Desarrollo web Integrado(GRUPO3)

TEMA: Gestión de Citas Medicas Hospital

Llacsahuanga Abad Carlos Eduardo-U22216426

Palacios Zapata Rodrigo - u22244605

Integrantes:

Docente: Richard Doann ASENCIOS VILCHEZ

Salón:Sección **37619**

Versión 1.0

Marzo 2025

# Historial de Versiones

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **VERSIÓN** | | **PARTES QUE**  **CAMBIAN** | **DESCRIPCIÓN DEL CAMBIO** | **FECHA DE CAMBIO** | **MODIFICADO POR** | **APROBADO POR** |
| 1.0 | | Descripción | AVANCE 1 | 24/04/2025 | Grupo |  |
|  |
|  |
|  | |  |  |  |  |  |

# 

# Índice

[Historial de Versiones 1](#_Toc196420616)

[Índice 2](#_Toc196420617)

[1 Definición de mi proyecto 4](#_Toc196420618)

[1.1 Nombre del Proyecto 4](#_Toc196420619)

[1.2 Breve Descripción de la Aplicación 4](#_Toc196420620)

[1.3 Justificación de la Elección del Problema 4](#_Toc196420621)

[1.4 Actores del Sistema 4](#_Toc196420622)

[1.5 Módulos Planificados 5](#_Toc196420623)

[1.6 Diagrama de Clases del Proyecto 5](#_Toc196420624)

[2 Identificación del Problema 5](#_Toc196420625)

[2.1 Proyecto generado con Spring Initializr. 5](#_Toc196420626)

[2.2 Configuración del entorno (dependencias necesarias: Spring Web, Spring Boot DevTools, etc.). 5](#_Toc196420627)

[2.3 Organización del código en paquetes (por ejemplo: controller, service, model, repository, dto). 6](#_Toc196420628)

[3 Modelo de dominio 7](#_Toc196420629)

[3.1 Entidades Principales: 7](#_Toc196420630)

[3.2 Relaciones de las Clases: 9](#_Toc196420631)

[4 Endpoints implementados (mínimo funcional) 9](#_Toc196420632)

[4.1 Medico 9](#_Toc196420633)

[**4.1.1** **Listar Todos los Medicos** 9](#_Toc196420634)

[**4.1.2** **Registrar un nuevo médico** 9](#_Toc196420635)

[**4.1.3** **Buscar médico por ID** 10](#_Toc196420636)

[**4.1.4** **Buscar médico por CMP** 11](#_Toc196420637)

[**4.1.5** **Actualizar un médico** 12](#_Toc196420638)

[**4.1.6** **Eliminar un medico** 12](#_Toc196420639)

[4.1.7 Buscar Por Nombre: 13](#_Toc196420640)

[13](#_Toc196420641)

[5 Uso de Test-Driven Development TDD 13](#_Toc196420642)

[5.1 Estrategia TDD aplicada 13](#_Toc196420643)

# Definición de mi proyecto

## Nombre del Proyecto

Gestión de Citas Médicas en un Hospital

## Breve Descripción de la Aplicación

Este proyecto consiste en el desarrollo de una aplicación web que permite gestionar citas médicas en un hospital. A través de esta plataforma, los pacientes podrán registrarse, solicitar citas médicas y visualizar sus datos personales, mientras que los médicos podrán consultar sus citas asignadas y actualizar su información profesional. Además, los administradores podrán supervisar la información general del sistema y gestionar tanto pacientes como médicos.

La aplicación ha sido construida utilizando **Spring Boot** como framework principal, aplicando buenas prácticas de desarrollo backend, integración de bases de datos y pruebas con herramientas como **JUnit** y **Postman**.

## Justificación de la Elección del Problema

En muchos centros hospitalarios, la gestión manual de citas médicas genera demoras, errores y dificultades en la atención oportuna de los pacientes. La falta de automatización también dificulta la supervisión de la disponibilidad de médicos, historial de atención y organización interna.

