

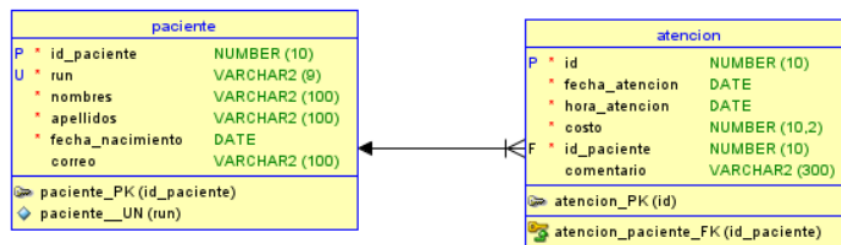
Guía práctica HOSPITAL V&M

PARTE 2

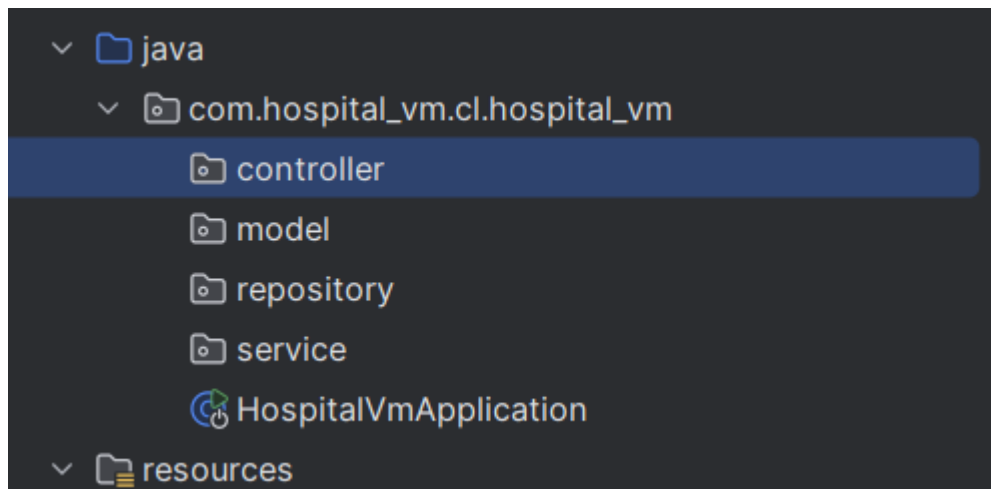
Modelo



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 modelo en Spring Boot.

