

Que es una Api Rest

- ▶ Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones. Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta). Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.

REST

- REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones que tener en cuenta en la arquitectura software que usaremos para crear aplicaciones web respetando HTTP.

JSON

- ▶ JSON es el lenguaje de programación más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.

```
{  
  "business_id": "PK6aSizckHFWk8i0oxt5DA",  
  "full_address": "400 Waterfront Dr E\nHomestead\nHomestead, PA 15120",  
  "hours": {},  
  "open": true,  
  "categories": [  
    "Burgers",  
    "Fast Food",  
    "Restaurants"  
  ],  
  "city": "Homestead",  
  "review_count": 5,  
  "name": "McDonald's",  
  "neighborhoods": [  
    "Homestead"  
  ],  
  "longitude": -79.910032,  
  "state": "PA",  
  "stars": 2,  
  "type": "Restaurant",  
  "url": "http://www.yelp.com/biz/mcdonald-s-homestead-2",  
  "zip_code": "15120",  
  "lat": 40.25555555555555  
}
```

Para que una API se considere de RESTful, debe cumplir los siguientes criterios:

- ▶ Arquitectura cliente-servidor compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP.
- ▶ Comunicación entre el cliente y el servidor sin estado, lo cual implica que no se almacena la información del cliente entre las solicitudes de GET y que cada una de ellas es independiente y está desconectada del resto.
- ▶ Datos que pueden almacenarse en caché y optimizan las interacciones entre el cliente y el servidor.

Para que una API se considere de RESTful, debe cumplir los siguientes criterios:

Una interfaz uniforme entre los elementos, para que la información se transfiera de forma estandarizada. Para ello deben cumplirse las siguientes condiciones:

- ▶ Los recursos solicitados deben ser identificables e independientes de las representaciones enviadas al cliente.
- ▶ El cliente debe poder manipular los recursos a través de la representación que recibe, ya que esta contiene suficiente información para permitirlo.
- ▶ Los mensajes autodescriptivos que se envíen al cliente deben contener la información necesaria para describir cómo debe procesarla.
- ▶ Debe contener hipertexto o hipermedios, lo cual significa que cuando el cliente acceda a algún recurso, debe poder utilizar hipervínculos para buscar las demás acciones que se encuentren disponibles en ese momento.

Para que una API se considere de RESTful, debe cumplir los siguientes criterios:

- ▶ Un sistema en capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores (los encargados de la seguridad, del equilibrio de carga, etc.) que participan en la recuperación de la información solicitada.
- ▶ Código disponible según se solicite (opcional), es decir, la capacidad para enviar códigos ejecutables del servidor al cliente cuando se requiera, lo cual amplía las funciones del cliente.

Métodos en una API REST

- ▶ **GET** es usado para recuperar un recurso.
- ▶ **POST** se usa la mayoría de las veces para crear un nuevo recurso. También puede usarse para enviar datos a un recurso que ya existe para su procesamiento. En este segundo caso, no se crearía ningún recurso nuevo.
- ▶ **PUT** es útil para crear o editar un recurso. En el cuerpo de la petición irá la representación completa del recurso. En caso de existir, se reemplaza, de lo contrario se crea el nuevo recurso.
- ▶ **PATCH** realiza actualizaciones parciales. En el cuerpo de la petición se incluirán los cambios a realizar en el recurso. Puede ser más eficiente en el uso de la red que PUT ya que no envía el recurso completo.
- ▶ **DELETE** se usa para eliminar un recurso.

Otras operaciones menos comunes pero también destacables son

- ▶ **HEAD** funciona igual que GET pero no recupera el recurso. Se usa sobre todo para testear si existe el recurso antes de hacer la petición GET para obtenerlo (un ejemplo de su utilidad sería comprobar si existe un fichero o recurso de gran tamaño y saber la respuesta que obtendríamos de la API REST antes de proceder a la descarga del recurso).
- ▶ **OPTIONS** permite al cliente conocer las opciones o requerimientos asociados a un recurso antes de iniciar cualquier petición sobre el mismo.

¿Cuáles son las ventajas de utilizar una API Rest?

► **Separación entre cliente y servidor**

Una de las ventajas de utilizar el modelo API Rest es la separación entre las aplicaciones de front-end y back-end.

Esto es importante para proteger el almacenamiento de datos, ya que no existe un tratamiento de las reglas comerciales, es decir, **solo se intercambia informaciones** sea para recuperar datos, o para insertar o eliminar nuevos registros.

Más visibilidad, confiabilidad y escalabilidad

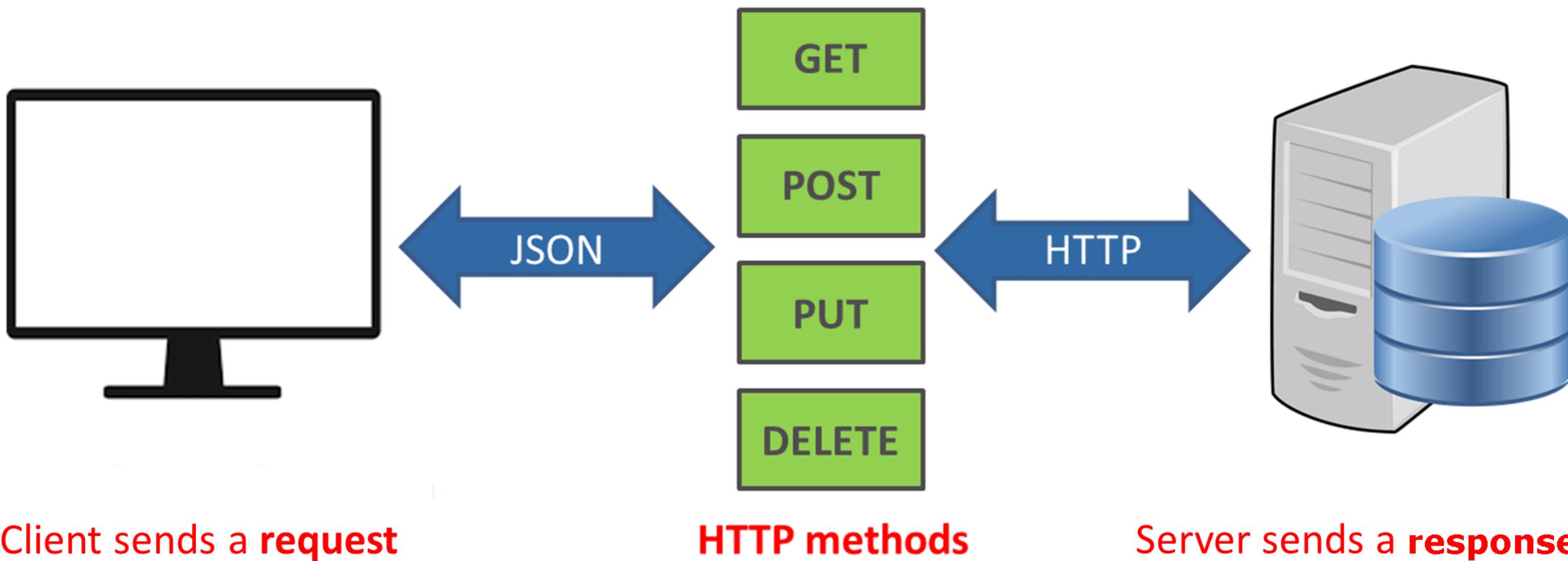
Debido a la separación cliente / servidor, hay mucha más facilidad durante el desarrollo de la aplicación. Esto se debe a que se puede escalar fácilmente, ya que no hay dificultad para vincular recursos.

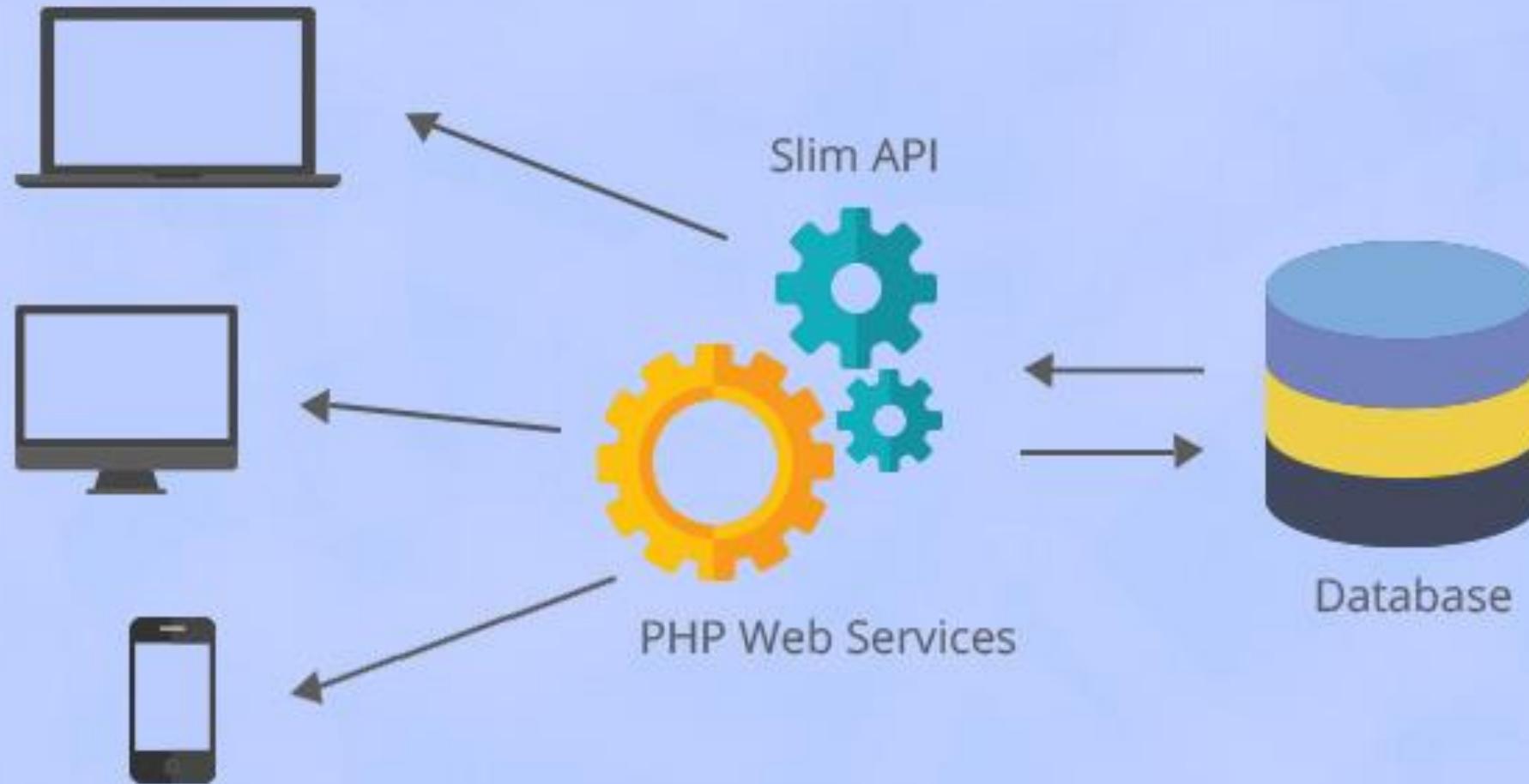
Como cada solicitud se realiza de forma única e independiente, es posible cambiar una solicitud a otro DNS, sin interferir con la aplicación.

En otras palabras, **una API Rest permite que la aplicación acceda a bases de datos desde diferentes servidores**, lo que a menudo es importante para el desarrollo en aplicaciones grandes. Por lo tanto, su uso garantiza una mayor visibilidad y credibilidad a la hora de utilizar estos recursos.

Multiplataforma

- ▶ Las requisiciones HTTP realizadas en API Rest devuelven datos en formato JSON. Cabe mencionar que existen otros posibles formatos de retorno, como XML, sin embargo, JSON es el más utilizado. Por lo tanto, la mayoría de los sitios que trabajan con este modelo reciben este formato de datos.
- ▶ **Esta característica es fundamental para el desarrollo de aplicaciones multiplataforma.** Eso se debe a que, al recibir los datos en este formato, la camada front-end de la aplicación es capaz de realizar el tratamiento adecuado para mostrar los resultados según el tipo de dispositivo utilizado.
- ▶ El uso de **Rest API es importante para agregar varias funciones al sitio.** Sus características permiten la integración con diferentes aplicaciones; entre ellos, redes sociales y sistemas de pago.
- ▶ Por eso, es una tecnología que garantiza una mayor fiabilidad y escalabilidad, además de facilitar el desarrollo de aplicaciones multiplataforma.