También que este proyecto nuestro profesor lo ha elegido para desarrollarlo a lo largo del curso

## Actores del Sistema

* **Paciente:** Usuario que accede al sistema para registrarse, solicitar citas y revisar su historial médico.
* **Médico:** Profesional de salud que gestiona su información, especialidad y visualiza las citas asignadas.
* **Administrador:** Encargado de gestionar usuarios (pacientes y médicos), verificar citas programadas y mantener la integridad del sistema.
* Cita Medica : Son donde el medico atiende a los pacientes
* Sala de atención: El lugar reservado donde los médicos harán su trabajo

## Módulos Planificados

* Módulo de Registro de Pacientes
* Módulo de Registro de Médicos
* Módulo de Gestión de Citas Médicas
* Módulo de Visualización de Historial por parte del paciente
* Módulo de Administración del sistema(No agregado)por mientras

## Diagrama de Clases del Proyecto

Diagrama

El contenido generado por IA puede ser incorrecto.

# Identificación del Problema

## Proyecto generado con Spring Initializr.

Hemos generado un Proyecto con **Java 21, Maven**: Generado usando Spring Initializr seleccionando Packaging “Jar” y Java 21, se configura el groupId como com.hospital.citasmedicas y el artefacto como citasmedicas.

## Configuración del entorno (dependencias necesarias: Spring Web, Spring Boot DevTools, etc.).

Dependecias del Proyecto:

* **spring-boot-starter-data-jpa**  
  Proporciona Spring Data JPA, que facilita la implementación de repositorios sobre bases de datos relacionales usando Hibernate como proveedor por defecto.
* **spring-boot-starter-thymeleaf**  
  Incluye el motor de plantillas Thymeleaf para generar vistas HTML dinámicas integradas con datos del backend.
* **spring-boot-starter-web**  
  Monta un servidor web embebido (Tomcat) y añade Spring MVC, RestControllers, validación, conversión JSON/HTTP, etc.
* **spring-boot-devtools**  
  Herramientas de desarrollo en caliente (hot-reload), recarga automática de beans y reinicio rápido del contexto para acelerar tu ciclo de desarrollo.
* **com.h2database:h2**  
  Base de datos en memoria para pruebas o entornos de desarrollo ligeros. No requiere configuración externa y resetea datos al reiniciar la app.
* **com.mysql:mysql-connector-j**  
  Driver JDBC para conectar tu aplicación Spring Boot con MySQL (usualmente en entornos de prueba o producción).
* **org.projectlombok:lombok**  
  Biblioteca que reduce el código repetitivo (getters/setters, constructores) mediante anotaciones. Opcional en tiempo de compilación.
* **spring-boot-starter-test**  
  Incluye JUnit, Mockito, AssertJ y otras utilidades para escribir y ejecutar tests unitarios e integrados sobre tu aplicación.

## Organización del código en paquetes (por ejemplo: controller, service, model, repository, dto).

El proyecto sigue el patrón **MVC** y la separación en capas:

com.hospital.citasmedicas

├─ controller // “Capa web”: Reciben peticiones HTTP, validan y delegan al servicio.

│ ├ CitaMedicaController.java

│ ├ PacienteController.java

│ ├ MedicoController.java

│ ├ HomeController.java

│ └ SalaController.java

│

├─ service // “Capa de negocio”: Contiene la lógica de la aplicación.

│ ├ CitaMedicaService.java

│ ├ PacienteService.java

│ └ MedicoService.java

│

├─ repository // “Capa de persistencia”: Interfaces que extienden JpaRepository.

│ ├ CitaMedicaRepository.java

│ ├ PacienteRepository.java

│ ├ MedicoRepository.java

│ └ SalaRepository.java

│

├─ model // Entidades JPA: mapeo de tablas de la base de datos.

│ ├ CitaMedica.java

│ ├ Paciente.java

│ ├ Medico.java

│ └ Sala.java

│

├─ dto // Objetos de transferencia (DTO): para recibir/enviar datos sin exponer entidades.

│ ├ CitaMedicaDTO.java

│ ├ PacienteDTO.java

│ └ MedicoDto.java

│

└─ CitasmedicasApplication.java // Clase principal con `@SpringBootApplication`.

