository<Appointment, Long> {

}
```

Ctrl⇧L to Cascade, Ctrl+I to Command

Creación del Controladorcita

```java
package com.espe.test.controllers;

import ...

@RestController  new *
@RequestMapping("/appointments")
public class AppointmentController {
    @Autowired
    private AppointmentService service;

    @GetMapping  new *
    public ResponseEntity<List<Appointment>> listar() { return ResponseEntity.ok(service.buscarTodos()); }

    @PostMapping  new *
    public ResponseEntity<?> crear(@RequestBody Appointment appointment){
        Appointment appointmentDB = service.guardar(appointment);
        return ResponseEntity.status(HttpStatus.CREATED).body(appointmentDB);
    }

    @PutMapping("/{id}")  new *
    public ResponseEntity<?> editar(@RequestBody Appointment appointment, @PathVariable Long id){
        Optional<Appointment> o = service.buscarPorID(id);
        if(o.isEmpty()){
            return ResponseEntity.notFound().build();
        }
        Appointment appointmentDB = o.get();
        appointmentDB.setTitle(appointment.getTitle());
        appointmentDB.setDescription(appointment.getDescription());
        appointmentDB.setAppointmentDate(appointment.getAppointmentDate());
        appointmentDB.setLocation(appointment.getLocation());
        appointmentDB.setStatus(appointment.getStatus());
        return ResponseEntity.status(HttpStatus.CREATED).body(service.guardar(appointmentDB));
```

1. Descripción general del sistema

El sistema consiste en una API RESTful desarrollada en Java 17 con Spring Boot, que gestiona la entidad Cita. Permite realizar operaciones de CRUD completo: crear, listar, buscar por ID, actualizar y eliminar libros.

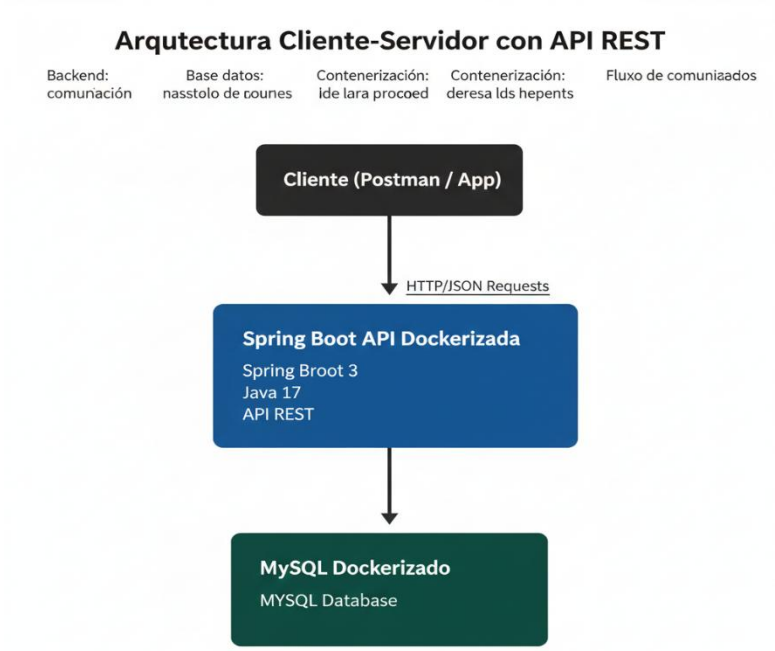
La API está conectada a una base de datos MySQL que corre dentro de un contenedor Docker, garantizando portabilidad y facilidad de despliegue.

El propósito principal es ofrecer un servicio backend que pueda integrarse con cualquier aplicación cliente, ya sea web o móvil.

2. Arquitectura utilizada

- Arquitectura: Cliente – Servidor con API REST.
- Backend: Spring Boot 3, Java 17.
- Base de datos: MySQL en contenedor Docker.
- Contenerización: Docker para la API y la base de datos, sin usar Docker Compose.
- Flujo de comunicación: Los clientes hacen peticiones HTTP (GET, POST, PUT, DELETE) y la API responde en formato JSON.

Diagrama conceptual:



3. Diseño REST aplicado

Endpoints principales implementados:

MÉTODO	URL	DESCRIPCIÓN	REQUEST BODY
GET	/appointments	Listar todas las citas	
GET	/appointments/{id}	Buscar citas por ID	
POST	/appointments	Crear un cita	
PUT	/appointments/{id}	Actualizar una cita	
DELETE	/appointments/{id}	Eliminar una cita	

Formato de respuesta: JSON, ejemplo:

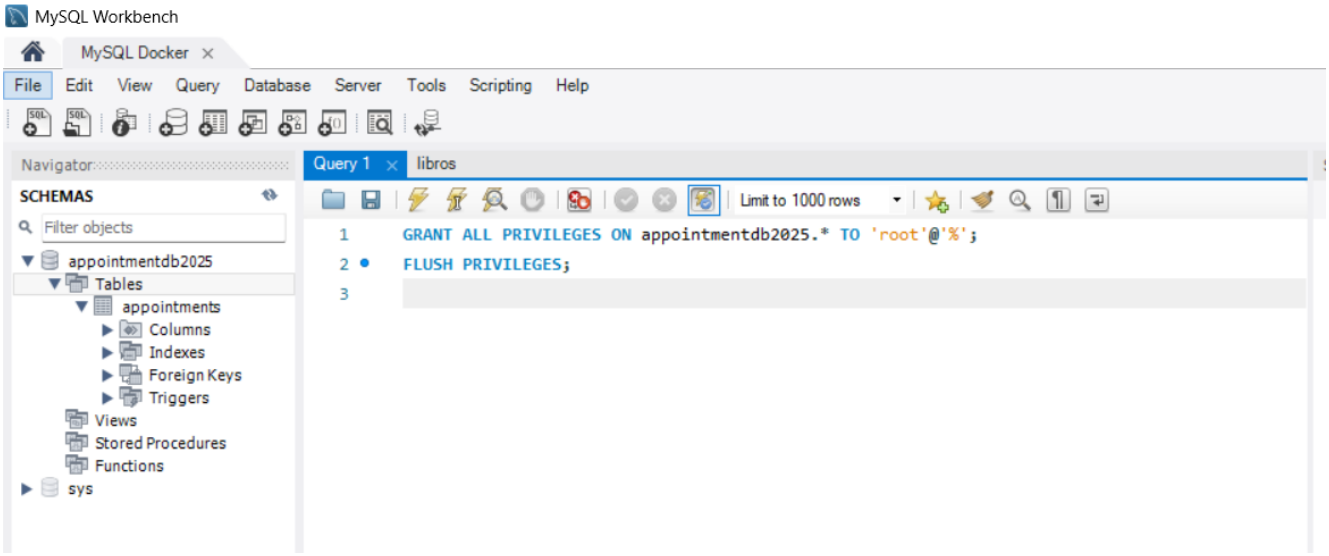
```
{
  "title": "Reunión con cliente",
  "description": "Revisión de requerimientos y planificación del proyecto",
  "appointmentDate": "2025-02-10T09:00:00",
  "location": "Oficina Central",
  "status": "Pendiente"
}
```

- **Crear la base de datos en Docker**

Levantar un contenedor MySQL

