

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

s Appointment.java × AppointmentRepository.java AppointmentService.java Appointment

```

1 package com.espe.test.models.entities;
2
3 > import ...
4
5
6 @Entity new *
7 @Table(name = "appointments")
8 public class Appointment {
9     @Id
10    @GeneratedValue(strategy = GenerationType.IDENTITY)
11    private Long id;
12
13    private String title; 2 usages
14
15    @Column(columnDefinition = "TEXT") 2 usages
16    private String description;
17
18    private LocalDateTime appointmentDate; 2 usages
19
20    private String location; 2 usages
21
22    private String status; 2 usages
23
24    > public Long getId() { return id; }
25
26
27
28    > public void setId(Long id) { this.id = id; }
29
30
31
32    public String getTitle() { 1 usage new *
33
34        return title;
35    }

```

Creación de service

s Appointment.java AppointmentRepository.java AppointmentService.java × Appoir

```

1 package com.espe.test.services;
2
3 > import ...
4
5
6
7
8 public interface AppointmentService { 3 usages 1 implementation new *
9
10    List<Appointment> buscarTodos(); 1 usage 1 implementation new *
11
12    Optional<Appointment> buscarPorID(Long id); 2 usages 1 implementation new *
13    Appointment guardar(Appointment appointment); 2 usages 1 implementation new *
14    void eliminar(Long id); 1 usage 1 implementation new *
15 }
16

```

Reglas de negocio según la entidad escogida

```
1 package com.espe.test.services;
2
3 > import ...
11
12 @Service new *
13 public class AppointmentServiceImpl implements AppointmentService {
14
15     @Autowired
16     private AppointmentRepository repository;
17
18     @Override 1 usage new *
19     @Transactional(readOnly = true)
20     public List<Appointment> buscarTodos() { return (List<Appointment>) repository.findAll(); }
23
24     @Override 2 usages new *
25     @Transactional(readOnly = true)
26     public Optional<Appointment> buscarPorID(Long id) { return repository.findById(id); }
29
30     @Override 2 usages new *
31     @Transactional
32     public Appointment guardar(Appointment appointment) { return repository.save(appointment); }
35
36     @Override 1 usage new *
37     @Transactional
38     public void eliminar(Long id) { repository.deleteById(id); }
41 }
42
43
```

Creación de un repositorio

```
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 Controlador

```

1 package com.espe.test.controllers;
2
3 > import ...
12
13 @RestController new *
14 @RequestMapping("/appointments")
15 public class AppointmentController {
16     @Autowired
17     private AppointmentService service;
18
19     @GetMapping new *
20     public ResponseEntity<List<Appointment>> listar() { return ResponseEntity.ok(service.buscarTodos()); }
21
22     @PostMapping new *
23     public ResponseEntity<?> crear(@RequestBody Appointment appointment){
24         Appointment appointmentDB = service.guardar(appointment);
25         return ResponseEntity.status(HttpStatus.CREATED).body(appointmentDB);
26     }
27
28     @PutMapping("/{id}") new *
29     public ResponseEntity<?> editar(@RequestBody Appointment appointment, @PathVariable Long id){
30         Optional<Appointment> o = service.buscarPorID(id);
31         if(o.isEmpty()){
32             return ResponseEntity.notFound().build();
33         }
34         Appointment appointmentDB = o.get();
35         appointmentDB.setTitle(appointment.getTitle());
36         appointmentDB.setDescription(appointment.getDescription());
37         appointmentDB.setAppointmentDate(appointment.getAppointmentDate());
38         appointmentDB.setLocation(appointment.getLocation());
39         appointmentDB.setStatus(appointment.getStatus());
40         return ResponseEntity.status(HttpStatus.CREATED).body(service.guardar(appointmentDB));
41     }
42

```

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.

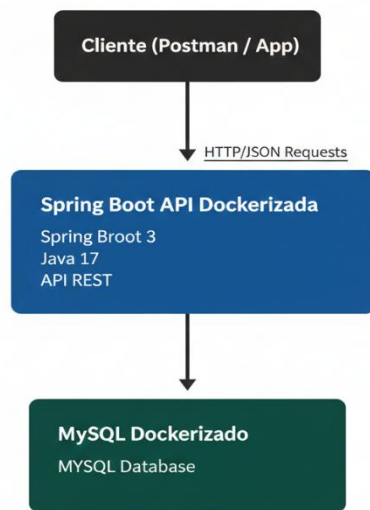
2. Arquitectura utilizada

- 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:

Arquitectura Cliente-Servidor con API REST

Backend: comunicación Base datos: nasstolo de rounes Contenerización: ide lara procoed Contenerización: deresa lds hepents Flujo de comunicados



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:

```

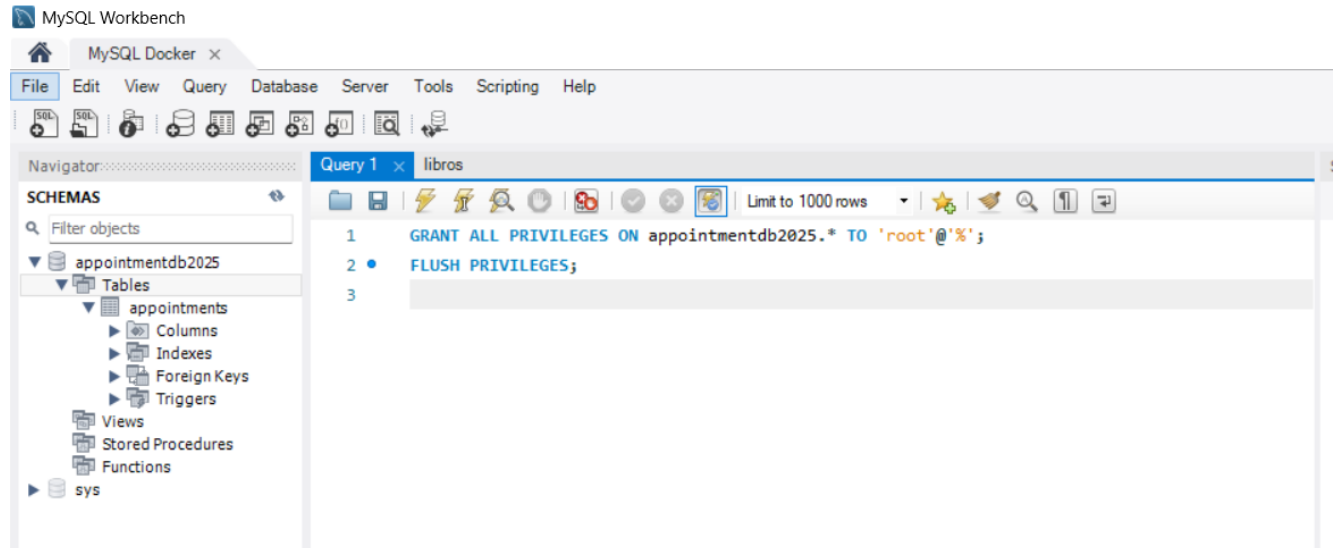
{
  "title": "Reunión con cliente",
  "description": "Revisión de requerimientos y planificación del proyecto",
  "appointmentDate": "2025-02-10T09:00:00",
  "location": "Oficina Central",
  "status": "Pendiente"
}
  
```

- 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 kslachon/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:eaf56b7430cee6c93871106367715e2675192093d8f67dbbde07136f7cfae60
=> => resolve docker.io/library/eclipse-temurin:17-jdk-alpine@sha256:eaf56b7430cee6c93871106367715e2675192093d8f67dbbde07136f7cfae60
=> [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 sha256:3118cc685d45052cddcadf7ad3d533db968dc2d75d1c08e7aadccf5f8be9e71
=> => exporting manifest list sha256:eee8ce8bb5df844fc1e11557aabff1333aea595e5b808677b95197fda1ef7526
=> => naming to docker.io/kslachon/test-api:1.0
=> => unpacking to docker.io/kslachon/test-api:1.0
```

