

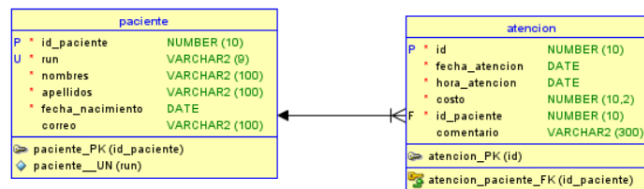
# Guía práctica HOSPITAL V&M

## PARTE 3

### Repository, Service y Controller

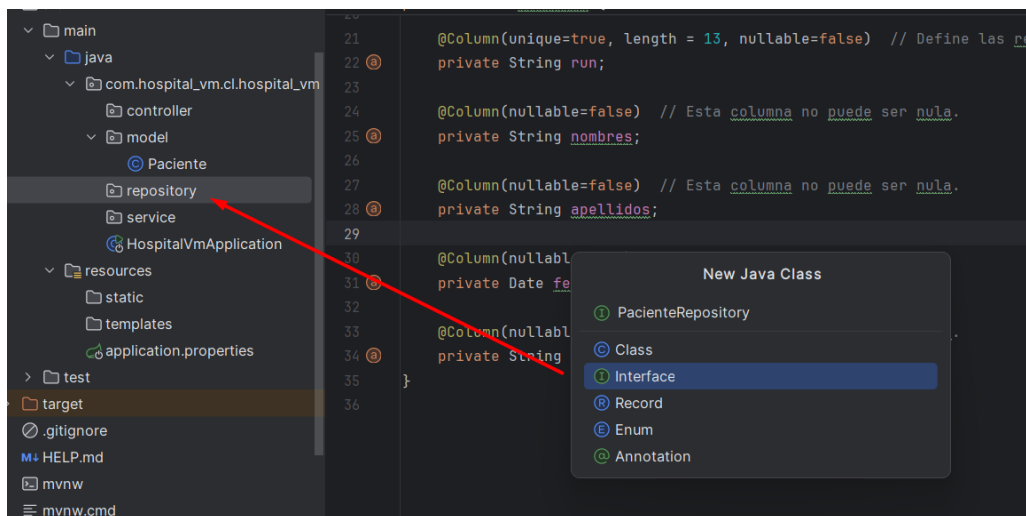


En esta guía, detallaremos paso a paso el desarrollo de un proyecto en Spring llamado "Hospital V&M". A continuación, se describen los pasos para desarrollar el repositorio, servicios y controladores en Spring Boot.



#### Paso 1: En la carpeta repository

- Crear una **interface** llamada **PacienteRepository**



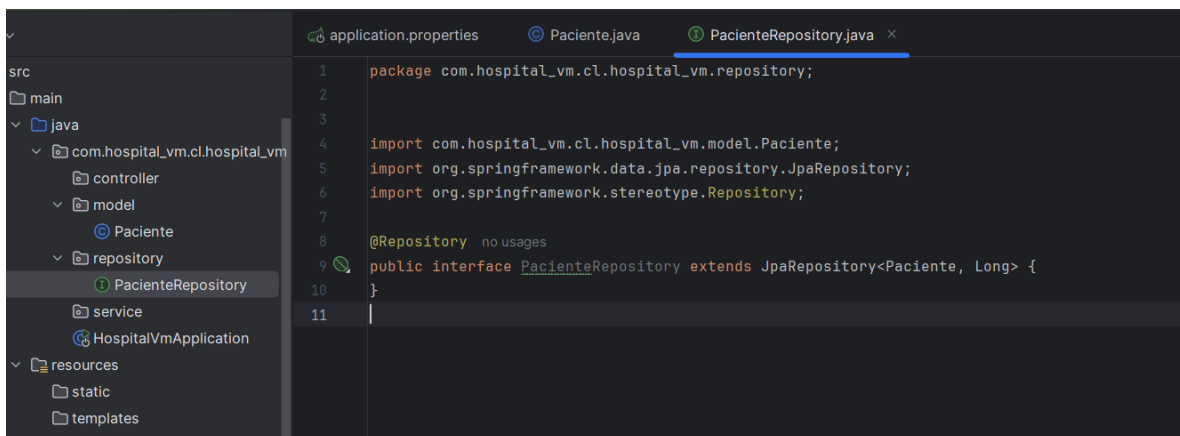


```

1 package com.hospital_vm.cl.hospital_vm.repository;
2
3 public interface PacienteRepository { no usages
4 }
5

```

Añadir lo siguiente en la interface **PacienteRepository**



```

1 package com.hospital_vm.cl.hospital_vm.repository;
2
3
4 import com.hospital_vm.cl.hospital_vm.model.Paciente;
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.stereotype.Repository;
7
8 @Repository no usages
9 public interface PacienteRepository extends JpaRepository<Paciente, Long> {
10 }
11

```

## ¿Qué es JpaRepository?

**JpaRepository** es una interfaz proporcionada por **Spring Data JPA** que extiende la interfaz **PagingAndSortingRepository**, que a su vez extiende la interfaz **CrudRepository**. **JpaRepository** proporciona un conjunto completo de métodos CRUD (Crear, Leer, Actualizar y Borrar) y métodos para la paginación y la ordenación.

### Funcionalidades Proporcionadas por JpaRepository

Cuando extiendes **JpaRepository**, obtienes acceso a un **conjunto de métodos predefinidos** que te permiten realizar operaciones de persistencia sin necesidad de implementar esos métodos manualmente.