Creamos un proyecto en eclipse

- ▶ creamos un proyecto con las siguientes dependencias
- ▶ Spring Boot DevTools
- ▶ Spring Data JPA
- ▶ MySQL Driver
- ▶ Spring Web

Dependencias

- ▶ **Spring Boot DevTools** es la herramienta de Spring Boot que nos permite reiniciar de forma automática nuestras aplicaciones cada vez que se produce un cambio en nuestro código.
- ▶ **Spring Data JPA** Puede entenderse como el reencapsulado y abstracción de la especificación JPA. La capa inferior todavía se implementa utilizando la tecnología JPA de Hibernate, y se hace referencia al lenguaje de consulta JPQL (Java Persistence Query Language), que es parte de todo el ecosistema Spring
- ▶ **MySQL Driver** El concepto de Driver hace referencia al conjunto de clases necesarias que implementa de forma nativa el protocolo de comunicación con la base de datos MySQL.
- ▶ **Spring Web** proporciona integración HTTP central, incluidos algunos filtros Servlet útiles, Spring HTTP Invoker, infraestructura para integrarse con otros marcos web y tecnologías HTTP.



New Spring Starter Project Dependencies



Spring Boot Version: 2.5.7

Frequently Used:

- MySQL Driver
- Spring Web

- Spring Boot DevTools

- Spring Data JPA

Available:

Type to search dependences

- Developer Tools
- Google Cloud Platform
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops

Selected:

- Spring Boot DevTools
- Spring Data JPA
- MySQL Driver
- Spring Web



< Back

Next >

Finish

Cancel

New Spring Starter Project

Service URL

Name

Use default location

Location

Type: Packaging:

Java Version: Language:

Group

Artifact

Version

Description

Package

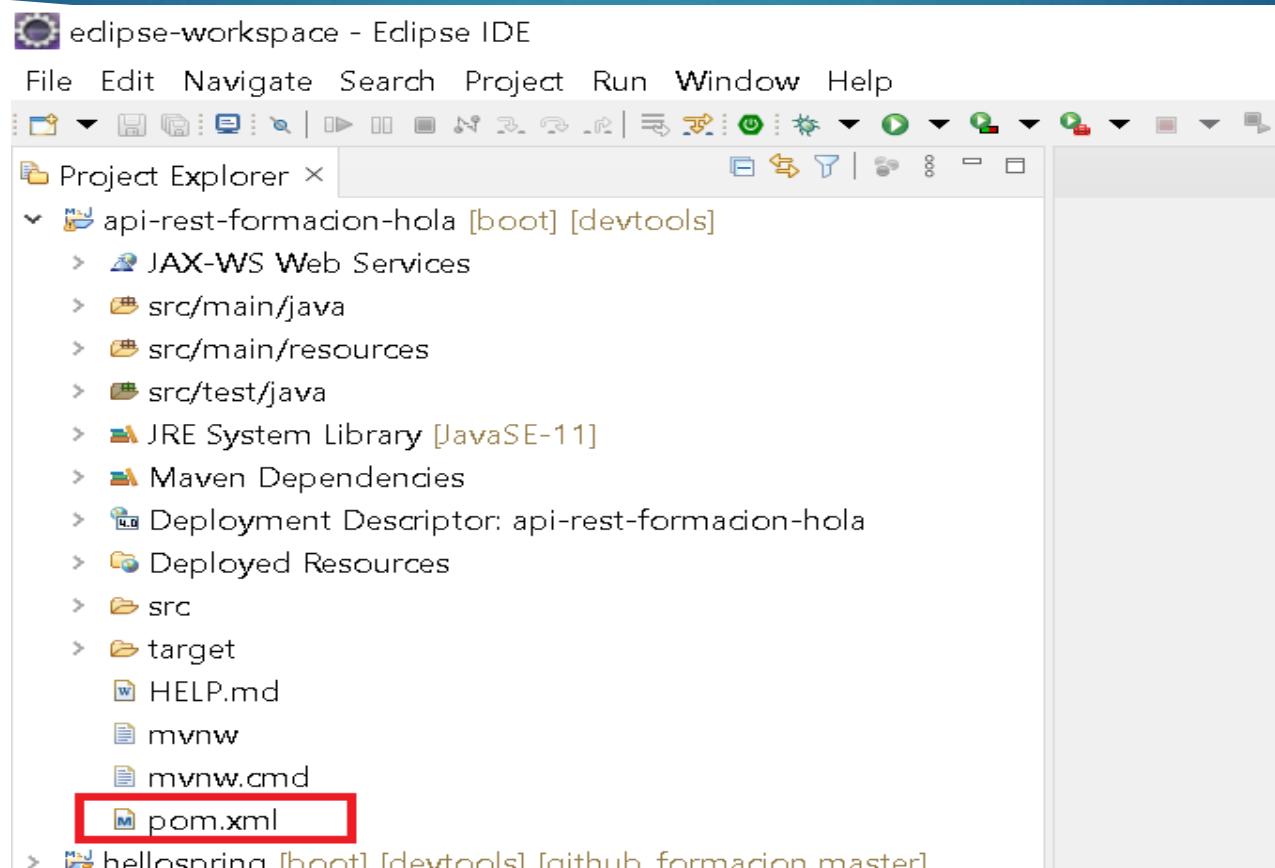
Working sets

Add project to working sets

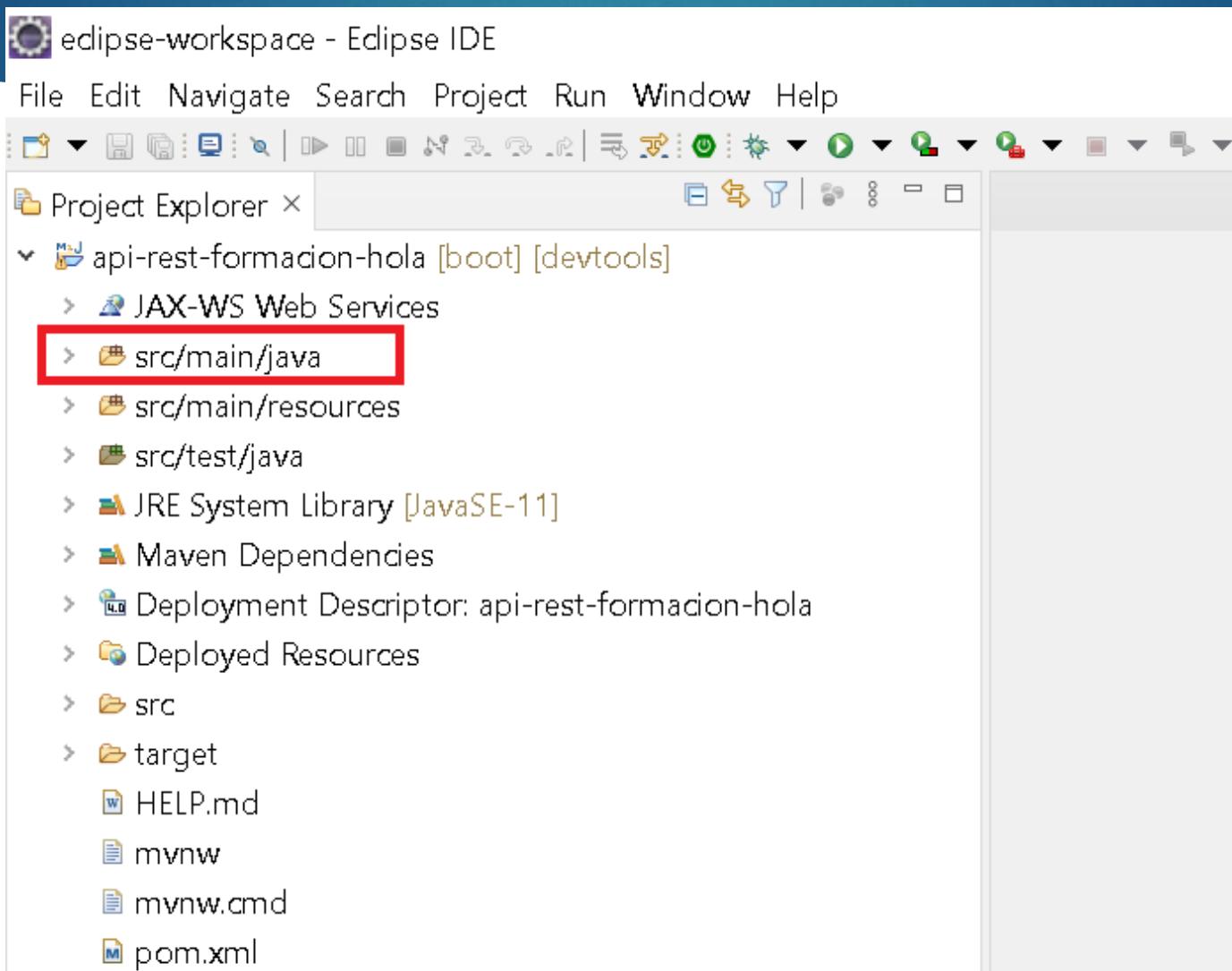
Working sets:

Estructura de un proyecto SpringBoot

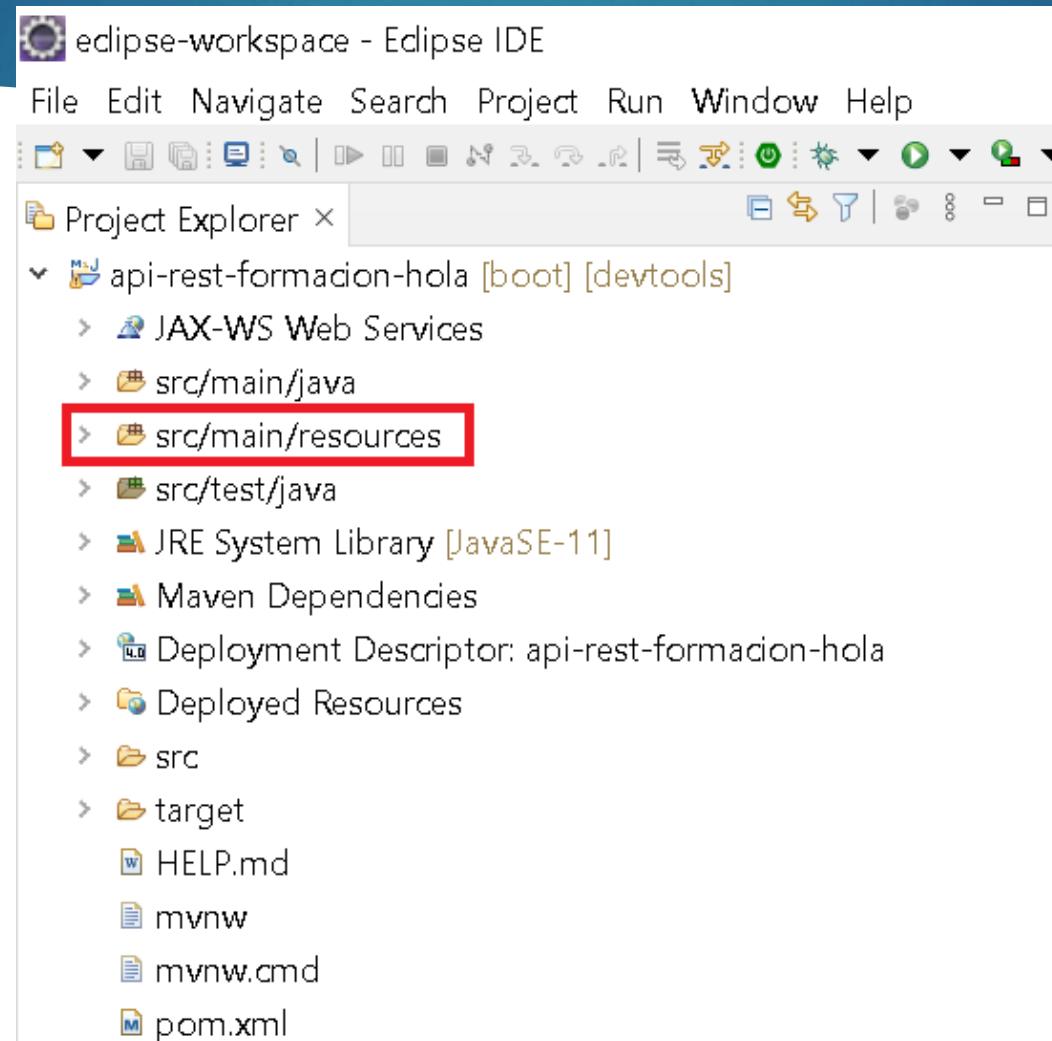
pom.xml es un archivo propio de un proyecto Maven, el cual contiene las configuraciones y dependencias que vamos a utilizar en nuestro proyecto