View build details: [docker-desktop://dashboard/build/desktop-linux/desktop-linux/yym2h9Kwebktr6pn41z69hs](#)
PS C:\Users\Usuario\Documents\Septimo\Distribuidas\SEGUNDO_P\test>

Le desplegamos nuestro proyecto en el Docker

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	angry_elgamal	e908c4a80064	httpd	8081:80	0%	1 month ago	
<input type="checkbox"/>	stupefied_jennings	60913c369255	nginx	8082:80	0%	1 month ago	
<input type="checkbox"/>	mysql-appointmentdb20	bc05cb50247b	mysql:8.0	3307:3306	1.41%	53 minutes ago	
<input type="checkbox"/>	test-api	e4b63e77099d	kslachon/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          NAMES               COMMAND                  CREATED        STATUS        PORTS
e4b63e77099d   kslachon/test-api:1.0   test-api            "java -jar app.jar"     2 minutes ago   Up 5 seconds   0.0.0.0:8080->8080/tcp,
[::]:8080->8080/tcp
bc05cb50247b   mysql:8.0       mysql-appointmentdb205  "docker-entrypoint.s..." 52 minutes ago   Up 52 minutes   0.0.0.0:3307->3306/tcp,
[::]:3307->3306/tcp
```

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

Create repository

Repository Name *
kslechon_appointmentdb2025

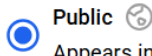


Short description
Kevin Lechon Citas

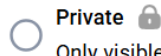
A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. [Get more](#)



Public
Appears in Docker Hub search results



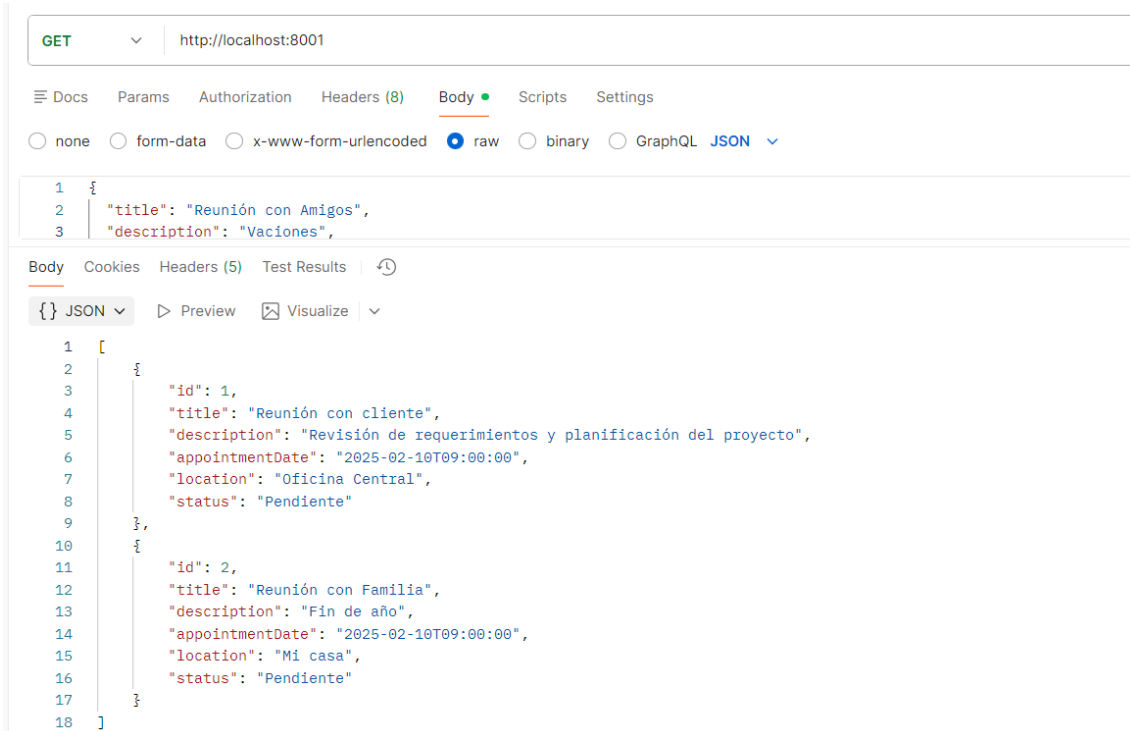
Private
Only visible to you

Cancel

Create

6. Evidencias de pruebas con Postman

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



GET http://localhost:8001

Docs Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "title": "Reunión con Amigos",
3   "description": "Vacaciones",