```
PS C:\Users\Usuario> docker run -d --name mysql-appointmentdb2025 -p 3307:3306 -e MYSQL_ROOT_PASSWORD=abcd -e MYSQL_DATABASE=appointmentdb2025 -
e MYSQL_USER=AppRoot -e MYSQL_PASSWORD=abcd mysql:8.0
bc05cb50247b767f8f99118c5c12766709047acd85d6b0e0fa9f3e355156e651
PS C:\Users\Usuario> docker ps
>>
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS                                              NAMES
bc05cb50247b   mysql:8.0   "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes   0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp   mysql-appointme
ntdb2025
```

Crear la base de datos requerida.



4. Entidad: Cita.java

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import java.time.LocalDateTime;

@Entity
public class Cita {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String paciente;
    private String doctor;
    private LocalDateTime fechaHora;
    private String motivo;

    // Getters y Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getPaciente() { return paciente; }
    public void setPaciente(String paciente) { this.paciente =
paciente; }

    public String getDoctor() { return doctor; }
    public void setDoctor(String doctor) { this.doctor = doctor; }

    public LocalDateTime getFechaHora() { return fechaHora; }
```

```
    public void setFechaHora(LocalDateTime fechaHora) { this.fechaHora
= fechaHora; }

    public String getMotivo() { return motivo; }
    public void setMotivo(String motivo) { this.motivo = motivo; }
}
```