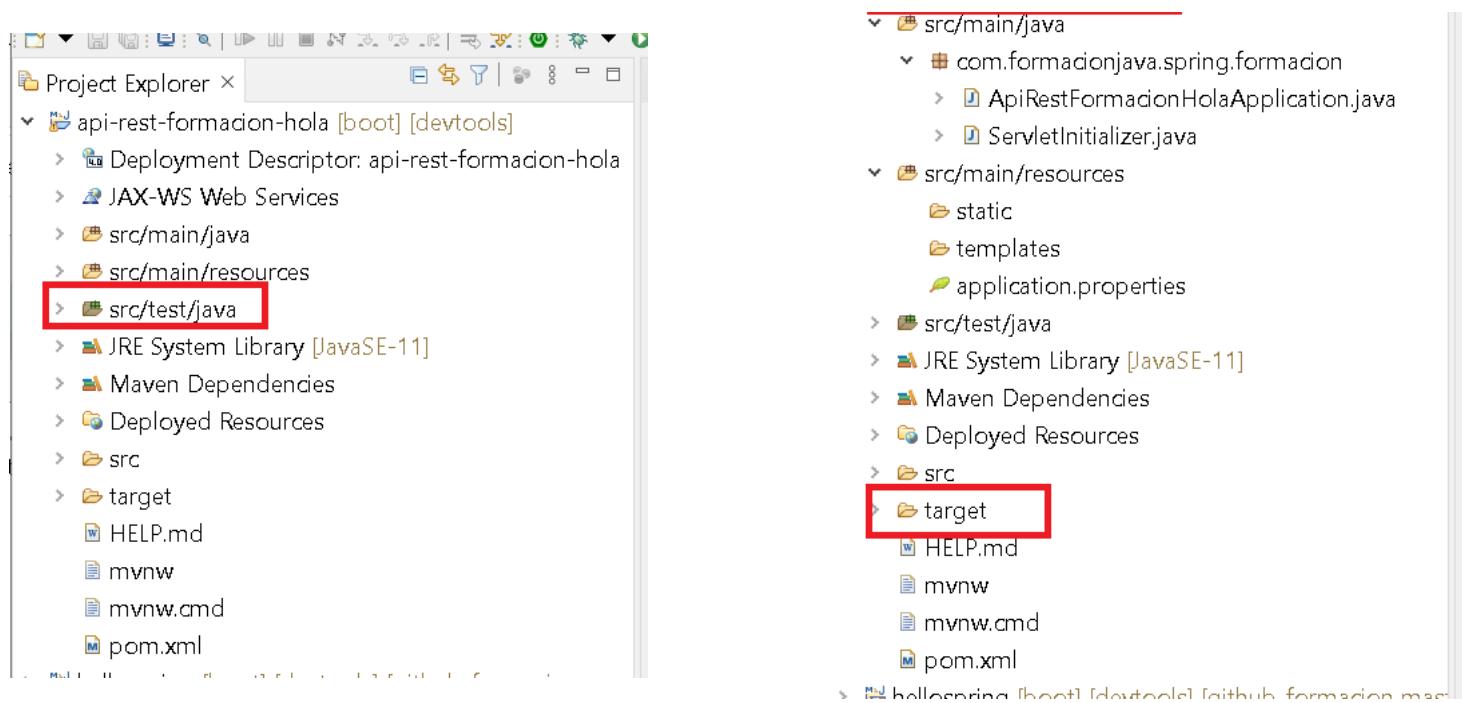
En src/main/java se encuentra nuestro paquete y clase principal, aquí va todo el código de java



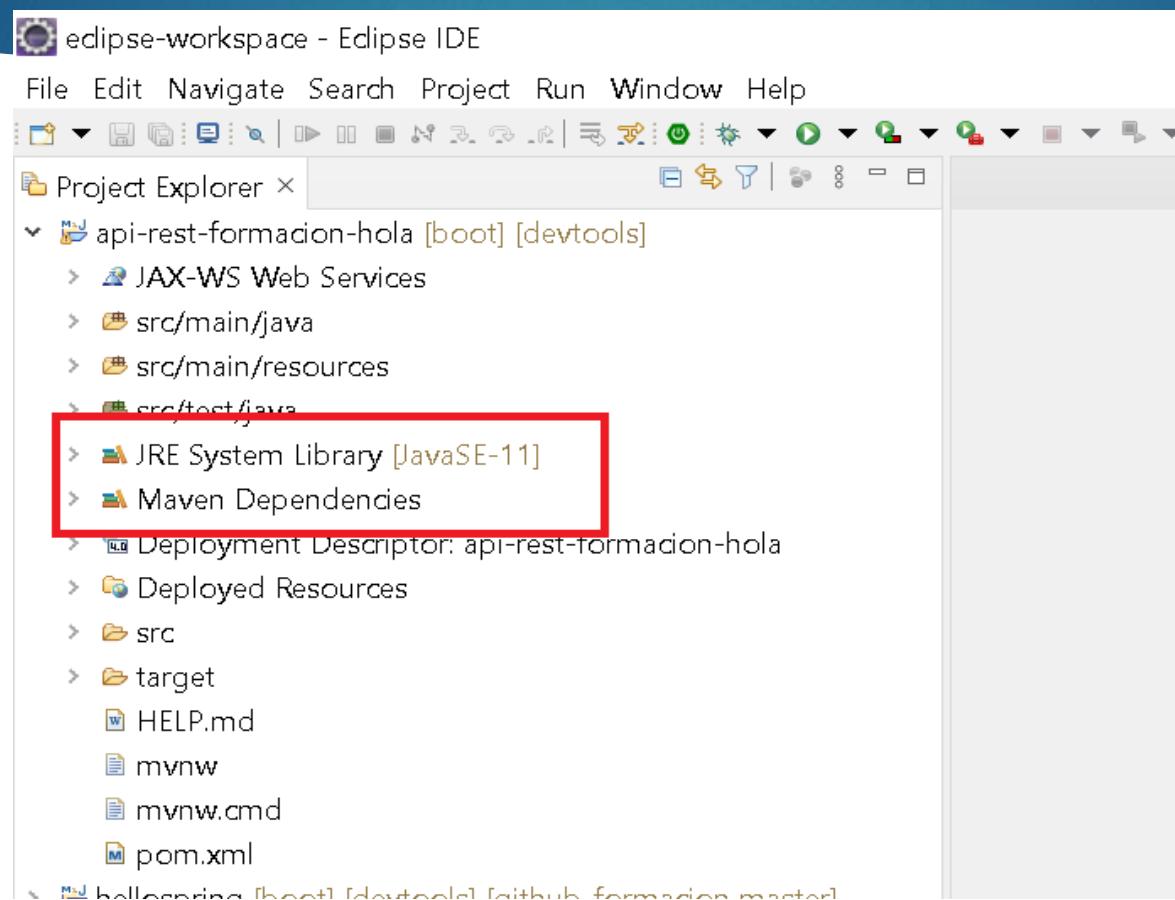
En `src/main/resources` tenemos las carpetas `static`, `templates` donde van los recursos estáticos del proyecto y el archivo `application.properties` donde configuramos conexiones a base de datos, puertos entre otros.



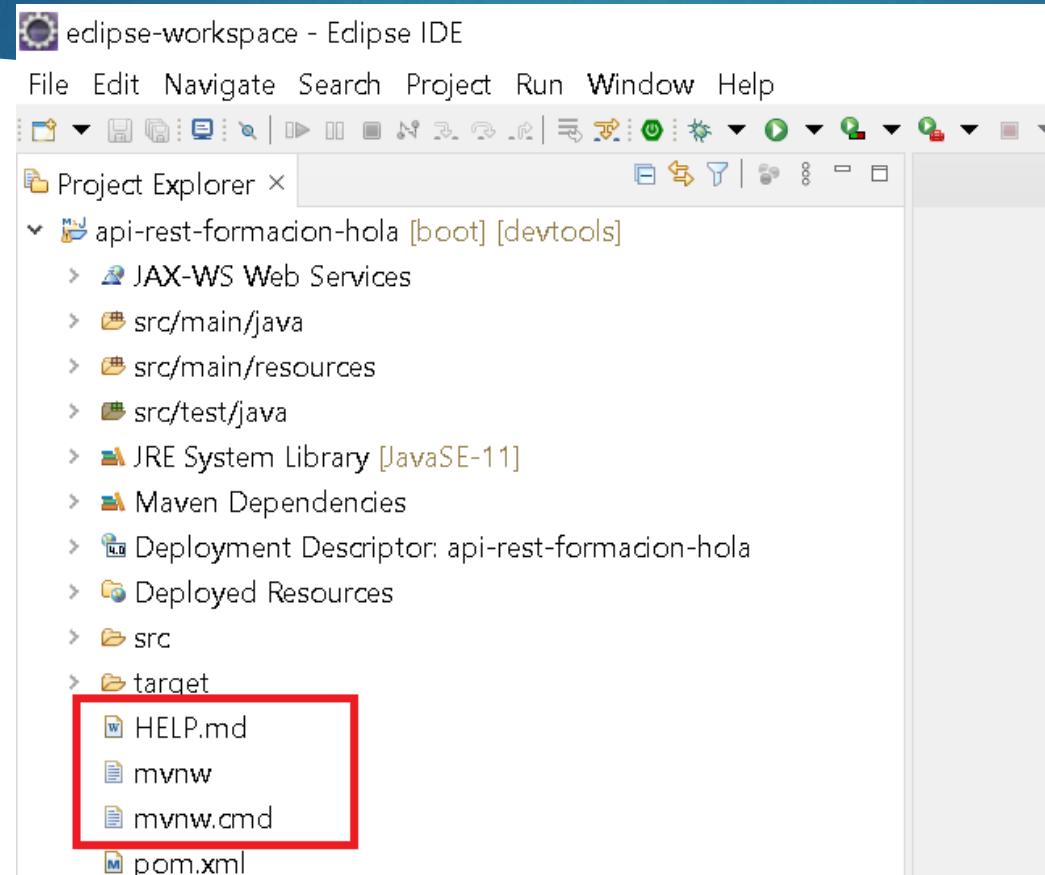
El directorio `src/test/java` se utiliza para los tests, el directorio `target` para compilar los archivos testados.