# Modelo de dominio

## Entidades Principales:

Todas las siguientes clases están anotadas con @Entity y representan las tablas de la base de datos:

* **Paciente**
  + Atributos:
    - id (Long, PK)
    - nombre (String)
    - apellidos (String)
    - dni (String, único)
    - genero (Enum: MASCULINO, FEMENINO, NO\_ESPECIFICADO)
    - email (String, único)
    - direccion (String)
    - telefono (String)
    - gravedadAfeccion (Enum: ASINTOMATICA, LEVE, MODERADA, GRAVE, CRITICA)
    - fechaNacimiento (LocalDate)
    - fechaIngreso (LocalDateTime)
  + Relación:
    - @OneToMany(mappedBy="paciente") List<CitaMedica> citas
* **Médico**
  + Atributos:
    - id (Long, PK)
    - nombre, apellidos (String)
    - dni (String, único)
    - email (String, único)
    - especialidad (Enum: CARDIOLOGIA, CIRUGIA, …)
    - fechaGraduacion, fechaIncorporacion (LocalDate)
    - cmp (String, único)
  + Relación:
    - @OneToMany(mappedBy="medico") List<CitaMedica> citas
* **Sala**
  + Atributos:
    - id (Long, PK)
    - numeroSala (Integer, único)
  + Relación:
    - @OneToMany(mappedBy="sala") List<CitaMedica> citas
* **CitaMedica**
  + Atributos:
    - id (Long, PK)
    - fecha (LocalDate)
    - hora (LocalTime)
    - motivo (String)
    - estado (String)

## Relaciones de las Clases:

* **Un Paciente puede tener muchas Citas Médicas**
* **Un Médico puede tener muchas Citas Médicas**
* **Una Sala puede albergar muchas Citas Médicas**
* Cada **Cita Médica** está asociada obligatoriamente a un único Paciente, un único Médico y (opcionalmente) a una Sala

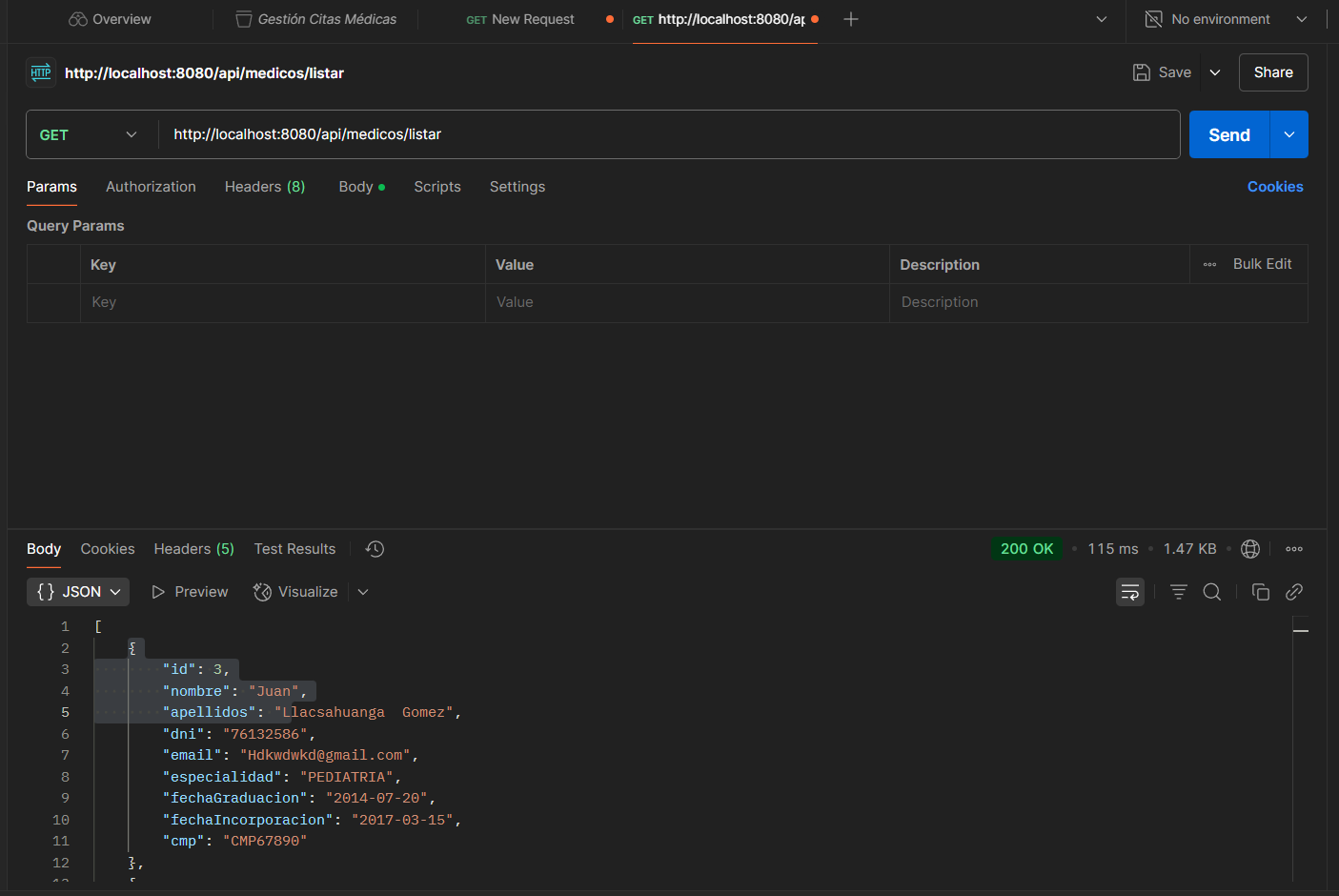
# Endpoints implementados (mínimo funcional)