Explicación:

- `@Entity`: Marca la clase como una entidad JPA, se mapeará a la tabla `cita`.
- `@Id` y `@GeneratedValue(strategy = GenerationType.IDENTITY)`: La base de datos generará automáticamente el ID.
- `paciente`, `doctor`, `fechaHora`, `motivo` → Columnas de la tabla que almacenan la información de la cita.

Repositorio: CitaRepository.java

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface CitaRepository extends JpaRepository<Cita, Long> {}
```

Explicación:

- Hereda los métodos CRUD básicos de JPA: `save()`, `findAll()`, `findById()`, `delete()`.
- `Cita`: Tipo de entidad.
- `Long`: Tipo de ID de la entidad.

Controlador: CitaController.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/citas")
public class CitaController {

    @Autowired
    private CitaRepository repository;

    // GET /citas → Lista todas las citas
    @GetMapping
    public List<Cita> getAll() {
        return repository.findAll();
    }

    // GET /appointments/{id} → Busca una cita por ID
    @GetMapping("/{id}")
    public ResponseEntity<Cita> getById(@PathVariable Long id) {
```

```
        return repository.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    // POST /citas → Crea una cita
    @PostMapping
    public Cita create(@RequestBody Cita cita) {
        return repository.save(cita);
    }

    // PUT /appointments/{id} → Actualiza una cita
    @PutMapping("/{id}")
    public ResponseEntity<Cita> update(@PathVariable Long id,
@RequestBody Cita cita) {
        return repository.findById(id)
                .map(c -> {
                    c.setPaciente(cita.getPaciente());
                    c.setDoctor(cita.getDoctor());
                    c.setFechaHora(cita.getFechaHora());
                    c.setMotivo(cita.getMotivo());
                    return ResponseEntity.ok(repository.save(c));
                })
                .orElse(ResponseEntity.notFound().build());
    }

    // DELETE /appointments/{id} → Elimina una cita
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable Long id) {
        return repository.findById(id)
                .map(c -> {
                    repository.delete(c);
                    return ResponseEntity.noContent().build();
                })
                .orElse(ResponseEntity.notFound().build());
    }
}
```

Explicación:

- `@RestController`: Maneja peticiones HTTP y devuelve JSON.
- `@RequestMapping("/appointments")`: Define la ruta base de la API.
- **Endpoints:**
    1. `GET /appointments` → Lista todas las citas.
    2. `GET /appointments/{id}` → Busca una cita por ID.
    3. `POST /appointments` → Crea una nueva cita.
    4. `PUT /appointments/{id}` → Actualiza una cita existente.
    5. `DELETE /appointments/{id}` → Elimina una cita por ID.
- Cada endpoint maneja correctamente los casos de éxito y errores (cita no encontrada).

.

1. **Evidencias de Docker**

```
[INFO] Finished at: 2025-12-01T18:28:28-05:00
[INFO] ------------------------------------------------------------------------
PS C:\Users\Usuario\Documents\Septimo\Distribuidas\SEGUNDO_P\test> docker build -t kslechon/test-api:1.0 .
[+] Building 5.8s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 379B
 => [internal] load metadata for docker.io/library/eclipse-temurin:17-jdk-alpine
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/eclipse-temurin:17-jdk-alpine@sha256:eaf56b7430cee6c93871106367715e2675192093d8f67dbbdbe07136f7cfae60
 => => resolve docker.io/library/eclipse-temurin:17-jdk-alpine@sha256:eaf56b7430cee6c93871106367715e2675192093d8f67dbbdbe07136f7cfae60
 => [internal] load build context
 => => transferring context: 53.21MB
 => CACHED [2/3] WORKDIR /app
 => [3/3] COPY target/test-0.0.1-SNAPSHOT.jar app.jar
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:06639ceca722365ae7522406d8e5994798864f21639bf0ec3f564aa2add42964
 => => exporting config sha256:441177ec4cf686872bb09ee37e500fed93f2da6406ad7fc12a50988e0f902772
 => => exporting attestation manifest sha256:3118cc685d45052cddcadf7ad3d533db968dc2d75d1c08e7aadccf5fbebe9e71
 => => exporting manifest list sha256:eee8ce8bb5df844fc1e11557aabff1333aea595e5b808677b95197fda1ef7526
 => => naming to docker.io/kslechon/test-api:1.0
 => => unpacking to docker.io/kslechon/test-api:1.0

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/yym2h9klwebktzr6pn41z69hs
PS C:\Users\Usuario\Documents\Septimo\Distribuidas\SEGUNDO_P\test>
```