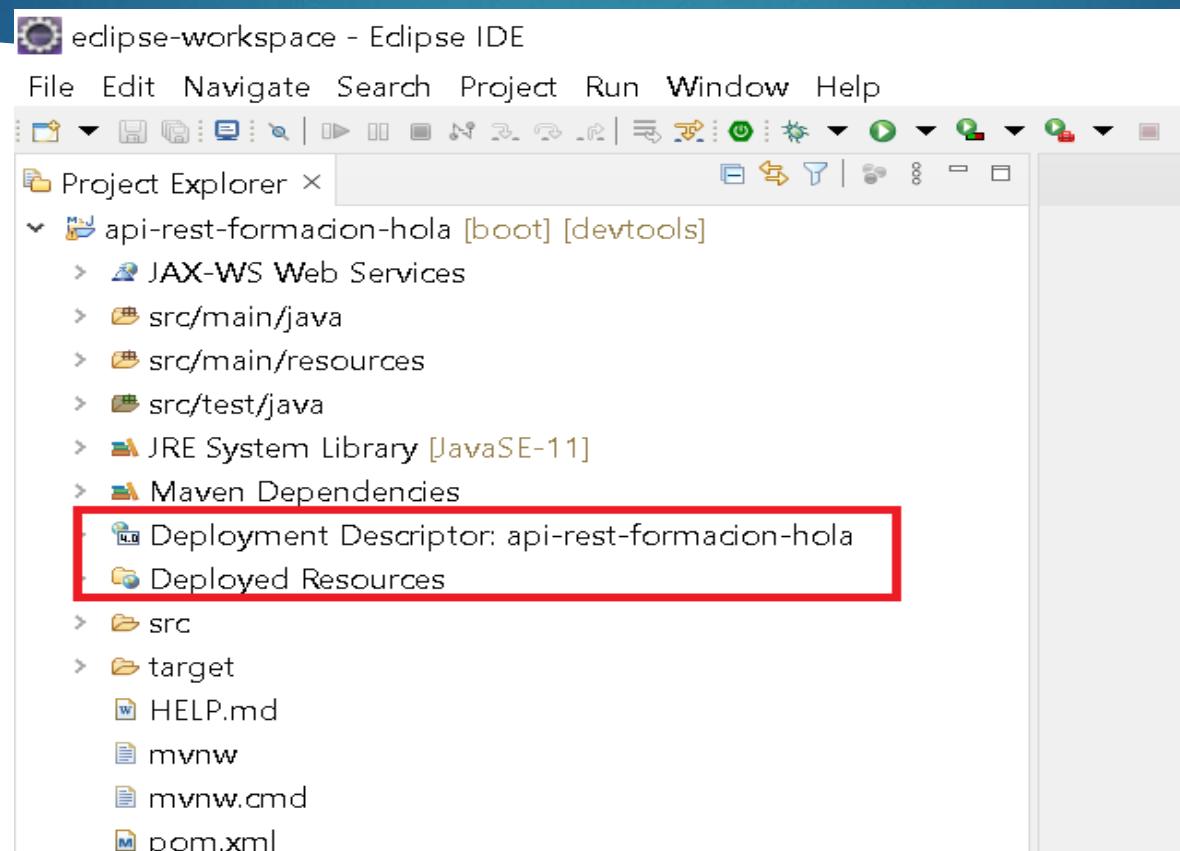
Estas son las librerías de java y dependencias de Maven utilizadas en el proyecto



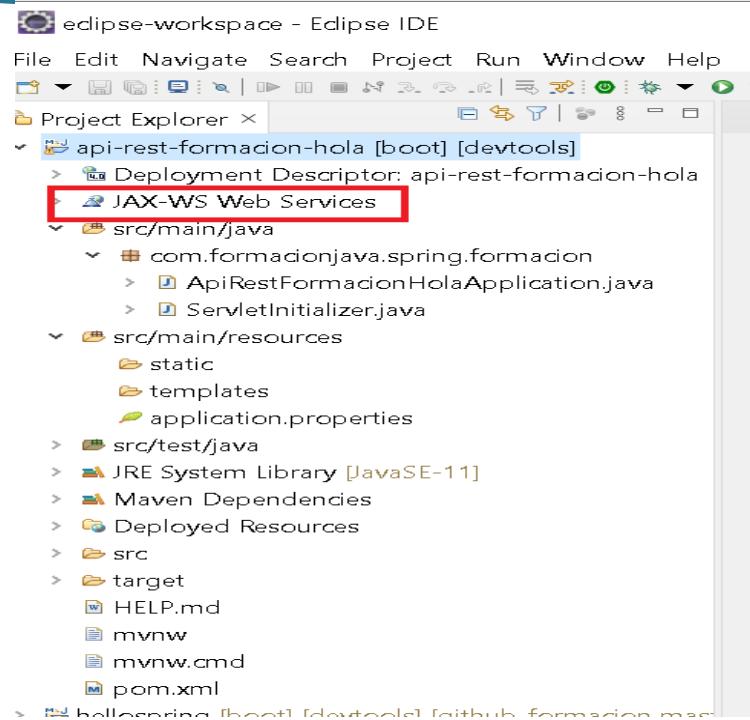
Archivos generados por maven



Archivos autogenerados de registros de deploys



Archivo de configuración de interfaz de servicio web, también para poder implementar protocolo SOAP entre otros.



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer View:** On the left, it lists several projects. The project "springboot-apirest [boot] [devtools]" is selected and highlighted with a red border. Inside this project, the file "SpringbootApirestApplication.java" is open and highlighted.
- Code Editor View:** The main editor displays the code for "SpringbootApirestApplication.java". The code is as follows:

```
1 package com.formacionjava.springboot.apirest;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class SpringbootApirestApplication {
7
8
9     public static void main(String[] args) {
10         SpringApplication.run(SpringbootApirestApplication.class, args);
11     }
12
13 }
14
```

The line `@SpringBootApplication` is highlighted with a red rectangular box. The code editor has a light blue background for the current file and a white background for other files.

Bottom Status Bar: The status bar shows tabs for Markers, Properties, Servers, Data Source Explorer, Snippets, and Console. The Console tab is active, displaying the message: "No consoles to display at this time."

@SpringBootApplication puede usar una sola anotación para habilitar esas tres funciones

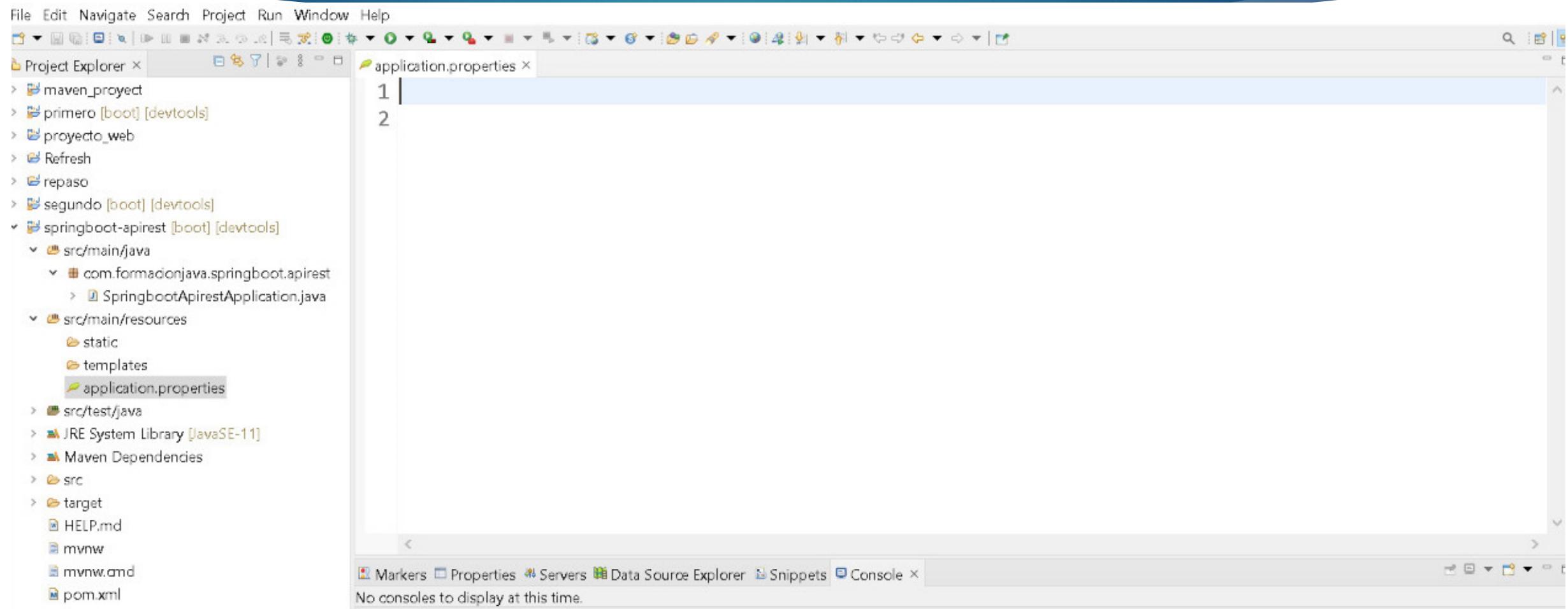
- ▶ `@EnableAutoConfiguration`: habilita el mecanismo de configuración automática de Spring Boot
- ▶ `@ComponentScan`: habilite el `@Componentescaneo` en el paquete donde se encuentra la aplicación (consulte las mejores prácticas)
- ▶ `@Configuration`: permite registrar beans adicionales en el contexto o importar clases de configuración adicionales
- ▶ La `@SpringBootApplication`anotación es equivalente a usar `@Configuration`, `@EnableAutoConfiguration`y `@ComponentScan`con sus atributos predeterminados.

The screenshot shows a Java development environment with the following interface elements:

- Project Explorer**: On the left, it lists several Maven projects. One project, "springboot-apirest [boot] [devtools]", is expanded, revealing its structure: src/main/java, src/main/resources, src/test/java, JRE System Library [JavaSE-11], Maven Dependencies, src, target, HELP.md, mvnw, mvnw.cmd, and pom.xml. This entire section is highlighted with a red rectangular border.
- Code Editor**: The main workspace displays the file `SpringbootApirestApplication.java`. The code is annotated with various annotations:

```
44 * {@code @EnableAutoConfiguration} and {@code @ComponentScan}.
45 *
46 * @author Phillip Webb
47 * @author Stephane Nicoll
48 * @author Andy Wilkinson
49 * @since 1.2.0
50 */
51 @Target(ElementType.TYPE)
52 @Retention(RetentionPolicy.RUNTIME)
53 @Documented
54 @Inherited
55 @SpringBootConfiguration
56 @EnableAutoConfiguration
57 @ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes = TypeExcl}
58                               @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.clas
59 public @interface SpringBootApplication {
60
```
- Toolbars**: Standard Java IDE toolbars are visible at the top.
- Bottom Bar**: Includes tabs for Markers, Properties, Servers, Data Source Explorer, Snippets, and Console, along with a message: "No consoles to display at this time."

Iniciamos la configuración en application.properties se encuentra en src/main/resource

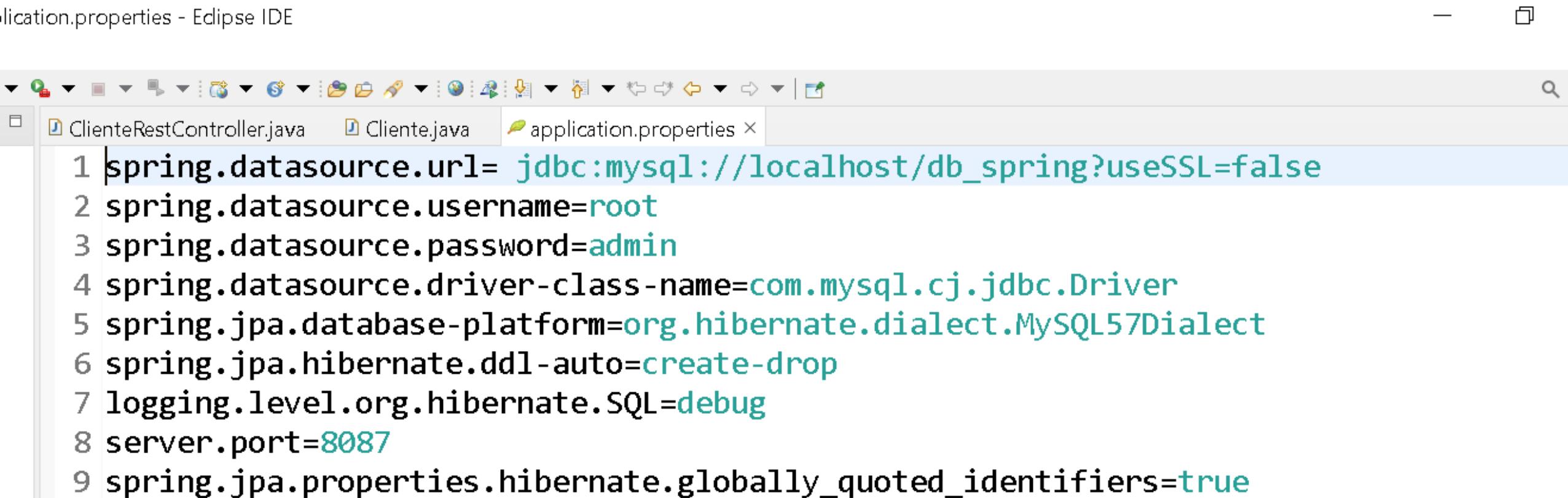


Configuramos con datos para conectarnos a la base de datos

```
application.properties
1 spring.datasource.url= jdbc:mysql://localhost/db_spring
2 spring.datasource.username=root
3 spring.datasource.password=admin
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.database-platform=org.hibernate.dialect.MySQL57Dialect
6 spring.jpa.hibernate.ddl-auto=create-drop
7 logging.level.org.hibernate.SQL=debug
```