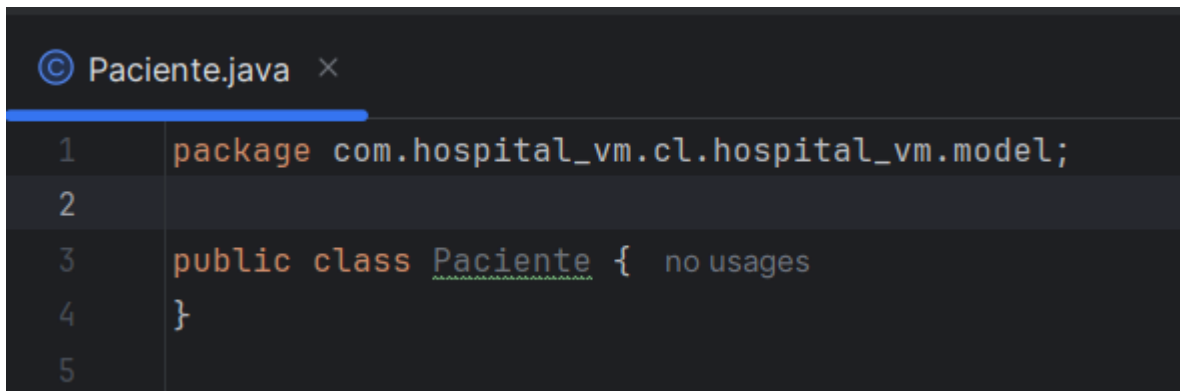
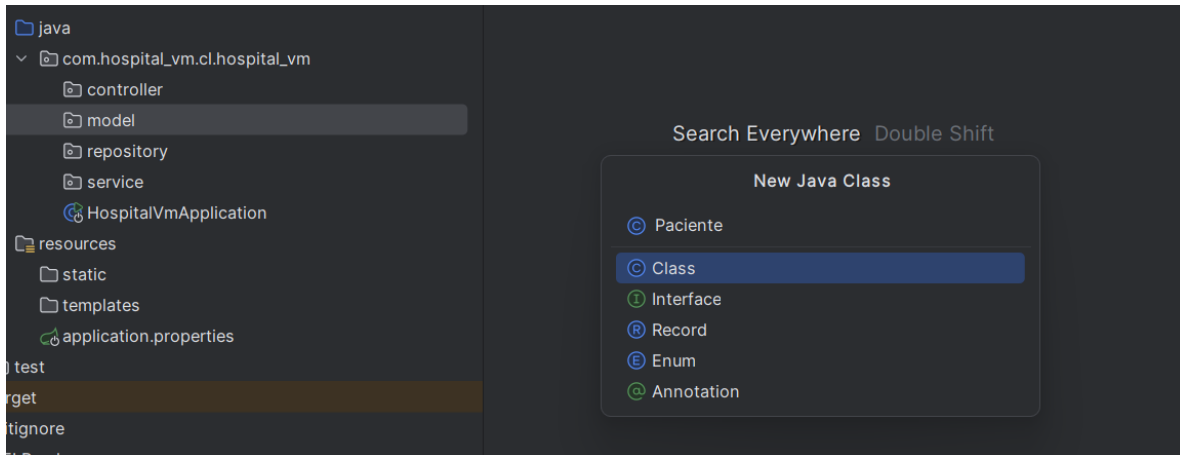


❑ Paso 1: Crea la estructura básica de las carpetas



Paso 2: Añadir modelo

- Crear clase Paciente



Añadir lo siguiente a la clase Paciente

- Vamos a copiar los atributos del modelo paciente
 - Dejaremos nombres como **nombre**, después lo cambiaremos
 - Dejaremos apellidos como **apellido**, después lo cambiaremos

```
Paciente.java x
1 package com.hospital_vm.cl.hospital_vm.model;
2
3 import jakarta.persistence.*;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 import java.util.Date;
9
10 @Entity no usages
11 @Table(name= "paciente")
12 @Data
13 @NoArgsConstructor
14 @AllArgsConstructor
15 public class Paciente {
16
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Integer id;
20
21     @Column(unique=true, length = 13, nullable=false)
22     private String run;
23
24     @Column(nullable=false)
25     private String nombre;
26
27     @Column(nullable=false)
28     private String apellido;
29
30     @Column(nullable=true)
31     private Date fechaNacimiento;
32
33     @Column(nullable=false)
34     private String correo;
35 }
36
```

Definiciones

¿Qué es @Entity?

La anotación @Entity se utiliza para marcar una clase como una entidad JPA. Esto significa que la clase se mapeará a una tabla en una base de datos relacional, y cada instancia de la clase representará una fila en la tabla.

Uso de @Entity

Al usar la anotación @Entity, debes cumplir con ciertos requisitos:

1. **Debe tener un identificador primario:** La entidad debe tener un campo que actúe como identificador primario. Este campo debe estar anotado con **@Id**. Opcionalmente, puedes usar la anotación **@GeneratedValue** para indicar que el valor del identificador se generará automáticamente.
2. **Debe estar mapeada a una tabla:** Puedes usar la anotación **@Table** para especificar el nombre de la tabla en la base de datos a la que se mapeará la entidad. Si no se especifica, JPA utilizará el nombre de la clase.
3. **Debe tener un constructor sin argumentos:** JPA requiere un constructor sin argumentos en la clase de entidad. Esto se puede proporcionar explícitamente o usando la anotación **@NoArgsConstructor** de Lombok.

Anotaciones de JPA

@Entity:

Marca la clase como una entidad JPA, lo que significa que JPA la tratará como una tabla en la base de datos.