Métodos comunes que proporciona JpaRepository:

#### CRUD (Create, Read, Update, Delete):

- **save(S entity)**: Guarda una entidad.
- **findById(ID id)**: Encuentra una entidad por su ID.
- **existsById(ID id)**: Verifica si una entidad con un ID dado existe.
- **findAll()**: Encuentra todas las entidades.
- **findAllById(Iterable<ID> ids)**: Encuentra todas las entidades por sus IDs.
- **count()**: Cuenta todas las entidades.
- **deleteById(ID id)**: Borra una entidad por su ID.
- **delete(S entity)**: Borra una entidad.
- **deleteAll()**: Borra todas las entidades.

#### Paginación y Ordenación:

- **findAll(Pageable pageable)**: Encuentra todas las entidades con paginación.
- **findAll(Sort sort)**: Encuentra todas las entidades con ordenación.

## Creación de métodos personalizados

Además de los métodos proporcionados por defecto, puedes definir métodos personalizados en tu repositorio utilizando el poder de Spring Data JPA. Hay varias formas de hacerlo:

1. **Consultas Basadas en Convenciones de Nombres:**
  - Spring Data JPA genera automáticamente consultas basadas en los nombres de los métodos que sigan ciertas convenciones.
2. **Uso de @Query:**
  - Puedes usar la anotación @Query para definir consultas JPQL o SQL nativas personalizadas.
3. **Consulta con Criterias API o Especificaciones:**
  - Para consultas más complejas, puedes usar Criterias API o las especificaciones de Spring Data JPA.

*Ejemplo (No es parte del caso)*

### Consultas Basadas en Convenciones de Nombres (ejemplo)

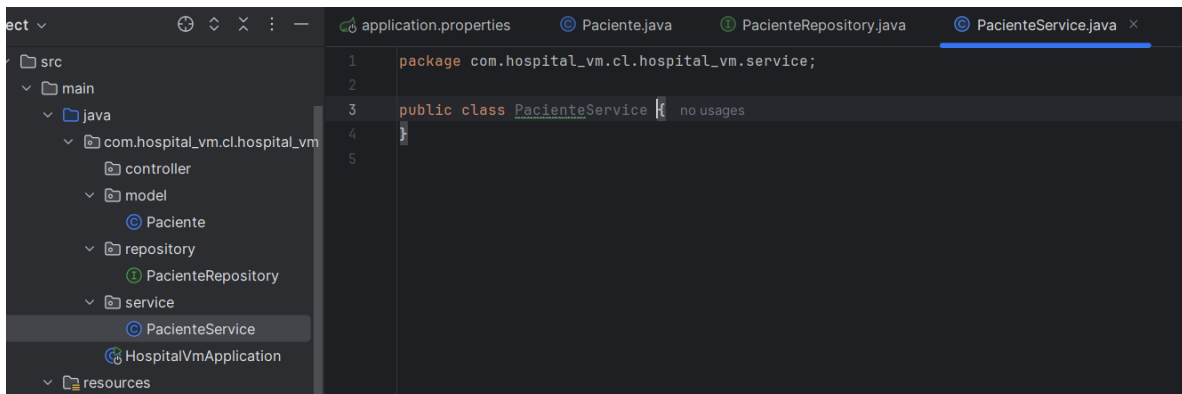
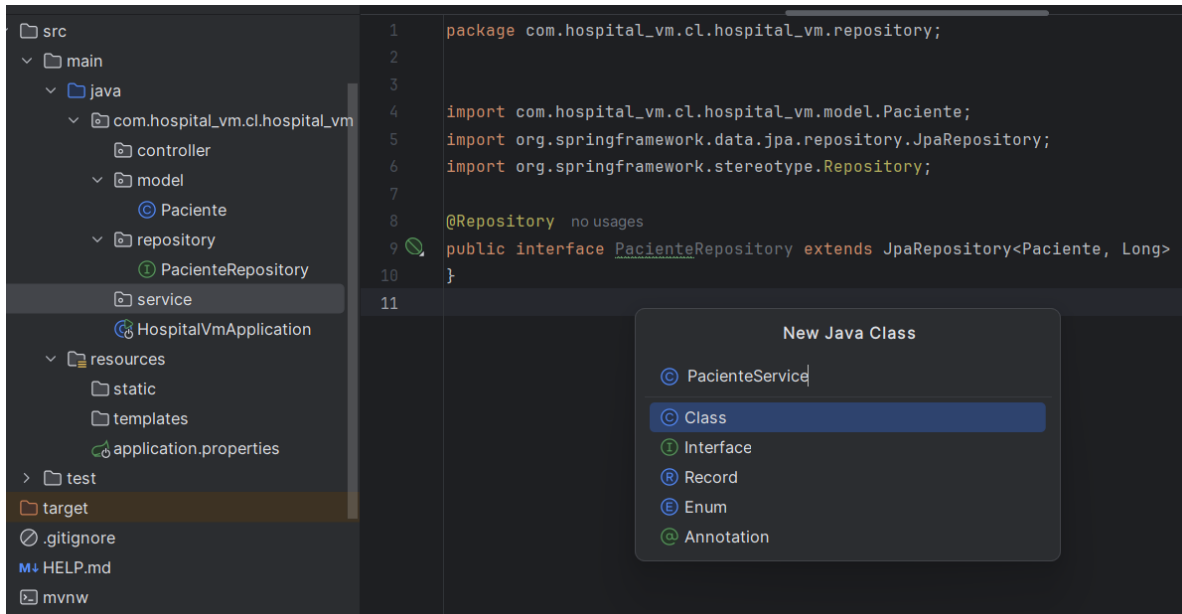
```
1 import com.hospital_vm.cl.hospital_vm.model.Paciente;
2 import org.springframework.data.jpa.repository.JpaRepository;
3 import org.springframework.stereotype.Repository;
4
5 import java.util.List;
6
7 @Repository no usages
8 public interface PacienteRepository extends JpaRepository<Paciente, Long> {
9
10     // Encuentra pacientes por apellidos
11     List<Paciente> findByApellidos(String apellidos); no usages
12
13     // Encuentra pacientes por correo electrónico
14     Paciente findByCorreo(String correo); no usages
15
16     // Encuentra pacientes por nombre y apellido
17     List<Paciente> findByNombreAndApellidos(String nombre, String apellidos); no usages
18 }
```