```

Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```

1 [
2   {
3     "id": 1,
4     "title": "Reunión con cliente",
5     "description": "Revisión de requerimientos y planificación del proyecto",
6     "appointmentDate": "2025-02-10T09:00:00",
7     "location": "Oficina Central",
8     "status": "Pendiente"
9   },
10  {
11    "id": 2,
12    "title": "Reunión con Familia",
13    "description": "Fin de año",
14    "appointmentDate": "2025-02-10T09:00:00",
15    "location": "Mi casa",
16    "status": "Pendiente"
17  }
18 ]

```

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

POST http://localhost:8001/appointments

Docs Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1  {
2    "title": "Reunión con Familia",
3    "description": "Navidad",
4    "appointmentDate": "2025-02-10T09:00:00",
5    "location": "Mi casa",
6    "status": "Pendiente"
7  }
```

Body Cookies Headers (5) Test Results

JSON Preview Visualize

```
1  {
2    "id": 2,
3    "title": "Reunión con Familia",
4    "description": "Navidad",
5    "appointmentDate": "2025-02-10T09:00:00",
6    "location": "Mi casa",
7    "status": "Pendiente"
8  }
```

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

PUT ▼ http://localhost:8001/appointments/2

Docs Params Authorization Headers (8) **Body** ● Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```

1  {
2    "title": "Reunión con Familia",
3    "description": "Fin de año",
4    "appointmentDate": "2025-02-10T09:00:00",
5    "location": "Mi casa",
6    "status": "Pendiente"
7  }

```

Body Cookies Headers (5) Test Results ↺

{} JSON ▼ ▶ Preview 🖼 Visualize ▼

```

1  {
2    "id": 2,
3    "title": "Reunión con Familia",
4    "description": "Fin de año",
5    "appointmentDate": "2025-02-10T09:00:00",
6    "location": "Mi casa",
7    "status": "Pendiente"
8  }

```

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

DELETE ▼ http://localhost:8001/appointments/3

Docs Params Authorization Headers (8) **Body** ● Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```

1  {
2    "title": "Reunión con Amigos",
3    "description": "Vacaciones",
4    "appointmentDate": "2025-02-10T09:00:00",
5    "location": "Mi casa",
6    "status": "Pendiente"
7  }

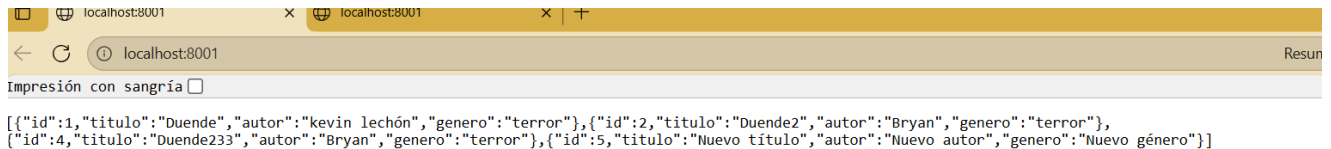
```

Body Cookies Headers (3) Test Results ↺

Raw ▼ ▶ Preview 🖼 Visualize ▼

1

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.