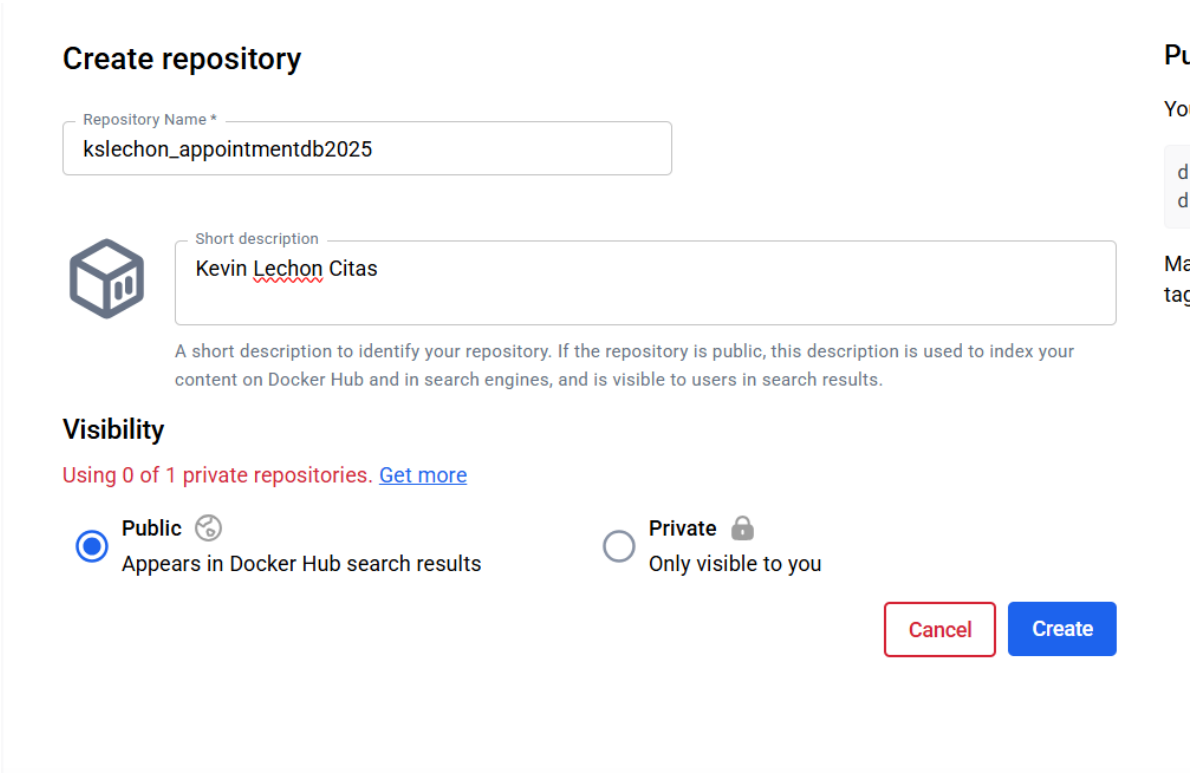
Le desplegamos nuestro proyecto en el Docker

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
☐	○ angry_elgamal	e908c4a80064	httpd	8081:80	0%	1 month ago	
☐	○ stupefied_jennings	60913c369255	nginx	8082:80	0%	1 month ago	
☐	● mysql-appointmentdb20	bc05cb50247b	mysql:8.0	3307:3306	1.41%	53 minutes ago	
☐	● test-api	e4b63e77099d	kslechon/test-api:1.0	8080:8080	0.2%	1 second ago	