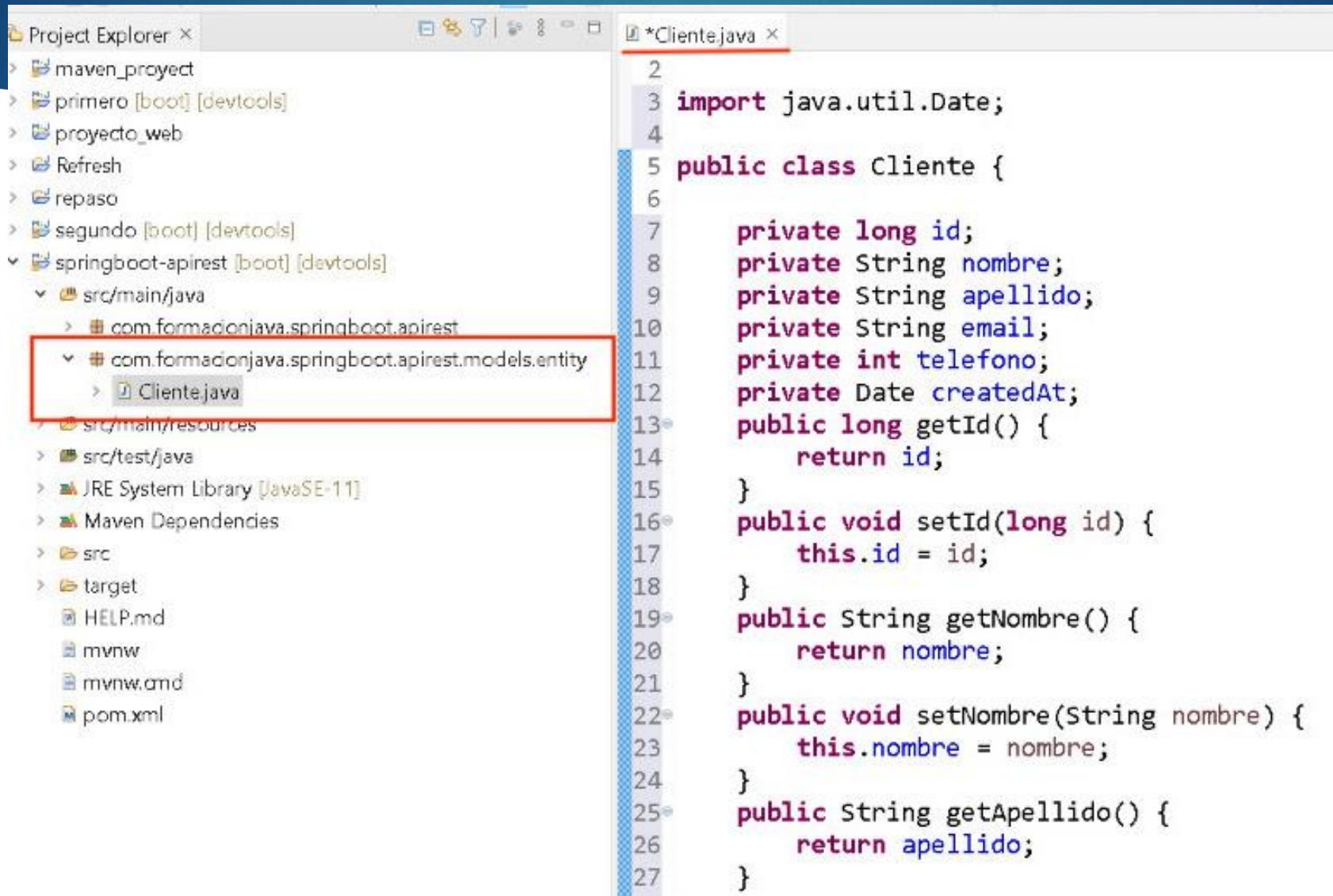
Si tenemos problemas con el puerto, podemos agregar estas dos líneas de código, server.port=8087, spring.jpa.properties.hibernate.globally_quoted_identifiers=true

application.properties - Eclipse IDE



```
1 spring.datasource.url= jdbc:mysql://localhost/db_spring?useSSL=false
2 spring.datasource.username=root
3 spring.datasource.password=admin
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.database-platform=org.hibernate.dialect.MySQL57Dialect
6 spring.jpa.hibernate.ddl-auto=create-drop
7 logging.level.org.hibernate.SQL=debug
8 server.port=8087
9 spring.jpa.properties.hibernate.globally_quoted_identifiers=true
```

Creamos un paquete y una clase para nuestro modelo Cliente con los siguientes atributos y sus correspondientes setters y getters



The screenshot shows a Java development environment with the following details:

- Project Explorer:** Shows the project structure. A red box highlights the package `com.formacionjava.springboot.apirest.models.entity` and the class `Cliente.java`.
- Code Editor:** Displays the `Cliente.java` file with the following code:

```
2 import java.util.Date;
3
4 public class Cliente {
5
6     private long id;
7     private String nombre;
8     private String apellido;
9     private String email;
10    private int telefono;
11    private Date createdAt;
12    public long getId() {
13        return id;
14    }
15    public void setId(long id) {
16        this.id = id;
17    }
18    public String getNombre() {
19        return nombre;
20    }
21    public void setNombre(String nombre) {
22        this.nombre = nombre;
23    }
24    public String getApellido() {
25        return apellido;
26    }
27}
```

Project Explorer

maven_proyect
primero [boot] [devtools]
proyecto_web
Refresh
repasso
segundo [boot] [devtools]
springboot-apirest [boot] [devtools]
src/main/java
 com.formacionjava.springboot.apirest
 com.formacionjava.springboot.apirest.models.entity
 Cliente.java
src/main/resources
src/test/java
JRE System Library [JavaSE-11]
Maven Dependencies
src
target
HELP.md
mvnw
mvnw.cmd
pom.xml

*Cliente.java

```
16°     public void setId(long id) {  
17°         this.id = id;  
18°     }  
19°     public String getNombre() {  
20°         return nombre;  
21°     }  
22°     public void setNombre(String nombre) {  
23°         this.nombre = nombre;  
24°     }  
25°     public String getApellido() {  
26°         return apellido;  
27°     }  
28°     public void setApellido(String apellido) {  
29°         this.apellido = apellido;  
30°     }  
31°     public String getEmail() {  
32°         return email;  
33°     }  
34°     public void setEmail(String email) {  
35°         this.email = email;  
36°     }  
37°     public int getTelefono() {  
38°         return telefono;  
39°     }  
40°     public void setTelefono(int telefono) {  
41°         this.telefono = telefono;
```

Serializable

- ▶ Es una clase ubicada en el paquete **Java.io.Serializable**, la cual no cuenta con ningún método, por lo que es una clase que sirve solamente para especificar que todo el estado de un objeto instanciado podrá ser escrito o enviado en la red como una trama de bytes.
- ▶ Hay que tomar en cuenta que, si un objeto tiene como atributo otro objeto, entonces se debe declarar ese objeto del tipo **Serializable**, y estos serán serializados primero. Esto se puede tomar como un árbol, donde las hojas son objetos que forman parte de otro objeto como un atributo, y todos ellos son marcados como serializables, para así entonces serializar primero las hojas del árbol y finalizar con la raíz.

Implementar Serializable como primer paso para que la clase cliente sea un entity

The screenshot shows an IDE interface with the Project Explorer on the left and the code editor on the right. The code editor displays the `Cliente.java` file, which implements the `Serializable` interface. A tooltip is open over the `implements Serializable` part of the code, providing information about the serializable class and suggesting ways to fix the issue.

```
2
3 import java.io.Serializable;
4 import java.util.Date;
5
6
7 public class Cliente implements Serializable{
8     private long id;
9     private String nombre;
10    private String apellido;
11    private Date fechaNacimiento;
12    private String tipoIdentificacion;
13    private String numeroIdentificacion;
14
15    public long getId() {
16        return id;
17    }
18    public void setId(long id) {
19        this.id = id;
20    }
21    public String getNombre() {
22        return nombre;
23    }
24    public void setNombre(String nombre) {
25        this.nombre = nombre;
26    }
27    public String getApellido() {
```

The tooltip content includes:

- The serializable class Cliente does not declare a static final serialVersionUID field of type long
- 4 quick fixes available:
 - + Add default serial version ID (highlighted)
 - + Add generated serial version ID
 - @ Add @SuppressWarnings('serial') to 'Cliente'
 - Configure problem severity

The screenshot shows a Java application structure in the Project Explorer and the corresponding code in the editor.

Project Explorer:

- maven_project
- primero [boot] [devtools]
- projeto_web
- Refresh
- repaso
- segundo [boot] [devtools]
- springboot-apirest [boot] [devtools]
- src/main/java
 - com.formacionjava.springboot.apirest
 - com.formacionjava.springboot.apirest.models.entity
 - Cliente.java
- src/main/resources
- src/test/java
- JRE System Library [JavaSE-11]
- Maven Dependencies
- src
- target
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml

Code Editor (Cliente.java):

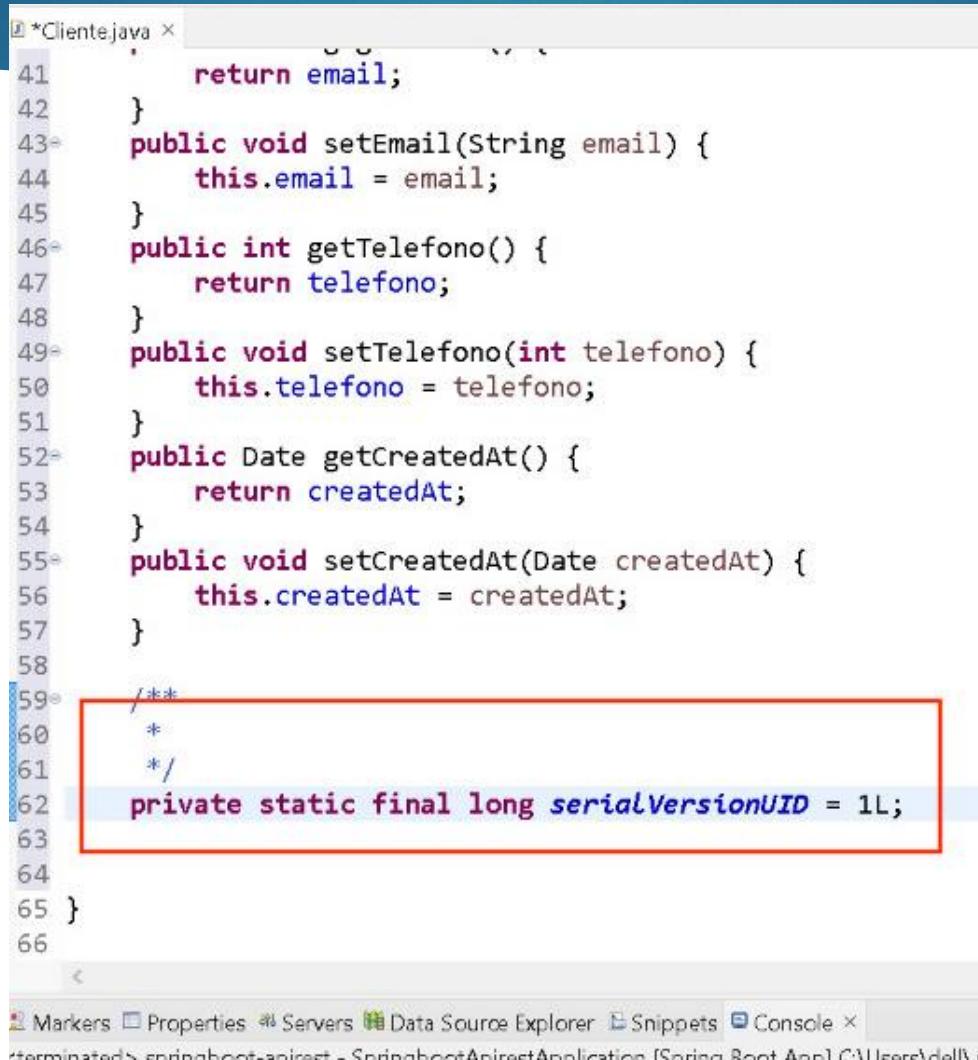
```
1 package com.formacionjava.springboot.apirest.models.entity;
2
3 import java.io.Serializable;
4 import java.util.Date;
5
6 import javax.persistence.Entity;
7 import javax.persistence.Table;
8
9
10 @Entity
11 @Table(name = "clientes")
12 public class Cliente implements Serializable{
13
14     /**
15      *
16     */
17     private static final long serialVersionUID = 1L;
18
19     private long id;
20     private String nombre;
21     private String apellido;
22     private String email;
23     private int telefono;
24     private Date createdAt;
25     public long getId() {
26         return id;
27     }
28 }
```

