PRÁCTICA 3 — Spring Boot + JPA + HTML + fetch + SQL Server (Parte 1)

---

1. Crear proyecto en Spring Initializr

Ir a:

https://start.spring.io

Configurar:

```
Project: Maven
Language: Java
Spring Boot: 3.x
Group: com.demo
Artifact: pacientes
Name: pacientes
Package: com.demo.pacientes
Packaging: Jar
Java: 17
```

Dependencies:

```
Spring Web
Spring Data JPA
MS SQL Server Driver

Generate & Download zip
```

Descomprimir

Abrir en VSCode

2. Estructura esperada

```
src/main/java/com/demo/pacientes
    PacientesApplication.java

src/main/resources
    application.properties
    static/
```

3. Si no cuenta con su base de datos de vida sana, haga una base de datos SQL Server con su respectivo login y acceso

Ejemplo:

```
CREATE DATABASE vida_sana;
GO
```

Tabla:

```
USE DemoSpring;

CREATE TABLE Pacientes(
    IdPaciente INT IDENTITY PRIMARY KEY,
    Nombre VARCHAR(100),
    Telefono VARCHAR(20)
);
```

Nota: Quite o deje el IDENTITY (autoincrement) a su criterio

4. application.properties

Vaya al archive properties de su proyecto Spring Boot y editar:

```
src/main/resources/application.properties
```

Contenido:

```
spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=
vida_sana;encrypt=false
spring.datasource.username=loginMedico
spring.datasource.password=medico123

spring.jpa.hibernate.ddl-auto=none

spring.jpa.show-sql=true

spring.jpa.hibernate.naming.physical-
strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

spring.jpa.database-platform=org.hibernate.dialect.SQLServerDialect
```

Este archivo es la configuración default para su proyecto. Con la línea de DDL le decimos que no cree tablas automáticamente y que use las que tenemos en nuestra BD; con la línea de model.naming establecemos que no sea case sensitive en cuanto al nombre de los campos en nuestra base de datos.

5. Crear package model (Carpeta)

`com.demo.pacientes.model`

Crear:

Paciente.java

```
package com.demo.pacientes.model;

import jakarta.persistence.*;

@Entity
@Table(name = "Pacientes")
public class Paciente {

    @Id
    //@GeneratedValue(strategy = GenerationType.IDENTITY)
    //use esta línea si su ID es auto incrementable
    @Column(name = "IdPaciente")
    private int idPaciente;

    @Column(name = "Nombre")
    private String nombre;

    @Column(name = "Telefono")
    private String telefono;

    public int getIdPaciente() {
        return idPaciente;
    }

    public void setIdPaciente(int idPaciente) {
        this.idPaciente = idPaciente;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
}
```

6. Crear repository

Package:

```
com.demo.pacientes.repository
```

PacienteRepository.java

```
package com.demo.pacientes.repository;

import com.demo.pacientes.model.Paciente;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PacienteRepository
        extends JpaRepository<Paciente, Integer> {

}
```

---

7. Crear controller REST

Dentro del package (carpeta) com.demo.pacientes.controller crear:

PacienteController.java

```
package com.demo.pacientes.controller;

import com.demo.pacientes.model.Paciente;
import com.demo.pacientes.repository.PacienteRepository;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/pacientes")
@CrossOrigin
public class PacienteController {

    private final PacienteRepository repo;

    public PacienteController(PacienteRepository repo) {
        this.repo = repo;
    }

    @GetMapping
    public List<Paciente> listar() {
        return repo.findAll();
    }


}
```

8. Crear carpeta static si no existe

`src/main/resources/static`

Y dentro de ella crear:

`index.html`

9. index.html (tabla con fetch)

```
<!DOCTYPE html>
<html>
<head>
    <title>Pacientes</title>
</head>
<body>

<h1>Lista de pacientes</h1>

<table border="1">
    <thead>
        <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Telefono</th>
        </tr>
    </thead>
    <tbody id="tabla"></tbody>
</table>


<script>

fetch("/api/pacientes")
.then(r => r.json())
.then(data => {

    const tabla = document.getElementById("tabla");

    data.forEach(p => {

        tabla.innerHTML +=
        "<tr>" +
        "<td>" + p.idPaciente + "</td>" +
        "<td>" + p.nombre + "</td>" +
        "<td>" + p.telefono + "</td>" +
        "</tr>";

    });

});
</script>

</body>
</html>
```

10. Ejecutar

Desde terminal:

```
mvnw spring-boot:run
```

Abrir los siguientes URL y observe cómo en el primero (el URL de la API) va a ver el JSON; en cambio, en el index.html puede ver esos mismos datos pero ya en una tabla HTML.

```
http://localhost:8080/api/pacientes
```

```
http://localhost:8080/index.html
```

Espere a la explicación de su profesor para incluir estilos en su HTML y aplicarlos a su tabla.