Mandamos Docker ps para ver que los dos contenedores esten alzados

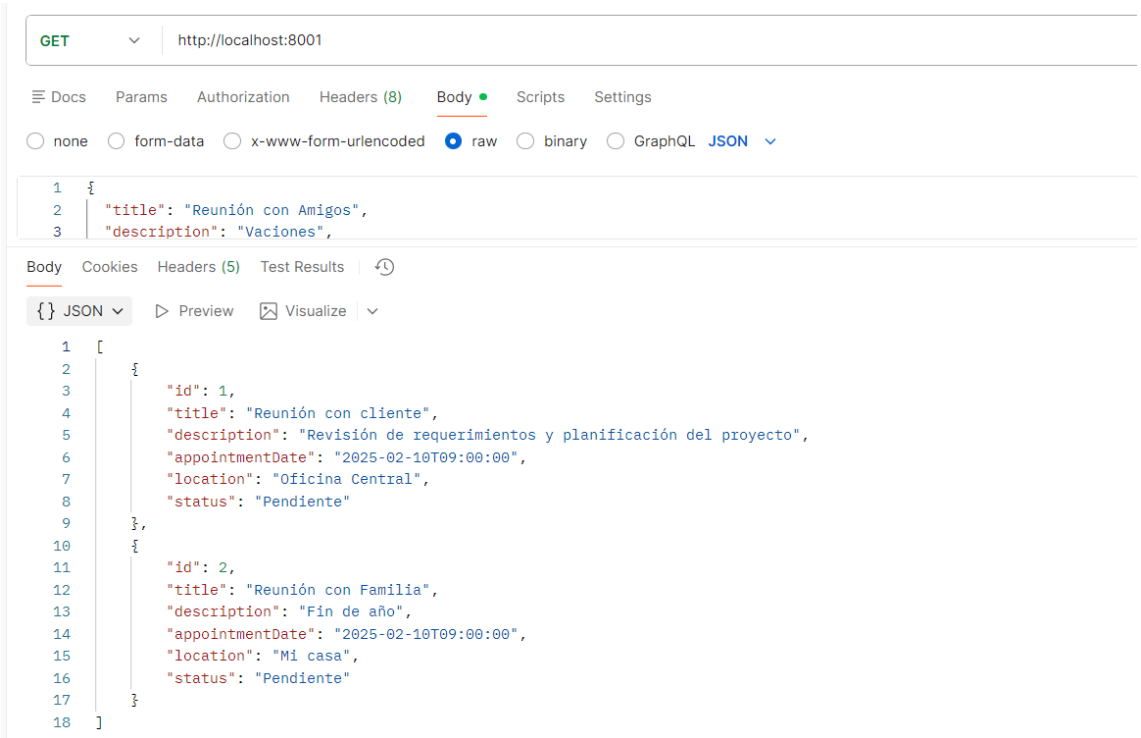
```
C:\Users\Usuario>docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED          STATUS          PORTS
                    NAMES
e4b63e77099d   kslechon/test-api:1.0     "java -jar app.jar"      2 minutes ago    Up 5 seconds    0.0.0.0:8080->8080/tcp,
 [::]:8080->8080/tcp   test-api
bc05cb50247b   mysql:8.0                 "docker-entrypoint.s…"   52 minutes ago   Up 52 minutes   0.0.0.0:3307->3306/tcp,
 [::]:3307->3306/tcp   mysql-appointmentdb2025
```

En este apartado lo que hacemos es crear un repositorio en github como podemos ver a continuación.