The line `private static final long serialVersionUID = 1L;` is highlighted with a red rectangle.

serialVersionUID

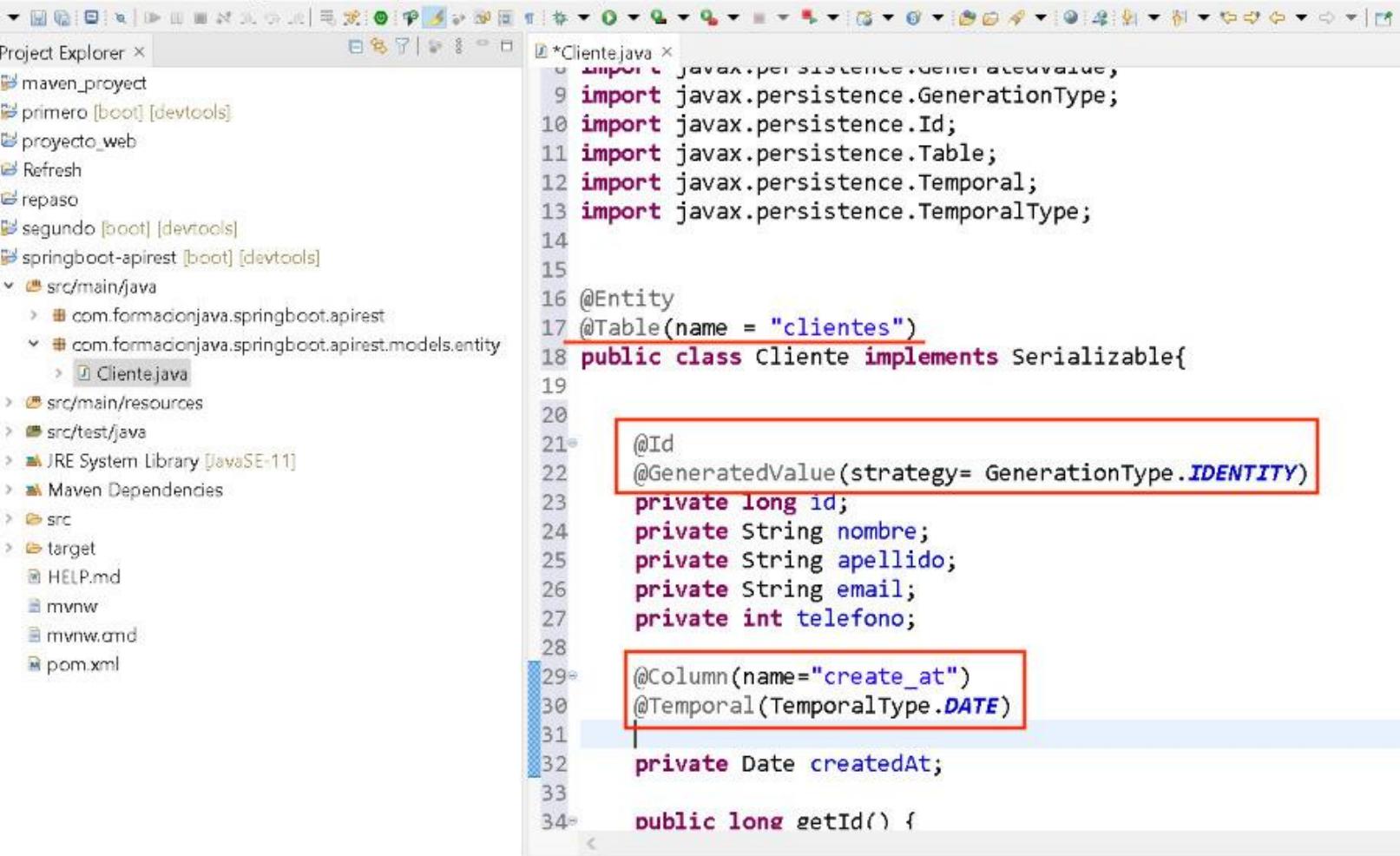
- ▶ El tiempo de ejecución de serialización asocia con cada clase serializable un número de versión, llamada serialVersionUID, que se usa durante la deserialización para verificar que el remitente el receptor de un objeto serializado hayan cargado clases para ese objeto que sean compatibles con respecto a la serialización. Si el receptor ha cargado una clase para el objeto que tiene una clase diferente serialVersionUID a la del remitente correspondiente, entonces la deserialización resultará en un InvalidClassException. Una clase serializable puede declarar la suya serialVersionUID explícitamente declarando un campo con nombre serialVersionUID que debe ser estático, final y de tipo long

Pasamos serialVersionUID a final de la clase



```
41         return email;
42     }
43     public void setEmail(String email) {
44         this.email = email;
45     }
46     public int getTelefono() {
47         return telefono;
48     }
49     public void setTelefono(int telefono) {
50         this.telefono = telefono;
51     }
52     public Date getCreatedAt() {
53         return createdAt;
54     }
55     public void setCreatedAt(Date createdAt) {
56         this.createdAt = createdAt;
57     }
58
59     /**
60      *
61      */
62     private static final long serialVersionUID = 1L;
63
64
65 }
```

Agregamos las siguientes anotaciones



The screenshot shows an IDE interface with the Project Explorer on the left and the code editor on the right. The code editor displays a Java class named Cliente.java. Two specific annotations are highlighted with red boxes:

```
*Cliente.java x
1 import java.util.Date;
2 import javax.persistence.GeneratedValue;
3 import javax.persistence.GenerationType;
4 import javax.persistence.Id;
5 import javax.persistence.Table;
6 import javax.persistence.Temporal;
7 import javax.persistence.TemporalType;
8
9
10 @Entity
11 @Table(name = "clientes")
12 public class Cliente implements Serializable{
13
14     @Id
15     @GeneratedValue(strategy= GenerationType.IDENTITY)
16     private long id;
17     private String nombre;
18     private String apellido;
19     private String email;
20     private int telefono;
21
22     @Column(name="create_at")
23     @Temporal(TemporalType.DATE)
24
25     private Date createdAt;
26
27     public long getId() {
28
29
30
31
32
33
34 }
```

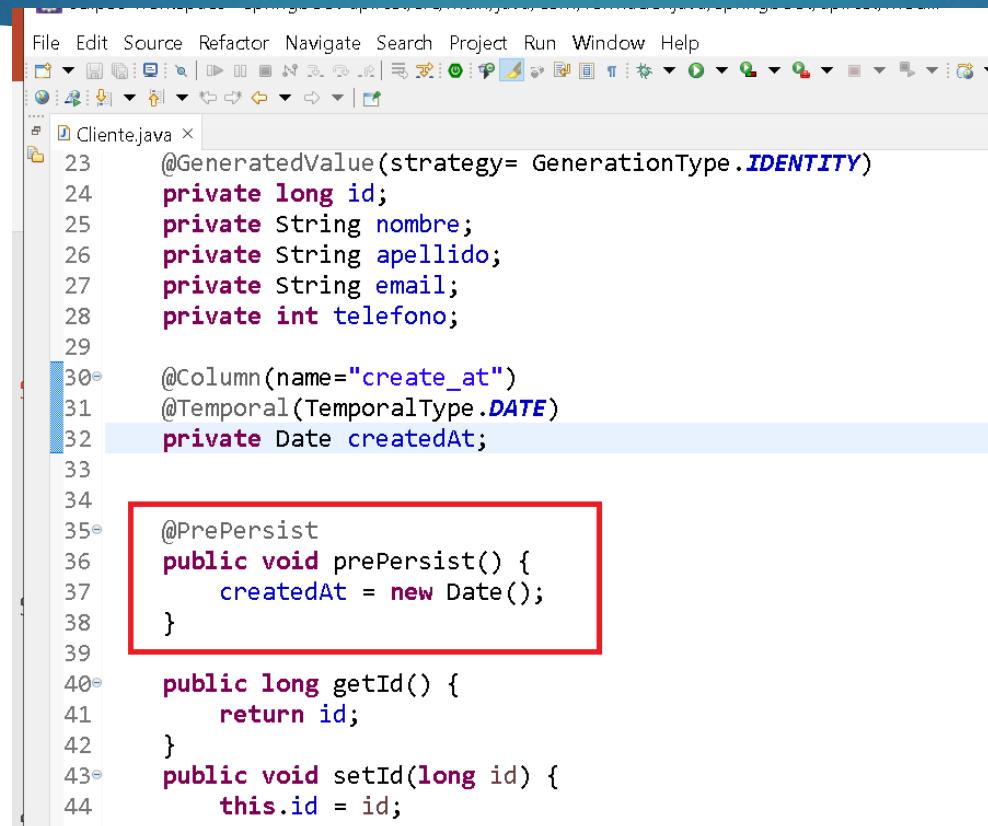
Que es una entidad

- ▶ Una entidad en Java es un objeto de persistencia.
- ▶ La persistencia es la habilidad de una aplicación para mantener(persistir) y recuperar información de sistemas de almacenamiento no volátiles.
- ▶ Una entidad representa una tabla en una base de datos, y cada instancia de entidad corresponde a una fila en la tabla.
- ▶ El estado de una entidad se representa por campos de persistencia o propiedades de persistencia.

Anotaciones

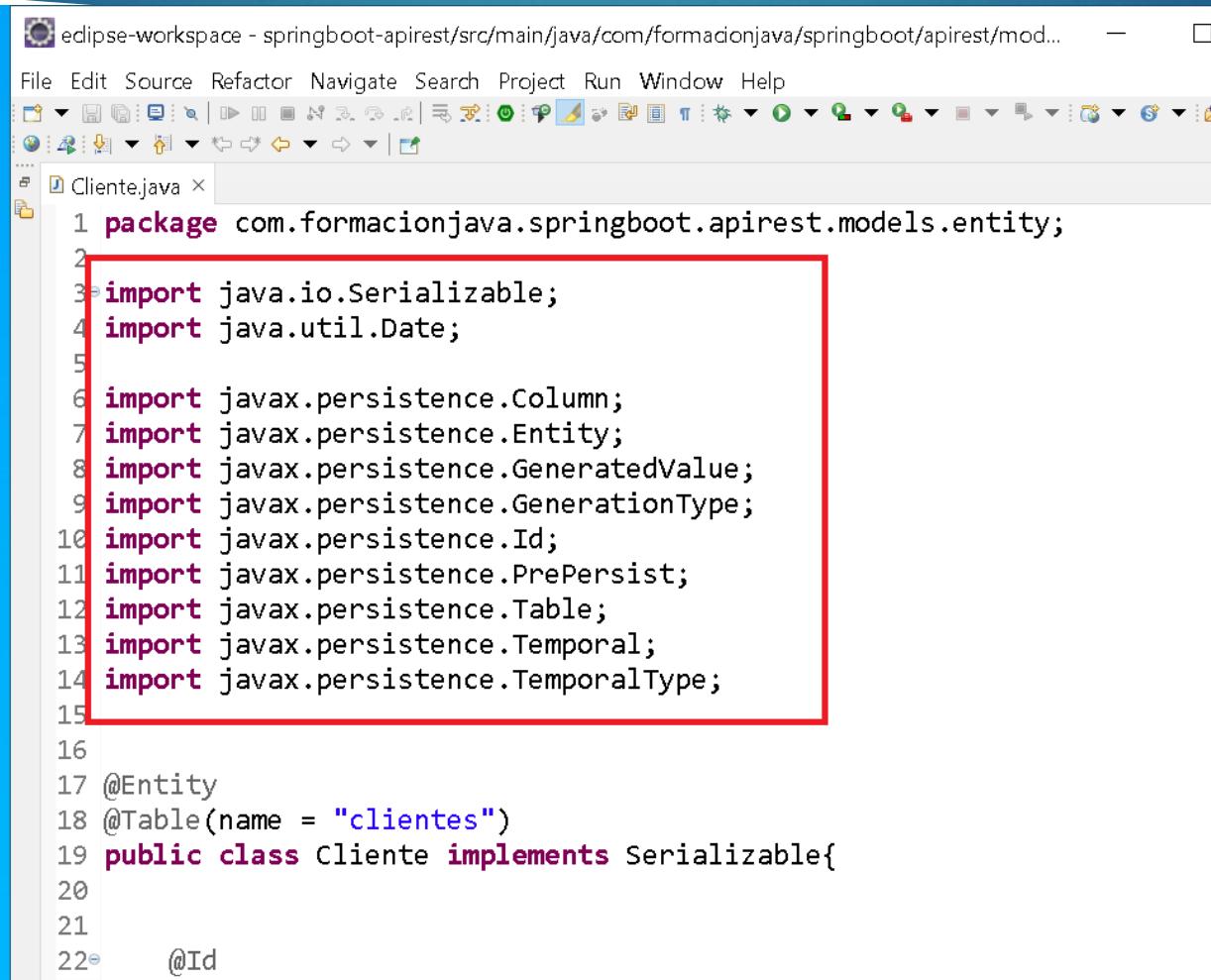
- ▶ **@Entity** Especifica que una clase es una entidad.
- ▶ **@Table** anotación especifica del nombre de la tabla de base datos que se utilizara para el mapeo.
- ▶ **@Column** se utiliza para identificar la correspondencia entre atributos en las clases de entidad y campos en las tablas de datos.
- ▶ **@Id** Especifica la llave primaria de una entidad.
- ▶ **@GeneratedValue** proporciona la especificación de estrategias de generación para los valores de las claves primarias.