Cada instancia de esta clase representa una fila en dicha tabla.

@Table(name= "paciente"):

Especifica el nombre de la tabla en la base de datos a la que se mapeará esta entidad.

En este caso, la tabla se llama **paciente**.

@Id:

Indica que el campo anotado es el identificador primario de la entidad.

Este campo será la clave primaria en la tabla de la base de datos.

@GeneratedValue(strategy = GenerationType.IDENTITY):

Define la estrategia para la generación del valor del identificador primario.

`GenerationType.IDENTITY` indica que el valor del ID se generará automáticamente por la base de datos, típicamente utilizando una columna auto-incremental.

@Column(unique=true, length = 13, nullable=false):

Configura la columna de la base de datos correspondiente al campo anotado.

unique=true asegura que los valores en esta columna sean únicos.

length = 13 especifica la longitud máxima de la columna.

nullable=false indica que la columna no puede contener valores nulos.

@Column(nullable=true):

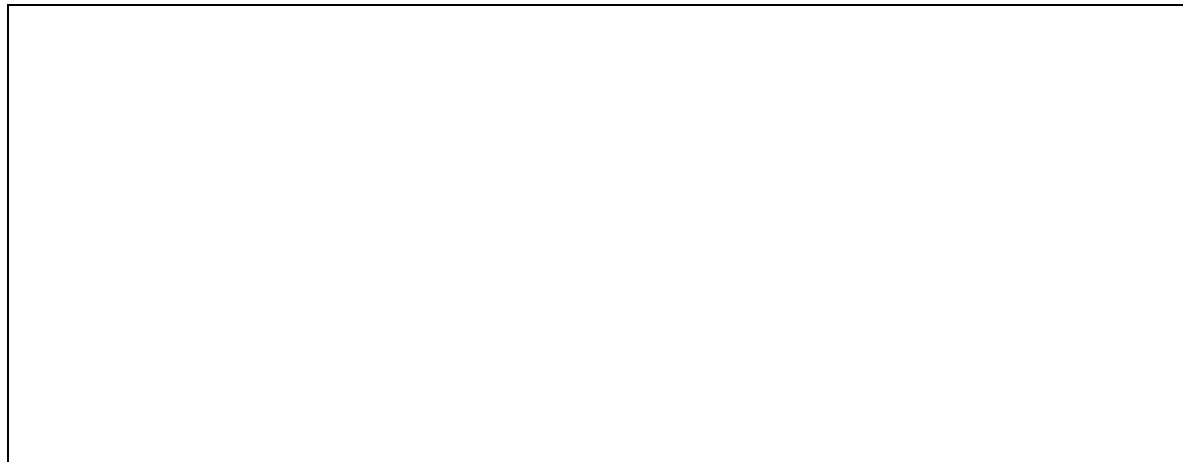
Indica que la columna puede contener valores nulos.

@Column(name="otro_nombre_columna"):

Indica que la columna puede tener otro nombre den la base de datos, pero en el modelo se llama diferente.

Actividad: Investiga Sprint data jpa y descubre que más podemos hacer con las @column