### Uso de @Query (ejemplo)

```
1 package com.hospital_vm.cl.hospital_vm.repository;
2
3 import com.hospital_vm.cl.hospital_vm.model.Paciente;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.query.Param;
7 import org.springframework.stereotype.Repository;
8
9 import java.util.List;
10
11 @Repository no usages
12 public interface PacienteRepository extends JpaRepository<Paciente, Long> {
13
14     // Usando JPQL
15     @Query("SELECT p FROM Paciente p WHERE p.apellidos= :apellido") no usages
16     List<Paciente> buscarPorApellidos(@Param("apellidos") String apellido);
17
18     // Usando SQL nativo
19     @Query(value = "SELECT * FROM paciente WHERE correo = :correo", nativeQuery = true) no usages
20     Paciente buscarPorCorreo(@Param("correo") String correo);
21 }
```

## Paso 2: Carpeta service

- Crear una clase llama **PacienteService**



Añadir lo siguiente al servicio **PacienteService**.

- Las funciones pueden estar en español o en inglés.
- Se dejaron en inglés para entender el verbo utilizando en el ORM.

```
application.properties  Paciente.java  PacienteRepository.java  PacienteService.java x
1  package com.hospital_vm.cl.hospital_vm.service;
2
3  import com.hospital_vm.cl.hospital_vm.model.Paciente;
4  import com.hospital_vm.cl.hospital_vm.repository.PacienteRepository;
5  import jakarta.transaction.Transactional;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.stereotype.Service;
8
9  import java.util.List;
10
11  @Service no usages
12  @Transactional
13  public class PacienteService {
14
15      @Autowired
16      private PacienteRepository pacienteRepository;
17
18      public List<Paciente> findAll() { no usages
19          return pacienteRepository.findAll();
20      }
21
22      public Paciente findById(long id) { no usages
23          return pacienteRepository.findById(id).get();
24      }
25
26      public Paciente save(Paciente paciente) { no usages
27          return pacienteRepository.save(paciente);
28      }
29
30      public void delete(Long id) { no usages
31          pacienteRepository.deleteById(id);
32      }
33  }
34
```

**Observación:** La función *save* funciona tanto como para crear o actualizar

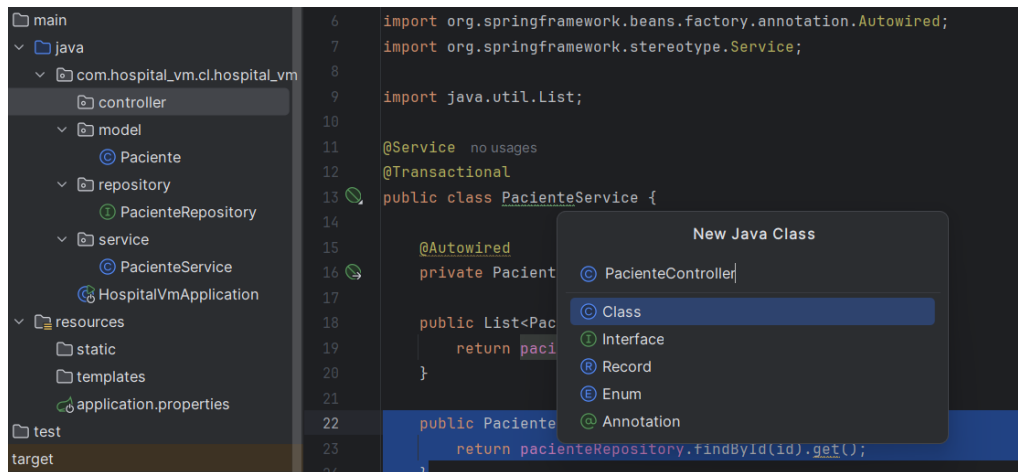
## ¿Qué es @Transactional?

La anotación **@Transactional** en la clase **PacienteService** indica que todos los métodos de esta clase deben ejecutarse dentro de una transacción. Esto significa que si cualquier método de esta clase falla, la transacción se revertirá (**rollback**), asegurando la consistencia de los datos.