## Medico

### **Listar Todos los Medicos**

**Request**

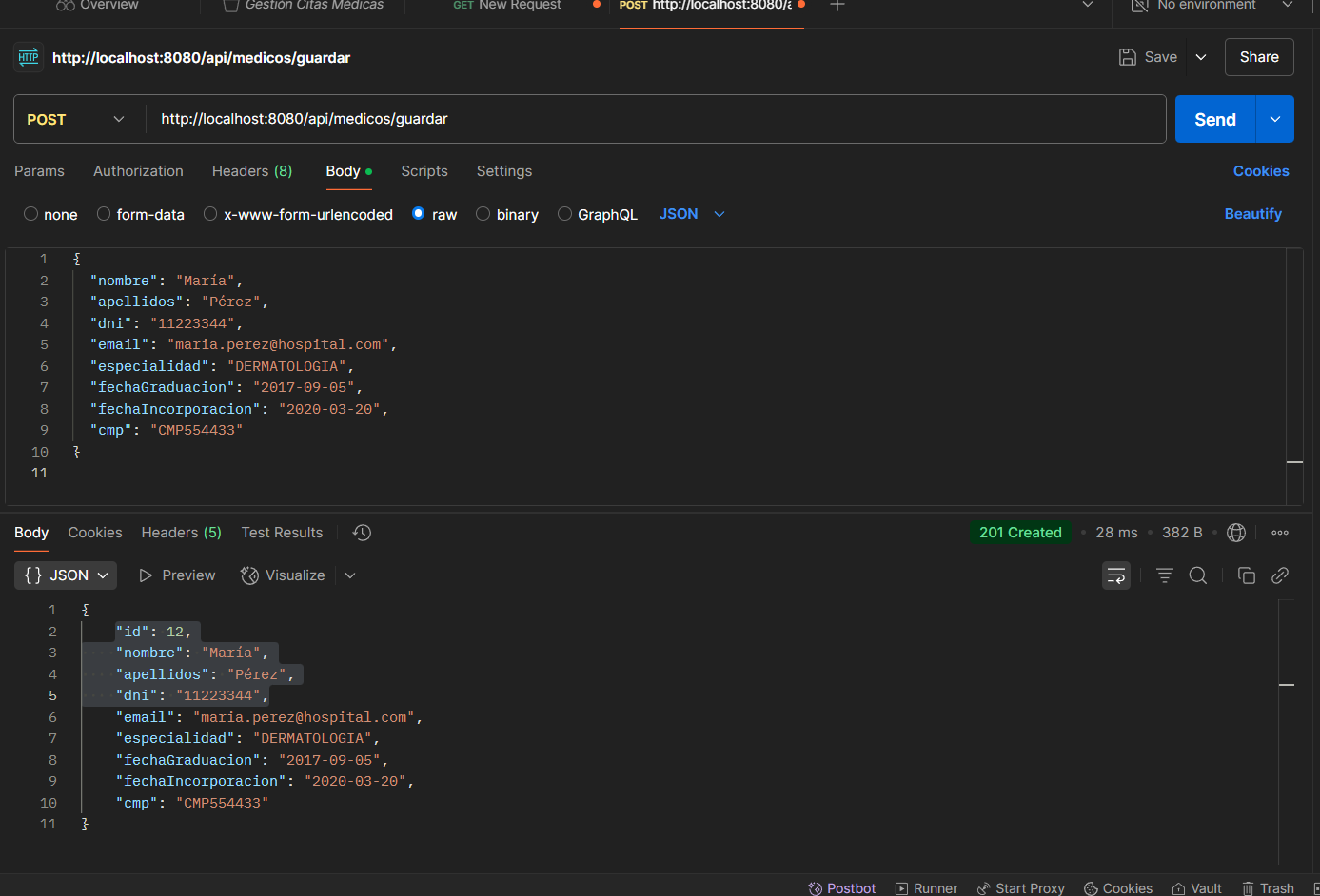
* **Método**: GET
* **URL**: http://localhost:8080/api/medicos/listar
* **Headers**:
  + Accept: application/json