<https://www.geeksforgeeks.org/what-is-spring-data-jpa/?ref=lbp>



Ejecutar programa

Cuando ejecutemos el programa nos vamos a encontrar que automáticamente realizo la creación de la base de datos, esto pasa por el estado de la configuración del proyecto con la base de datos.

```

import java.util.Date;

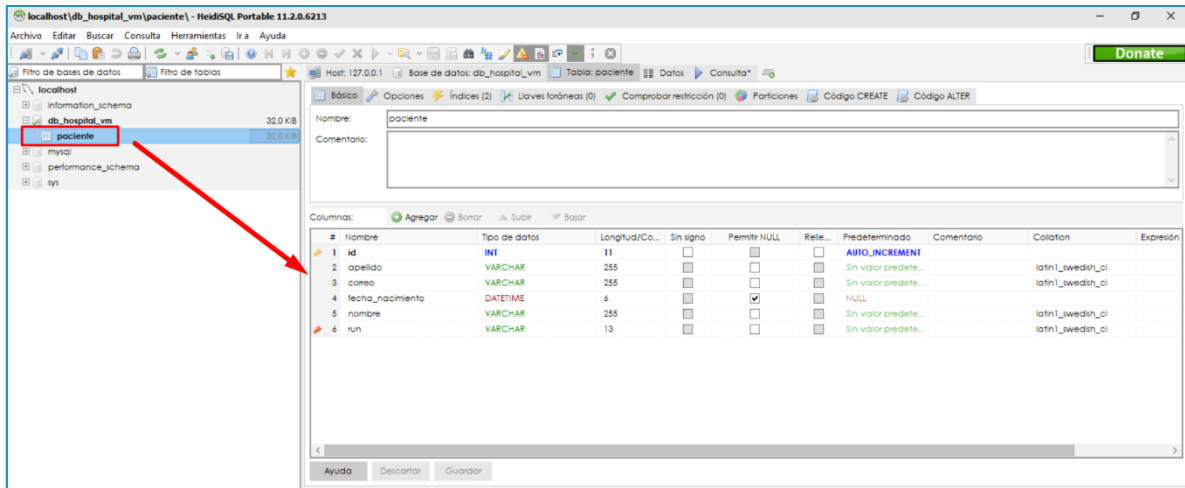
@Entity
@Table(name = "paciente")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Paciente {

    create table paciente (
      id integer not null auto_increment,
      apellido varchar(255) not null,
      correo varchar(255) not null,
      fecha_nacimiento datetime(6),
      nombre varchar(255) not null,
      run varchar(13) not null,
      primary key (id)
    ) engine=InnoDB

    alter table paciente
    drop index UKhkn138kpjgyff23m124ihokw

    alter table paciente
    add constraint UKhkn138kpjgyff23m124ihokw unique (run)
  
```

Verificamos que la tabla nueva se creó.



Qué pasa si nos equivocamos en escribir la variable y el programa realizo el cambio subiendolo a la base de datos.

Queremos pasar de

- nombre a nombres
- apellido a apellidos

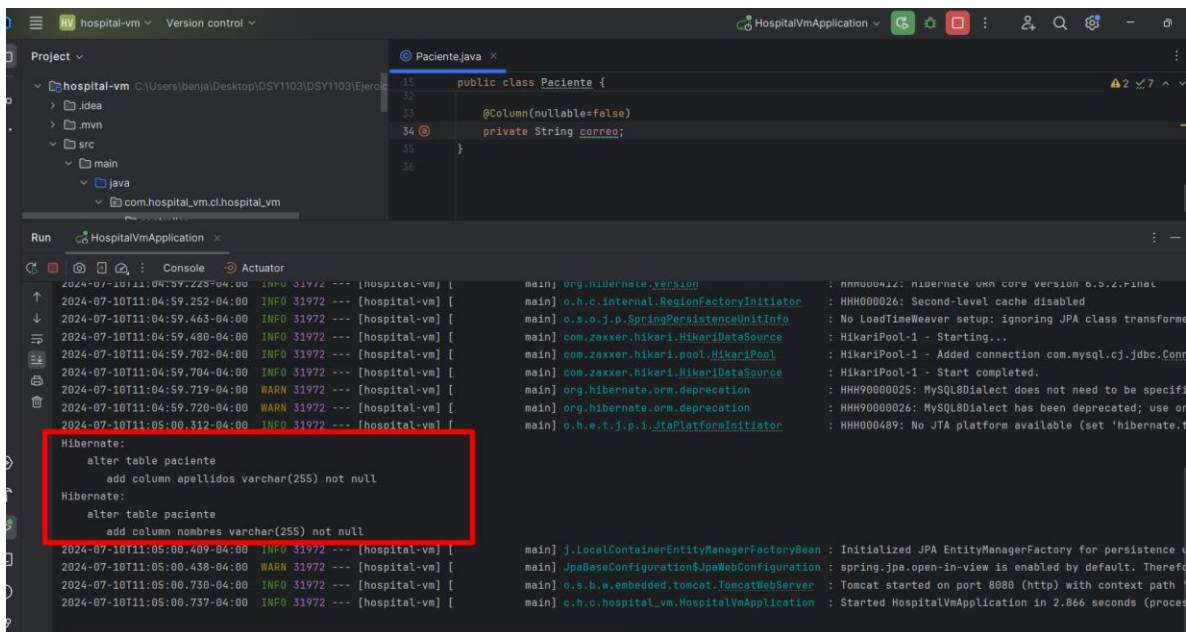
```
@Column(nullable=false) /
private String nombre;

@Column(nullable=false) /
private String apellido;
```

```
@Column(nullable=false) //
private String nombres;

@Column(nullable=false) //
private String apellidos;
```