En el servicio podemos utilizar lo que instanciamos de la interface

### Paso 3: Carpeta controlador

- Crear la clase **PacienteController**



```

6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.stereotype.Service;
8
9  import java.util.List;
10
11  @Service no usages
12  @Transactional
13  public class PacienteService {
14
15      @Autowired
16      private PacienteRepository pacienteRepository;
17
18      public List<Paciente> findAll() {
19          return pacienteRepository.findAll();
20      }
21
22      public Paciente findById(Long id) {
23          return pacienteRepository.findById(id).get();
24      }
25  }
  
```



#### Paso 4: Listar pacientes de la base de datos

- Añadimos solo el método **listar** en PacienteController
- Este método nos entregará todos los pacientes

```

1 package com.hospital_vm.cl.hospital_vm.controller;
2
3 import com.hospital_vm.cl.hospital_vm.model.Paciente;
4 import com.hospital_vm.cl.hospital_vm.service.PacienteService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RestController;
11
12 import java.util.List;
13
14 @RestController
15 @RequestMapping("/api/v1/pacientes")
16 public class PacienteController {
17
18     @Autowired
19     private PacienteService pacienteService;
20
21     @GetMapping
22     public ResponseEntity<List<Paciente>> listar() {
23         List<Paciente> pacientes = pacienteService.findAll();
24         if (pacientes.isEmpty()) {
25             return ResponseEntity.noContent().build(); //alternativa 2 -> return new ResponseEntity<>(HttpStatus.NO_CONTENT);
26         }
27         return ResponseEntity.ok(pacientes); //alternativa 2 -> return new ResponseEntity<>(pacientes, HttpStatus.OK);
28     }
29 }
30
  
```

Analicemos el retorno **ResponseEntity<List<Paciente>>**

Es una clase en Spring Framework que representa una respuesta HTTP completa, incluyendo el cuerpo, el estado y los encabezados. En este caso específico, se está utilizando para devolver una lista de objetos Paciente como cuerpo de la respuesta HTTP.

- **ResponseEntity:**
  - Es una clase genérica en Spring que permite crear una **respuesta HTTP** con un tipo específico de cuerpo de respuesta.
  - Permite configurar el cuerpo, el estado y los encabezados de la respuesta.
- **<List<Paciente>>:**
  - Especifica el tipo del cuerpo de la respuesta, que en este caso es una lista de objetos Paciente.

#### ¿Porque es mejor usar ResponseEntity?

**Flexibilidad:** Permite devolver respuestas HTTP completas con cuerpo, estado y encabezados personalizados.

**Claridad:** Hace explícito el tipo de cuerpo de respuesta que se devuelve.

**Manejo de Errores:** Facilita el manejo de diferentes estados HTTP y mensajes de error.

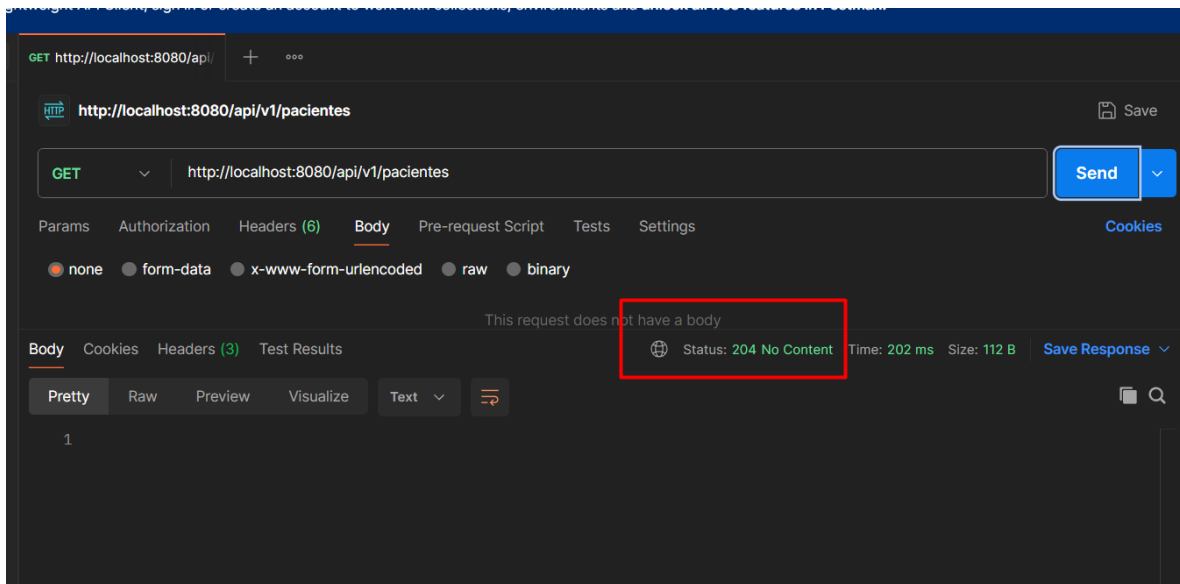
**Consistencia:** Proporciona una forma consistente de construir respuestas HTTP en tus controladores.