### **Registrar un nuevo médico**

**Request**

* **Método**: POST
* **URL**: http://localhost:8080/api/medicos/guardar
* **Headers**:
  + Content-Type: application/json
  + Accept: application/json
* **Body** (raw JSON):



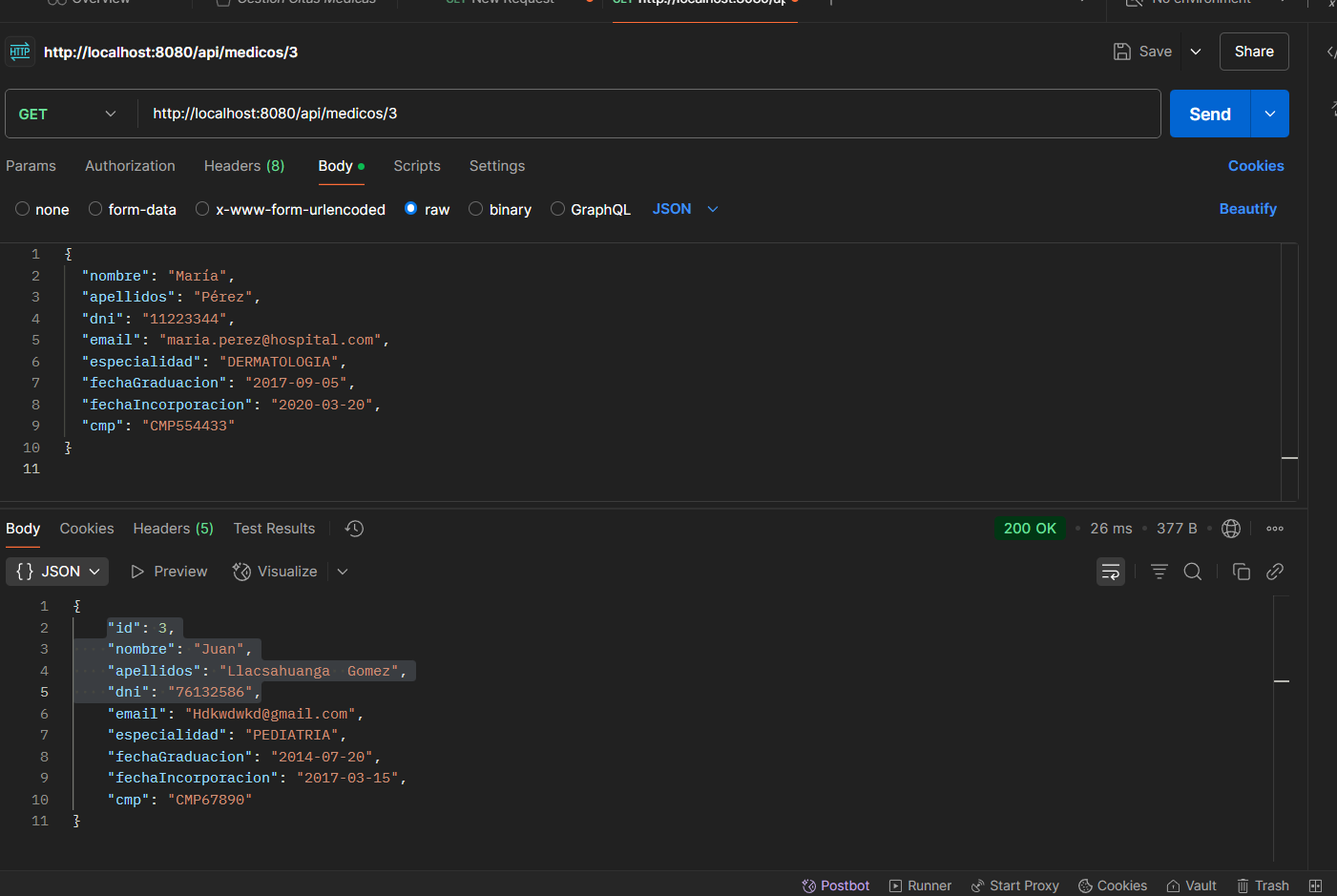
### **Buscar médico por ID**

**Método**: GET

**URL**: http://localhost:8080/api/medicos/3

**Headers**:

* Accept: application/json



### **Buscar médico por CMP**

**Método**: GET

**URL**: http://localhost:8080/api/medicos/cmp/CMP554433

**Headers**:

* Accept: application/json

Texto

El contenido generado por IA puede ser incorrecto.