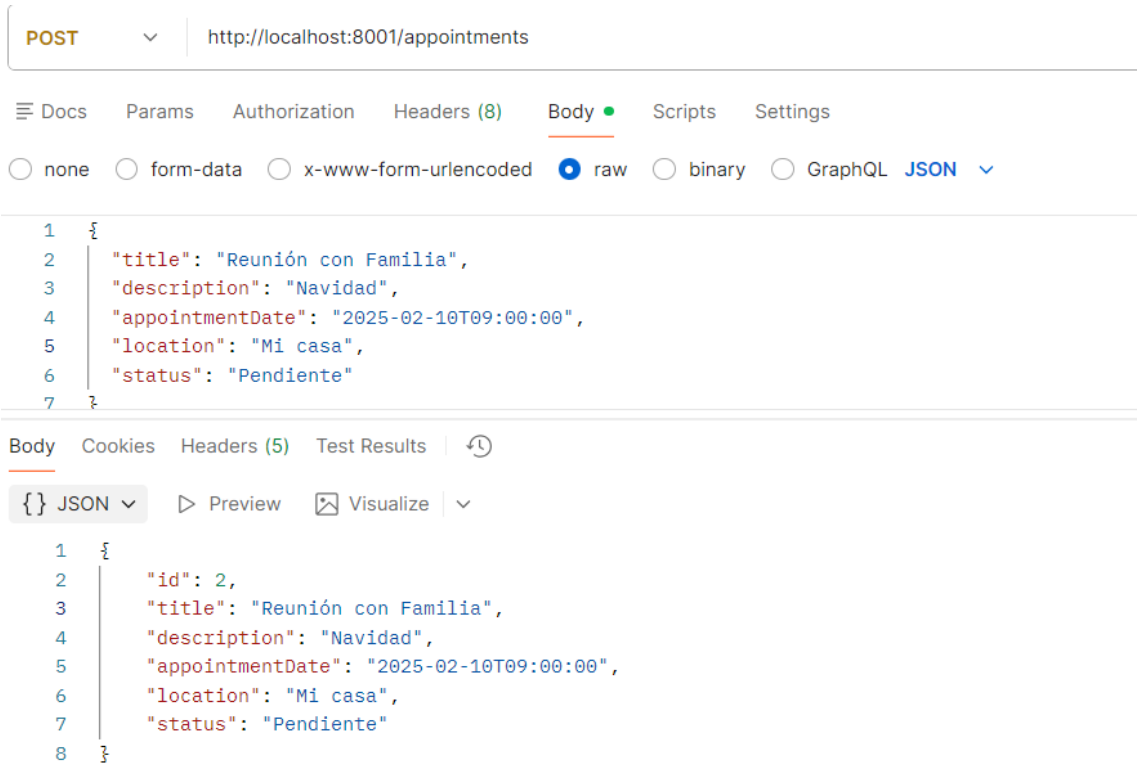


6. **Evidencias de pruebas con Postman**

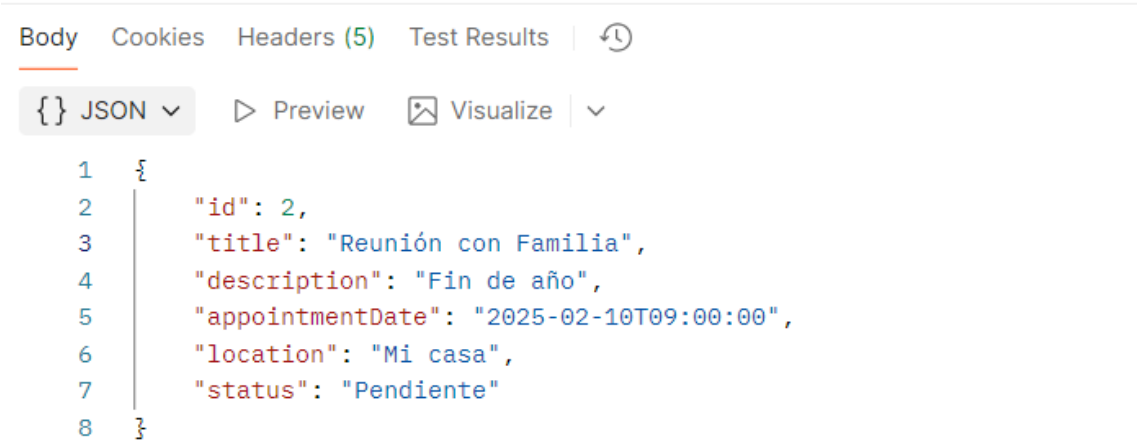
- **GET /citas:** Devuelve todos los libros.