## Ejecutar programa

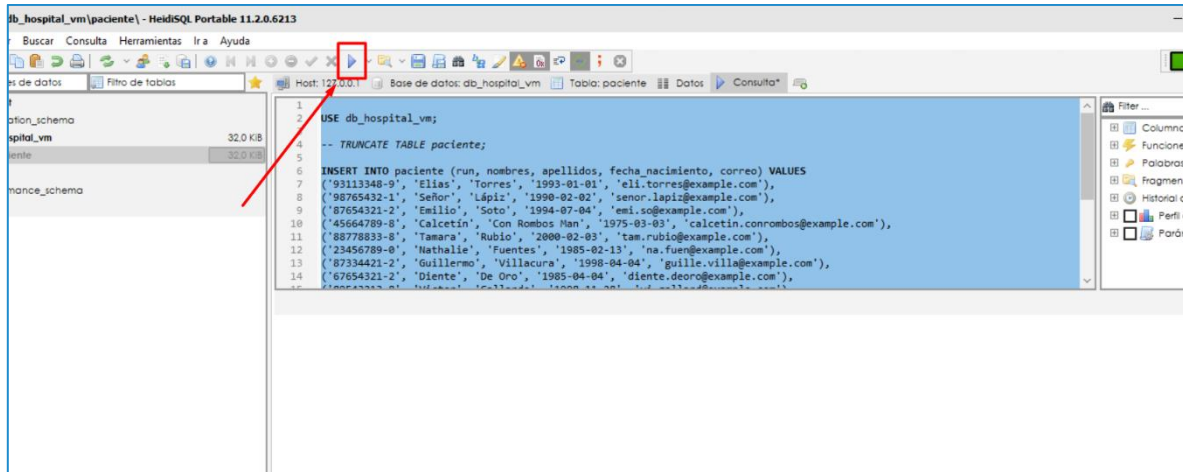
Probamos el **endpoint** en POSTMAN.

- Método GET <http://localhost:8080/api/v1/pacientes>

Funciona correctamente ya que en la base de datos no tenemos información



Añadir usuarios de forma masiva directamente en la base de datos

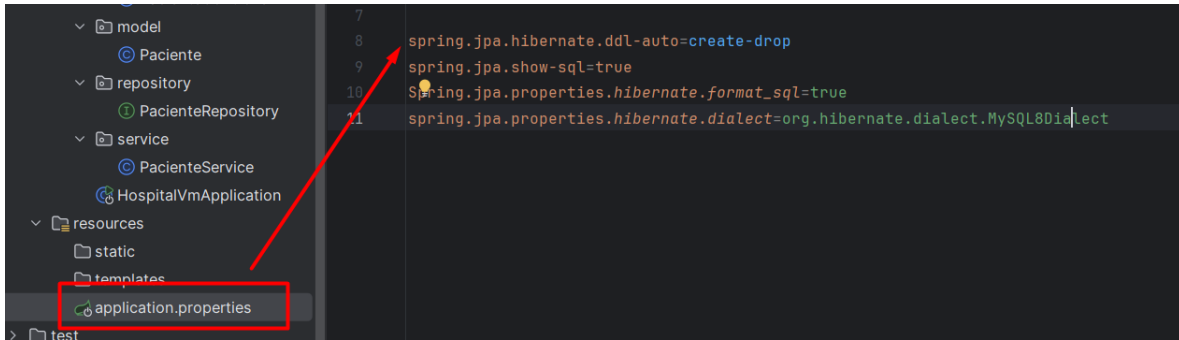


### Script de carga de usuarios masivos

```
USE db_hospital_vm;
-- TRUNCATE TABLE paciente;

INSERT INTO paciente (run, nombres, apellidos, fecha_nacimiento, correo) VALUES
('93113348-9', 'Elias', 'Torres', '1993-01-01', 'eli.torres@example.com'),
('98765432-1', 'Señor', 'Lápiz', '1990-02-02', 'senor.lapiz@example.com'),
('87654321-2', 'Emilio', 'Soto', '1994-07-04', 'emi.so@example.com'),
('45664789-8', 'Calcetín', 'Con Rombos Man', '1975-03-03', 'calcetin.conrombos@example.com'),
('88778833-8', 'Tamara', 'Rubio', '2000-02-03', 'tam.rubio@example.com'),
('23456789-0', 'Nathalie', 'Fuentes', '1985-02-13', 'na.fuen@example.com'),
('87334421-2', 'Guillermo', 'Villacura', '1998-04-04', 'guille.villa@example.com'),
('67654321-2', 'Diente', 'De Oro', '1985-04-04', 'diente.deoro@example.com'),
('89543213-8', 'Victor', 'Gallardo', '1998-11-28', 'vi.gallard@example.com'),
('34567890-3', 'Chico', 'Terry', '1992-05-05', 'chico.terry@example.com'),
('25618901-5', 'Benjamin', 'Mora', '1990-07-07', 'bej.mora@example.com'),
('76543210-4', 'Taca', 'Taca', '1988-06-06', 'taca.taca@example.com'),
('89283701-2', 'Jorge', 'Niochet', '1986-05-17', 'jo.niochet@example.com'),
('22222111-5', 'Guaripolo', '', '1979-07-07', 'guaripolo@example.com'),
('45178901-K', 'Fernando', 'Silva', '1979-08-07', 'fsilva@example.com'),
('65432109-6', 'Policarpo', 'Avendaño', '1991-08-08', 'policarpo.avendano@example.com'),
('32761432-1', 'Abraham', 'Sepulveda', '1990-02-02', 'brama.sep@example.com'),
('56789012-7', 'Bodoque', '', '1983-09-09', 'bodoque@example.com'),
('45678901-5', 'Camilo Alexis', 'Muñoz', '1989-02-07', 'cam.munoz@example.com'),
('54321098-8', 'Juanín', 'Juan Harry', '1987-10-10', 'juanin.juanharry@example.com');
```