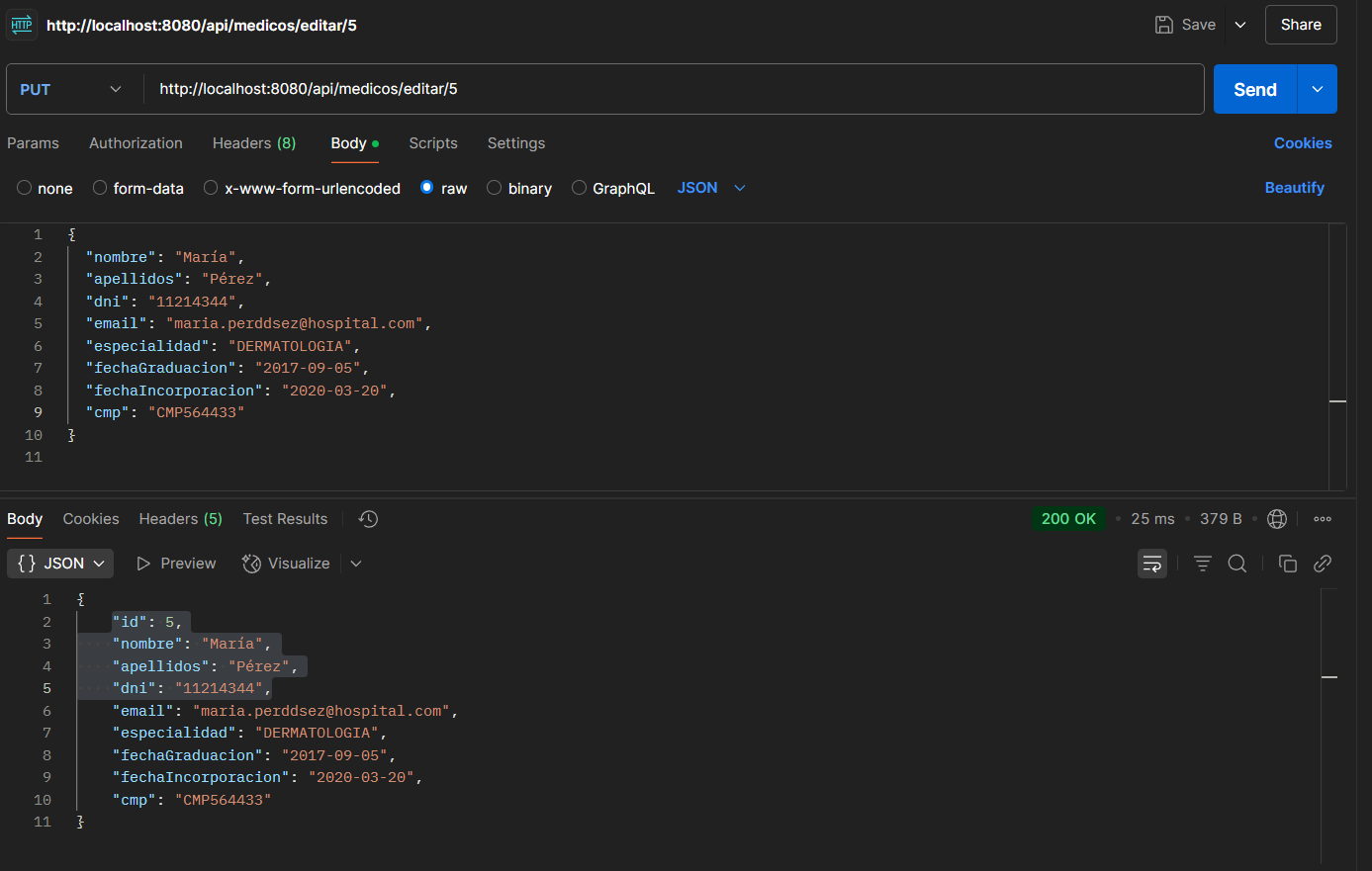
### **Actualizar un médico**

**Método**: PUT

**URL**: http://localhost:8080/api/medicos/editar/3

**Headers**:

* Content-Type: application/json
* Accept: application/json



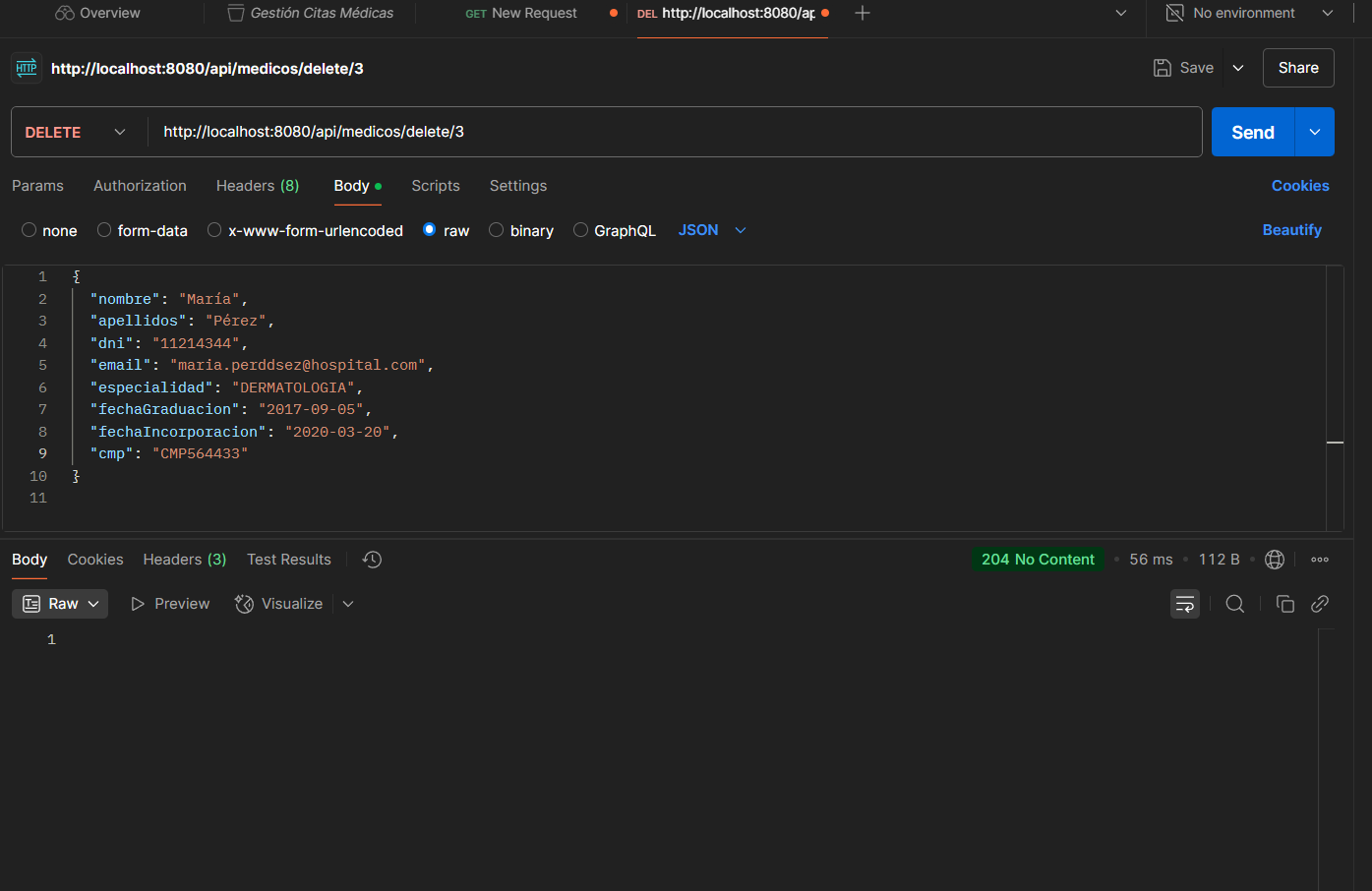
### **Eliminar un medico**

**Método**: DELETE

**URL**: http://localhost:8080/api/medicos/delete/3

**Headers**:

* Accept: application/json



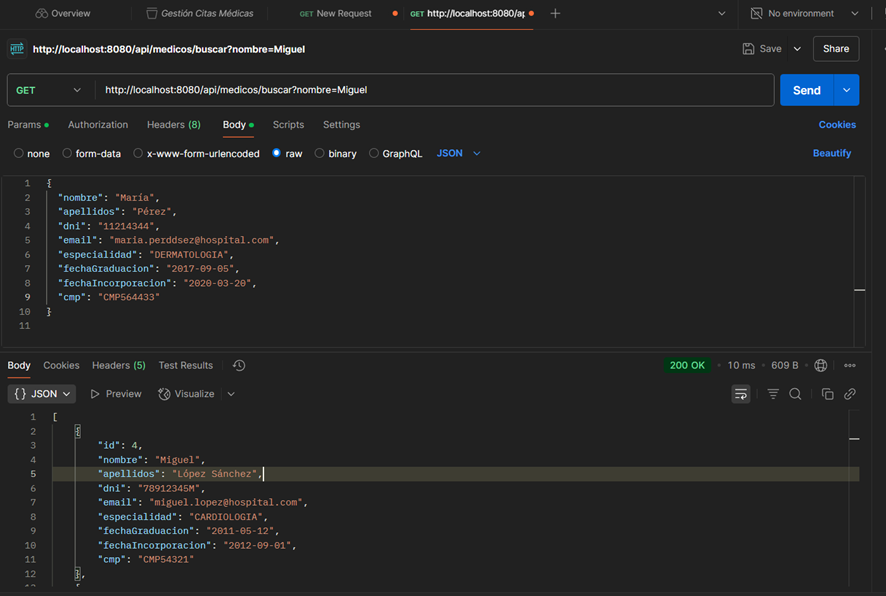
### Buscar Por Nombre:

**Método**: GET

**URL**: http://localhost:8080/api/medicos/buscar?nombre=Miguel

**Headers**:

* Accept: application/json



Y así con las demás clases..

Acá hemos usado el Programa Postman que nos permite:  
 Enviar solicitudes a servidores web y recibir respuestas, sin necesidad de escribir código complejo. En esencia, facilita la interacción con APIs, permitiendo probar, documentar y automatizar pruebas.

# Uso de Test-Driven Development TDD

## Estrategia TDD aplicada

1. **Red**: escribimos primero la interfaz de la prueba unitarias en CitaMedicaServiceTest, definiendo el comportamiento deseado (p. ej. que no acepte agendar citas con fecha anterior al día de hoy).
2. **Green**: ejecutamos la prueba en rojo, luego implementamos rápidamente la lógica mínima en CitaMedicaService para pasar el test (lanzar una excepción si la fecha es pasada).
3. **Refactor**: revisamos el código, extraemos métodos comunes (p. ej. validaciones de fecha) y limpiamos duplicaciones.