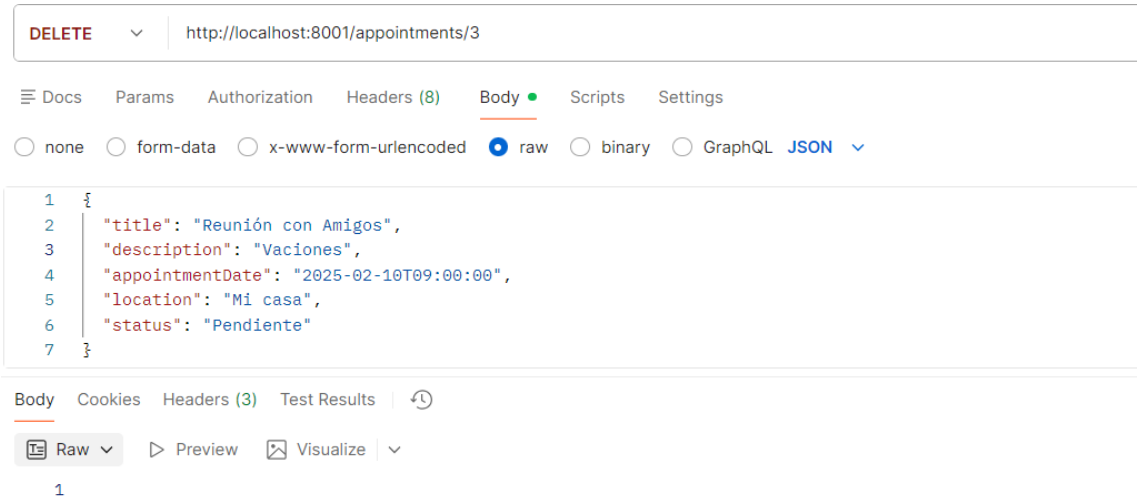
- **POST /citas:** Inserta un nuevo libro.



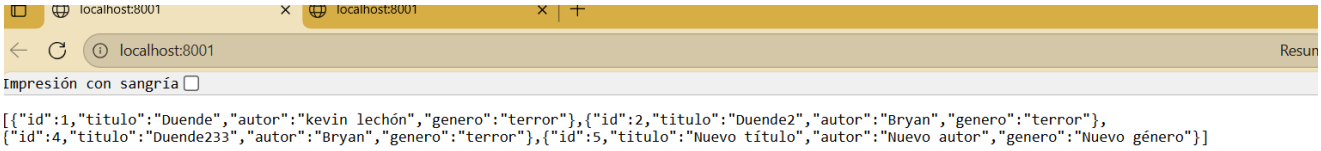
- **PUT /books/{id}:** Actualiza libro existente.



- **DELETE /books/{id}:** Elimina libro por ID.



En este apartado podemos ver nuestra Api en web



7. **Pasos para ejecutar la aplicación**

1. **Levantar MySQL:**

docker run -d --name mysql-appointmentdb2025 -p 3307:3306 -e MYSQL_ROOT_PASSWORD=abcd -e MYSQL_DATABASE=appointmentdb2025 -e MYSQL_USER=AppRoot -e MYSQL_PASSWORD=abcd mysql:8.0

2. **Construir el JAR:**

mvn clean package -DskipTests

3. **Construir imagen Docker de la API:**

docker build -t kslechon/sisdb2025-api:1.0 .

4. **Ejecutar contenedor de la API:**

docker run -d -p 8080:8080 --name test-api kslechon/test-api:1.0

Conclusiones

- La API RESTful para la entidad citas permite crear, listar, actualizar, buscar por ID y eliminar registros, cumpliendo con los principios de diseño REST y garantizando la gestión completa de los datos.
- La aplicación Spring Boot se pudo contenerizar con Docker, lo que asegura portabilidad y facilidad de despliegue en distintos entornos sin depender de configuraciones locales.
- La conexión con MySQL en contenedor Docker funcionó correctamente, y las pruebas mediante Postman permitieron verificar tanto casos exitosos como errores, asegurando la confiabilidad de la API.

Recomendaciones

- Implementar variables de entorno o un archivo de configuración seguro para las credenciales de la base de datos, evitando exponer información sensible en el código.
- Usar Swagger o OpenAPI para documentar los endpoints y versionar la API, facilitando el mantenimiento y la integración con otros sistemas.
- Incorporar pruebas unitarias y de integración automatizadas, así como herramientas de monitoreo para detectar errores o caídas del servicio en tiempo real.