*Si ya tenemos información creada y queremos borrar la base de datos podemos hacer un **create-drop** y esto elimina la base de datos*

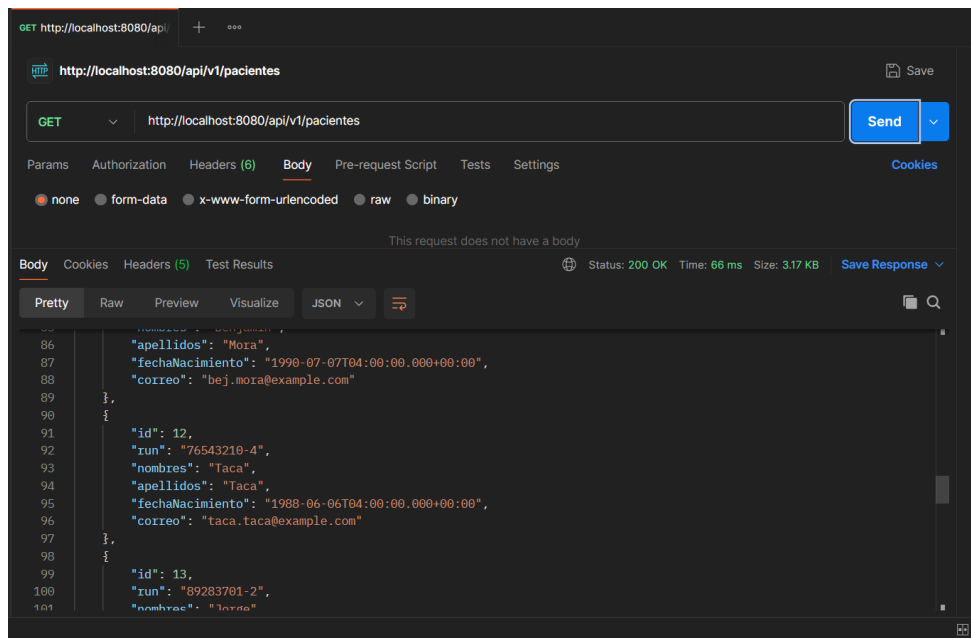


```
spring.jpa.hibernate.ddl-auto=create-drop
```

Probamos de nuevo el **endpoint** en POSTMAN.

- Método GET <http://localhost:8080/api/v1/paciente>

Funciona correctamente, entregando información de los usuarios.



## Paso 5: Carpeta controlador

- Añadir los siguientes métodos faltantes

```

HospitalVmApplication
PacienteController.java
1 package com.hospital_vm.cl.hospital_vm.controller;
2
3 import com.hospital_vm.cl.hospital_vm.model.Paciente;
4 import com.hospital_vm.cl.hospital_vm.service.PacienteService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.*;
9
10 import java.util.List;
11
12 @RestController
13 @RequestMapping("/api/v1/pacientes")
14 public class PacienteController {
15
16     @Autowired
17     private PacienteService pacienteService;
18
19     @GetMapping
20     public ResponseEntity<List<Paciente>> listar() {
21         List<Paciente> pacientes = pacienteService.findAll();
22         if (pacientes.isEmpty()) {
23             return ResponseEntity.noContent().build();
24             //alternativa 2 -> return new ResponseEntity<>(HttpStatus.NO_CONTENT);
25         }
26         return ResponseEntity.ok(pacientes);
27         //alternativa 2 -> return new ResponseEntity<>(pacientes, HttpStatus.OK);
28     }
29
30     @PostMapping
31     public ResponseEntity<Paciente> guardar(@RequestBody Paciente paciente) {
32         Paciente productoNuevo = pacienteService.save(paciente);
33         return ResponseEntity.status(HttpStatus.CREATED).body(productoNuevo);
34         // return new ResponseEntity<>(productoNuevo, HttpStatus.ACCEPTED);
35     }
36
37     @GetMapping("/{id}")
38     public ResponseEntity<Paciente> buscar(@PathVariable Integer id) {
39         try {
40             Paciente paciente = pacienteService.findById(id);
41             return ResponseEntity.ok(paciente);
42         } catch (Exception e) {
43             return ResponseEntity.notFound().build();
44         }
45     }
46

```

```

46
47 @PutMapping("/{id}")
48 public ResponseEntity<Paciente> actualizar(@PathVariable Integer id, @RequestBody Paciente paciente) {
49     try {
50         Paciente pac = pacienteService.findById(id);
51         pac.setId(id);
52         pac.setRun(paciente.getRun());
53         pac.setNombres(paciente.getNombres());
54         pac.setApellidos(paciente.getApellidos());
55         pac.setFechaNacimiento(paciente.getFechaNacimiento());
56         pac.setCorreo(paciente.getCorreo());
57
58         pacienteService.save(pac);
59         return ResponseEntity.ok(paciente);
60     } catch (Exception e) {
61         return ResponseEntity.notFound().build();
62     }
63 }
64
65 @DeleteMapping("/{id}")
66 public ResponseEntity<?> eliminar(@PathVariable Long id) {
67     try {
68         pacienteService.delete(id);
69         return ResponseEntity.noContent().build();
70     } catch (Exception e) {
71         return ResponseEntity.notFound().build();
72     }
73 }
74 }
75

```

## Paso 6: Probar la aplicación en POSTMAN

### CREAR PACIENTE

- Método: **POST**
- URL: <http://localhost:8080/api/v1/pacientes>
- Body: marcar raw
  - Parámetro de entrada JSON

```

{
  "run": "981276344-K",
  "nombres": "Alexis",
  "apellidos": "Sanchez",
  "fechaNacimiento": "1988-11-19T04:00:00.000+00:00",
  "correo": "alex.tocopilla@example.com"
}

```

Funciona correctamente, añade al nuevo usuario y le asigna un ID autoincrementable.