Cuando ejecutamos el programa vemos que se hizo ***“alter table paciente”*** de los cambios realizados.



The screenshot shows an IDE with the following components:

- Project View:** Shows the project structure for 'hospital-vm'.
- Code Editor:** Displays the `Paciente.java` file with the following code:


```
public class Paciente {
    @Column(nullable=false)
    private String conreg;
}
```
- Run Console:** Shows the output of the application. A red box highlights the following Hibernate SQL commands:


```
Hibernate:
alter table paciente
add column apellidos varchar(255) not null

Hibernate:
alter table paciente
add column nombres varchar(255) not null
```
- Log Output:** Shows various logs including Hibernate version information, JPA configuration, and Tomcat startup details.

Y si observamos la base de datos, veremos que se añadió apellidos y nombres, pero no los sobrescribió.

MySQL Workbench interface showing the 'paciente' table structure. The table has 8 columns:

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir NULL	Rel
1	id	INT	11	<input type="checkbox"/>	<input type="checkbox"/>	
2	apellido	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	
3	correo	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	
4	fecha_nacimiento	DATETIME	6	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
5	nombre	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	
6	run	VARCHAR	13	<input type="checkbox"/>	<input type="checkbox"/>	
7	apellidos	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	
8	nombres	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	

Vamos a cambiar la propiedad de **hibernate.ddl-auto = create-drop** solo por una vez para que borre y se vuelva a iniciar la base de datos, luego volvemos a como estaba a **update**

```

1 spring.application.name=hospital-vm
2
3 spring.datasource.url=jdbc:mysql://localhost:3306/db_hospital_vm
4 spring.datasource.username=root
5 spring.datasource.password=
6
7
8 spring.jpa.hibernate.ddl-auto=create
9 spring.jpa.show-sql=true
10 Spring.jpa.properties.hibe
11 spring.jpa.properties.hibe

```

The IDE interface shows the 'application.properties' file. A red arrow points to the line `spring.jpa.hibernate.ddl-auto=create`. A tooltip is visible over this line, showing options: `create` (Create the schema and destroy previous data) and `create-drop` (Create and then destroy the schema at the ...).


```

1 spring.application.name=hospital-vm
2
3 spring.datasource.url=jdbc:mysql://localhost:3306/db_hospital_vm
4 spring.datasource.username=root
5 spring.datasource.password=
6
7
8 spring.jpa.hibernate.ddl-auto=create-drop
9 spring.jpa.show-sql=true
10 spring.jpa.properties.hibernate.format_sql=true
11 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
  
```

Ejecutamos la aplicación y va a borrar la base de datos

```

Hibernate:
drop table if exists paciente
Hibernate:
create table paciente (
  id integer not null auto_increment,
  fecha_nacimiento datetime(6),
  run varchar(13) not null,
  apellidos varchar(255) not null,
  correo varchar(255) not null,
  nombres varchar(255) not null,
  primary key (id)
) engine=InnoDB
Hibernate:
alter table paciente
  
```

Verificamos los cambios en la base de datos

#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir NULL
1	id	INT	11	<input type="checkbox"/>	<input type="checkbox"/>
2	fecha_nacimiento	DATETIME	6	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	run	VARCHAR	13	<input type="checkbox"/>	<input type="checkbox"/>
4	apellidos	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>
5	correo	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>
6	nombres	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>