POST http://localhost:8080/api/v1/pacientes

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

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

```

1  {
2    "run": "981276344-K",
3    "nombres": "Alexis",
4    "apellidos": "Sanchez",
5    "fechaNacimiento": "1988-11-19T04:00:00.000+00:00",
6    "correo": "alex.tocopilla@example.com"
7  }

```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 225 ms Size: 327 B Save Response

Pretty Raw Preview Visualize JSON

```

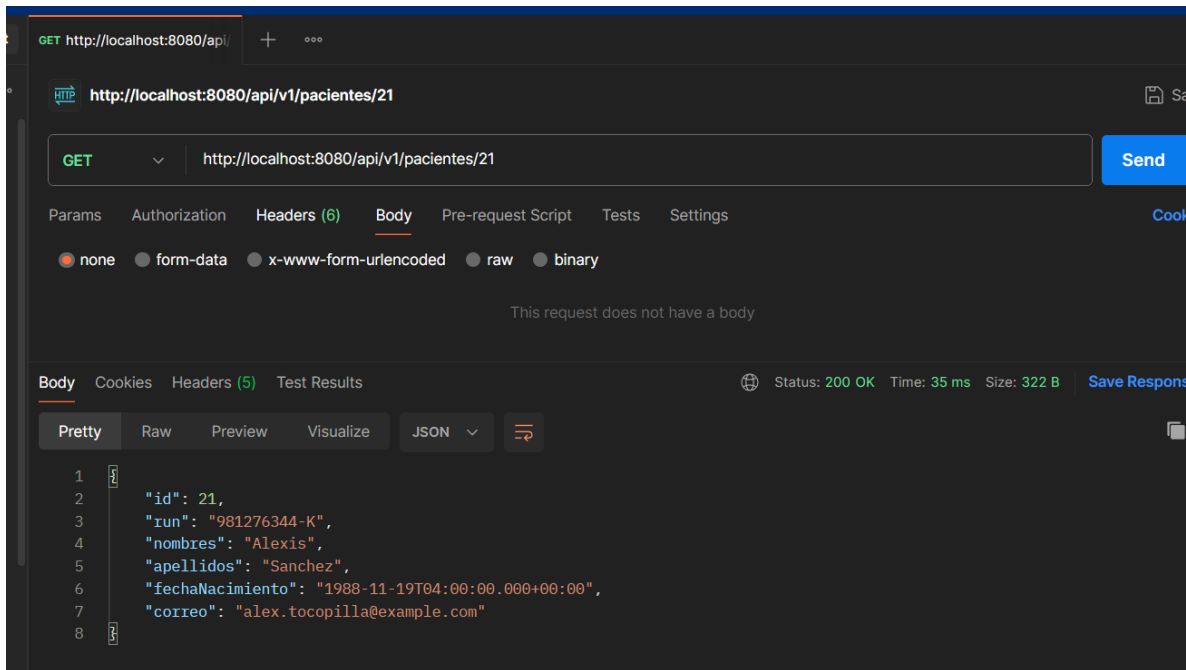
1  {
2    "id": 21,
3    "run": "981276344-K",
4    "nombres": "Alexis",
5    "apellidos": "Sanchez",
6    "fechaNacimiento": "1988-11-19T04:00:00.000+00:00",
7    "correo": "alex.tocopilla@example.com"
8  }

```

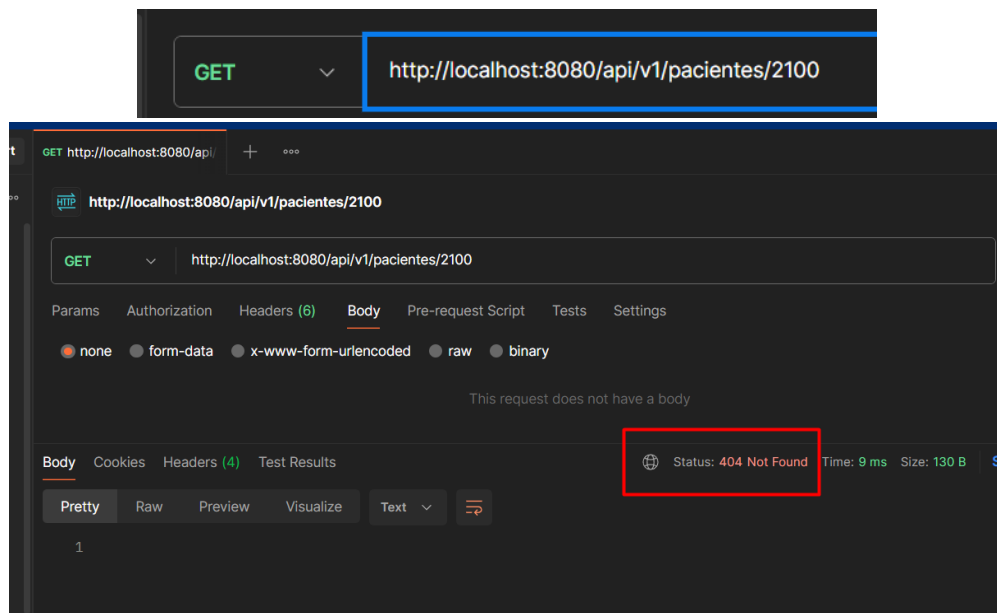
## BUSCAR PACIENTE

- Método: **GET**
- URL: <http://localhost:8080/api/v1/pacientes/{id}>

GET http://localhost:8080/api/v1/pacientes/21



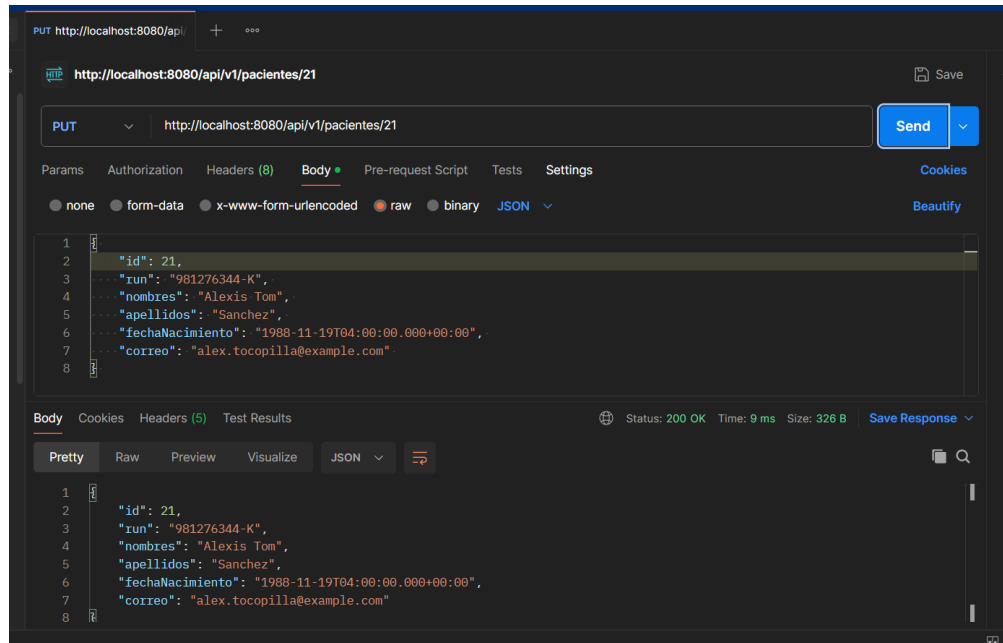
Si no existe, debe mostrar un error 404





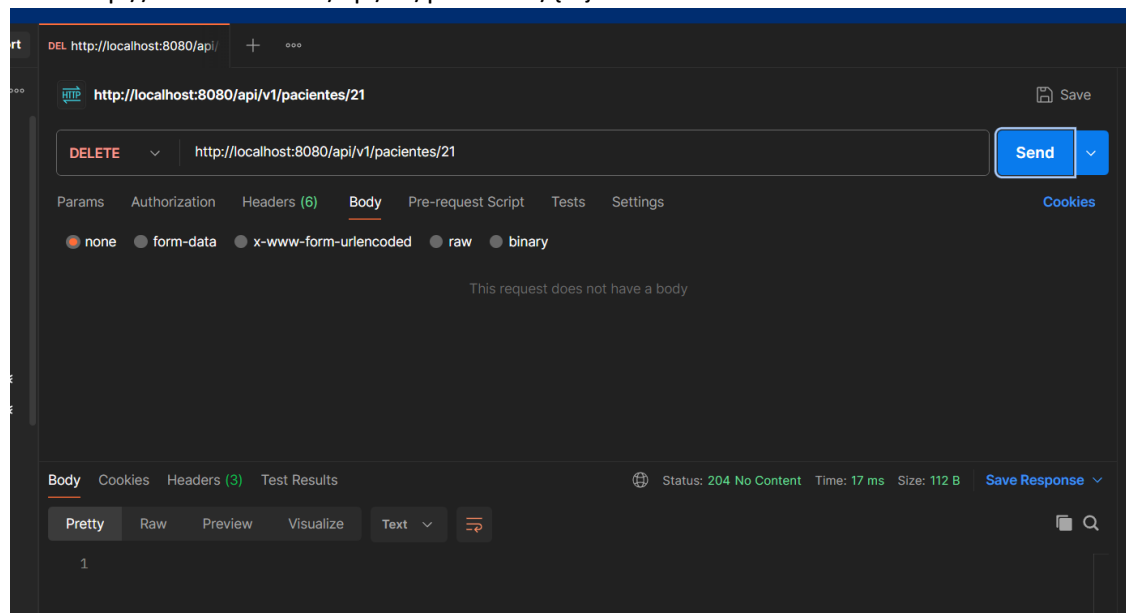
## ACTUALIZAR PACIENTE

- Método: **PUT**
- URL: <http://localhost:8080/api/v1/pacientes/{id}>
- Se añade el segundo nombre al atributo de nombres.



## ELIMINAR PACIENTE

- Método: **DELETE**
- URL: `http://localhost:8080/api/v1/pacientes/{id}`



Después eliminar el recurso se verifica si existe.

- Nos entrega un resultado 404 ya que el recurso no existe