Agregamos el siguiente método para que nos tome la fecha actual



```
File Edit Source Refactor Navigate Search Project Run Window Help
Client.java x
23     @GeneratedValue(strategy= GenerationType.IDENTITY)
24     private long id;
25     private String nombre;
26     private String apellido;
27     private String email;
28     private int telefono;
29
30     @Column(name="create_at")
31     @Temporal(TemporalType.DATE)
32     private Date createdAt;
33
34
35     @PrePersist
36     public void prePersist() {
37         createdAt = new Date();
38     }
39
40     public long getId() {
41         return id;
42     }
43     public void setId(long id) {
44         this.id = id;
```

A tener en cuenta estos imports en la clase



```
edipse-workspace - springboot-apirest/src/main/java/com/formacionjava/springboot/apirest/mod...
File Edit Source Refactor Navigate Search Project Run Window Help
Cliente.java x
1 package com.formacionjava.springboot.apirest.models.entity;
2
3 import java.io.Serializable;
4 import java.util.Date;
5
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.GeneratedValue;
9 import javax.persistence.GenerationType;
10 import javax.persistence.Id;
11 import javax.persistence.PrePersist;
12 import javax.persistence.Table;
13 import javax.persistence.Temporal;
14 import javax.persistence.TemporalType;
15
16
17 @Entity
18 @Table(name = "clientes")
19 public class Cliente implements Serializable{
20
21
22     @Id
```

Compilamos y verificamos que nos cree la tabla solicitada en entity

The screenshot shows the Eclipse IDE interface with the following details:

- Java Code:** A snippet of Java code is shown at the top:

```
11 import javax.persistence.Table;
```
- Console Output:** The main part of the screenshot displays the command-line output from the application's startup. The output is timestamped and shows various INFO and DEBUG logs from different components of the Spring Boot application, including the Spring framework, Hibernate, and Tomcat. The logs indicate the application is starting up, scanning for repositories, initializing services, and connecting to a database. A specific log entry at the bottom shows the creation of a table named 'clientes':

```
3:18:21.662 DEBUG 8204 --- [ restartedMain] org.hibernate.SQL : create table clientes (id bigint not null auto_increm
```

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

db_spring

- Tables
 - clientes
 - Columns
 - id
 - apellido
 - create_at
 - email
 - nombre
 - telefono
 - Indexes
 - Foreign Keys
 - Triggers
 - Views
 - Stored Procedures
 - Functions

prueba_db

Administration Schemas

Information

Schema: db_spring

Query 1 Administration - Startup / Shutdown

Local instance MySQL80

Startup / Shutdown MySQL Server

The database server is started and ready for client connections. To shut the server down, use the "Stop Server" button.

The database server instance is **running** [Stop Server](#) [Bring Offline](#)

If you stop the server, neither you nor your applications can use the database and all current connections will be closed.

Startup Message Log

```
2021-11-18 22:30:24 - Workbench will use cmd shell commands to start/stop this instance
2021-11-18 22:30:24 - Server is running
2021-11-18 23:04:29 - Checking server status...
2021-11-18 23:04:29 - MySQL server is currently running
2021-11-18 23:04:30 - Checking server status...
2021-11-18 23:04:30 - MySQL server is currently running
2021-11-18 23:18:54 - Checking server status...
2021-11-18 23:18:54 - MySQL server is currently running
```

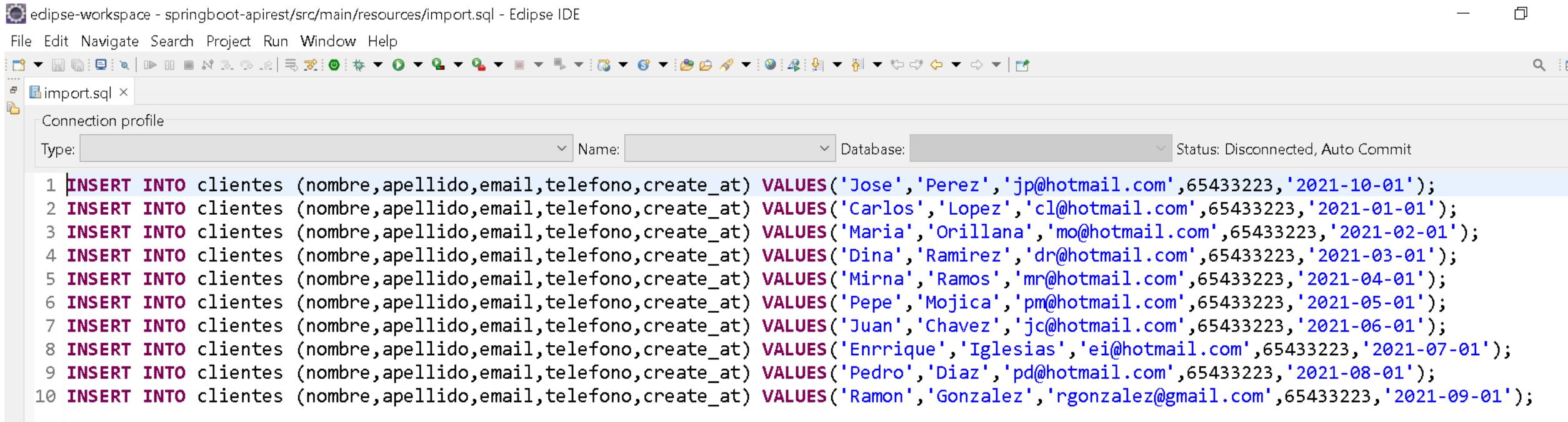
Refresh Status

Action Output

#	Time	Action	Message
---	------	--------	---------

Añadir datos de prueba

- ▶ Dentro de la carpeta src/main/resources creamos un archivo llamado import.sql y de esta forma una vez compilamos Spring de forma automática leerá este archivo y realizara el insert de los datos de prueba



The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace - springboot-apirest/src/main/resources/import.sql - Eclipse IDE". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Find, and Run. The left sidebar shows a project tree with "import.sql x" selected. The main editor area displays the following SQL code:

```
1 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Jose','Perez','jp@hotmail.com',65433223,'2021-10-01');
2 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Carlos','Lopez','cl@hotmail.com',65433223,'2021-01-01');
3 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Maria','Orillana','mo@hotmail.com',65433223,'2021-02-01');
4 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Dina','Ramirez','dr@hotmail.com',65433223,'2021-03-01');
5 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Mirna','Ramos','mr@hotmail.com',65433223,'2021-04-01');
6 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Pepe','Mojica','pm@hotmail.com',65433223,'2021-05-01');
7 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Juan','Chavez','jc@hotmail.com',65433223,'2021-06-01');
8 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Enrrique','Iglesias','ei@hotmail.com',65433223,'2021-07-01');
9 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Pedro','Diaz','pd@hotmail.com',65433223,'2021-08-01');
10 INSERT INTO clientes (nombre,apellido,email,telefono,create_at) VALUES('Ramon','Gonzalez','rgonzalez@gmail.com',65433223,'2021-09-01');
```

Compilamos y verificamos que se inserta los datos de prueba

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database schema. A red box highlights the 'db_spring' schema, which contains a 'Tables' folder and the 'clientes' table.
- Query Editor:** Displays the SQL query: `SELECT * FROM db_spring.clientes;`
- Result Grid:** Shows the data from the 'clientes' table. A red box highlights the entire grid. The data is as follows:

	id	apellido	create_at	email	nombre	telefono
1	1	Perez	2021-10-01	jp@hotmail.com	Jose	65433223
2	2	Lopez	2021-01-01	cl@hotmail.com	Carlos	65433223
3	3	Orillana	2021-02-01	mo@hotmail.com	Maria	65433223
4	4	Ramirez	2021-03-01	dr@hotmail.com	Dina	65433223
5	5	Ramos	2021-04-01	mr@hotmail.com	Mirna	65433223
6	6	Mojica	2021-05-01	pm@hotmail.com	Pepe	65433223
7	7	Chavez	2021-06-01	jc@hotmail.com	Juan	65433223
8	8	Iglesias	2021-07-01	ei@hotmail.com	Enrique	65433223
9	9	Diaz	2021-08-01	pd@hotmail.com	Pedro	65433223
10	10	Gonzalez	2021-09-01	rgonzalez@gmail.com	Ramon	65433223

Creamos el paquete y la interfaz para utilizar CrudRepository

The screenshot shows a Java code editor in an IDE with the following details:

- Project Explorer:** Shows the project structure with several Maven projects and their subfolders.
- Current File:** *ClienteDao.java
- Code Content:**

```
1 package com.formacionjava.springboot.apirest.models.dao;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import com.formacionjava.springboot.apirest.models.entity.Cliente;
6
7 public interface ClienteDao extends CrudRepository<Cliente, Long>{
8
9 }
```
- Tooltips/Information:** A tooltip for the `CrudRepository` interface is displayed, providing its definition, annotations (`@NoRepositoryBean`), and author information (Oliver Gierke, Eberhard Wolff, Jens Schauder).
- Bottom Status Bar:** A message "Press 'F2' for focus" is visible.

CrudRepository

- ▶ CrudRepository implementa operaciones CRUD básicas, que incluyen contar, eliminar, deleteById, save, saveAll, findById y findAll. Lo podemos implementar gracias a SpringDataJPA
- ▶ DAO significa objeto de acceso a datos. Por lo general, la clase DAO es responsable de dos conceptos. Encapsular los detalles de la capa de persistencia y proporcionar una interfaz CRUD para una sola entidad.

Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer x

maven_proyect

primero [boot] [devtools]

proyecto_web

Refresh

repaso

segundo [boot] [devtools]

springboot-apirest [boot] [devtools]

src/main/java

> com.formacionjava.springboot.apirest

> com.formacionjava.springboot.apirest.models.dao

> ClienteDao.java

> com.formacionjava.springboot.apirest.models.entity

src/main/resources

src/test/java

JRE System Library [JavaSE-11]

Maven Dependencies

src

target

HELP.md

mvnw

mvnw.cmd

pom.xml

* ClienteDao.java CrudRepository.class x

```
2* * Copyright 2008-2021 the original author or authors.
16 package org.springframework.data.repository;
17
18 import java.util.Optional;
19
20 /**
21  * Interface for generic CRUD operations on a repository for a specific type.
22 *
23 * @author Oliver Gierke
24 * @author Eberhard Wolff
25 * @author Jens Schauder
26 */
27 @NoRepositoryBean
28 public interface CrudRepository<T, ID> extends Repository<T, ID> {
29
30 /**
31 * Saves a given entity. Use the returned instance for further operations as the same
32 * entity instance completely.
33 *
34 * @param entity must not be {@literal null}.
35 * @return the saved entity; will never be {@literal null}.
36 * @throws IllegalArgumentException in case the given {@literal entity} is {@literal null}.
37 */
```

Markers Properties Servers Data Source Explorer Snippets Console x

```
springboot-apirest - SpringbootApirestApplication [Spring Boot App] C:\Users\dell\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86
3:18:18.347 INFO 8204 --- [ restartedMain] c.t.s.a.SpringbootApirestApplication      : Star
3:18:18.348 INFO 8204 --- [ restartedMain] c.f.s.a.SpringbootApirestApplication      : No a
3:18:18.412 INFO 8204 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devt
```

Creamos los servicios en el siguiente paquete y la interfaz

The screenshot shows the Eclipse IDE interface with the following details:

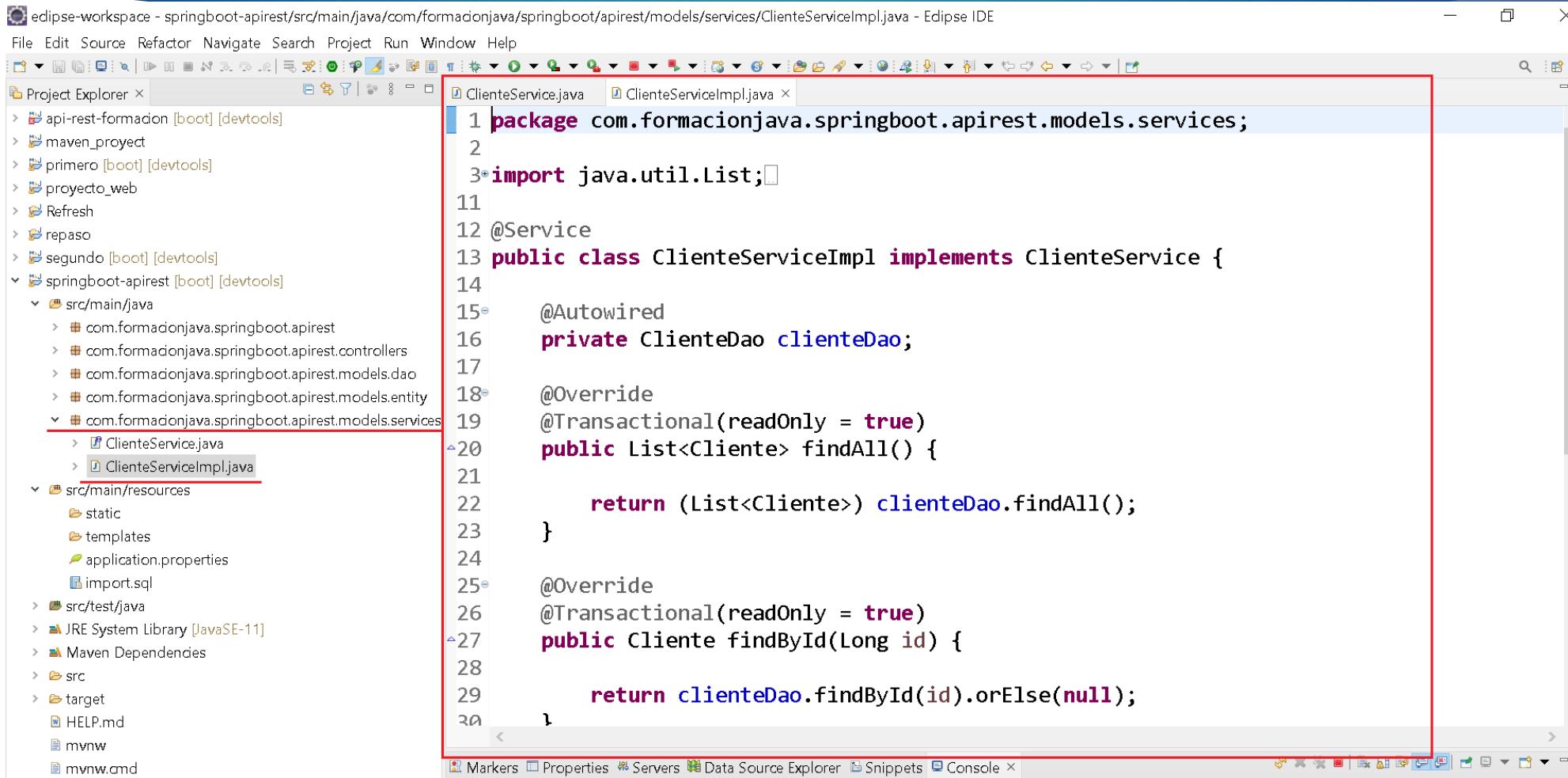
- Title Bar:** eclipse-workspace - springboot-apirest/src/main/java/com/formacionjava/springboot/apirest/models/services/ClienteService.java - Eclipse IDE
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Eclipse toolbar icons.
- Project Explorer:** Shows several projects: api-rest-formacion [boot] [devtools], maven_proyect, primero [boot] [devtools], proyecto_web, Refresh, repaso, segundo [boot] [devtools], and springboot-apirest [boot] [devtools]. The springboot-apirest project is expanded, showing its structure: src/main/java, src/main/resources, and src/test/java. Under src/main/java, there is a package com.formacionjava.springboot.apirest.models.services which contains ClienteService.java and ClienteServiceImpl.java. Under src/main/resources, there are static, templates, application.properties, and import.sql files. Under src/test/java, there is JRE System Library [JavaSE-11] and Maven Dependencies.
- Code Editor:** The code editor window displays the contents of ClienteService.java, which is highlighted with a red border. The code defines a public interface ClienteService with methods for finding all clients, finding a client by ID, saving a client, and deleting a client.
- Bottom Bar:** Standard Eclipse bottom toolbar icons.

```
package com.formacionjava.springboot.apirest.models.services;
import java.util.List;
import com.formacionjava.springboot.apirest.models.entity.Cliente;
public interface ClienteService {
    public List<Cliente> findAll();
    public Cliente findById(Long id);
    public Cliente save(Cliente cliente);
    public void delete(Long id);
}
```

ClienteService

- ▶ Implementamos aquí todos los métodos que utilizaremos en nuestra api rest

Creamos la clase ClienteServiceImpl que implementara la interfaz ClienteService y definiendo los métodos



```
1 package com.formacionjava.springboot.apirest.models.services;
2
3 import java.util.List;
4
5 @Service
6 public class ClienteServiceImpl implements ClienteService {
7
8     @Autowired
9     private ClienteDao clienteDao;
10
11     @Override
12     @Transactional(readOnly = true)
13     public List<Cliente> findAll() {
14
15         return (List<Cliente>) clienteDao.findAll();
16     }
17
18     @Override
19     @Transactional(readOnly = true)
20     public Cliente findById(Long id) {
21
22         return clienteDao.findById(id).orElse(null);
23     }
24 }
```

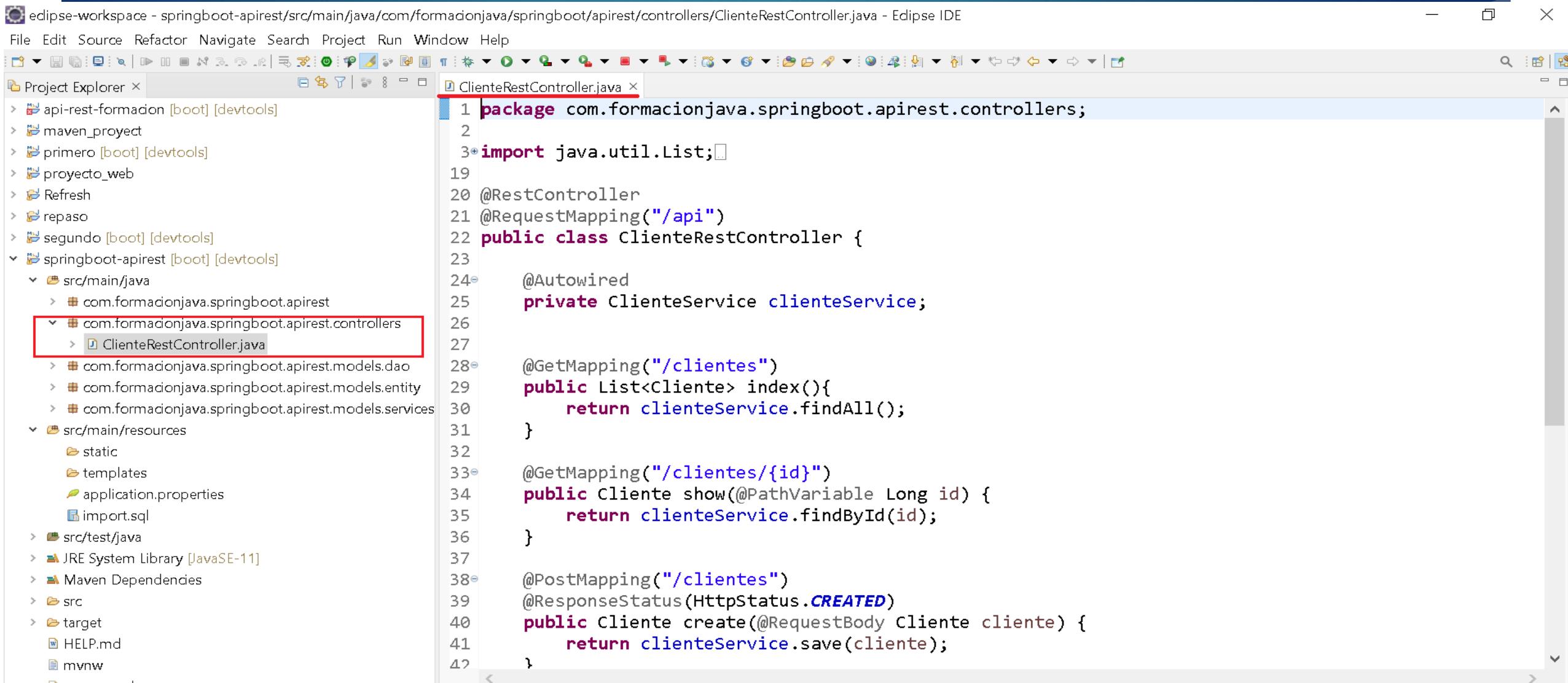
The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure. The `springboot-apirest` project is expanded, revealing its `src/main/java` directory which contains several packages and files, including `ClienteService.java` and `ClienteServiceImpl.java`.
- Code Editor:** The `ClienteServiceImpl.java` file is open. The code implements a service layer for managing Clientes. It includes three methods:
 - `public Cliente findById(Long id)`: Returns a Cliente by its ID, using the `clienteDao.findById(id).orElse(null)` pattern.
 - `public Cliente save(Cliente cliente)`: Returns the saved Cliente after calling `clienteDao.save(cliente)`.
 - `public void delete(Long id)`: Deletes a Cliente by its ID using the `clienteDao.deleteById(id)` method.
- Annotations:** The code uses `@Override`, `@Transactional`, and `@Override` annotations.
- Toolbars and Status Bar:** Standard Eclipse toolbars are visible at the top. The status bar at the bottom displays the project name (`springboot-apirest`), application type (`Spring Boot App`), and system information (`C:\Users\dell\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v202107`).

Las anotaciones utilizadas

- ▶ @Autowired es una de las anotaciones más habituales cuando trabajamos con Spring Framework ya que se trata de la anotación que permite injectar unas dependencias con otras dentro de Spring
- ▶ @Transactional readonly nos provee de las capacidades de inyección de gestión transaccional distribuida y transparente para nuestros repositorios y servicios
- ▶ La pregunta que tenemos que hacernos . ¿En algún momento vamos a solicitar una modificación de la lista?. Es bastante evidente que en el 95% de las ocasiones no y por lo tanto podemos marcar con Spring @Transactional readonly el método para que no guarde el estado y tengamos un mejor rendimiento de la consulta

Por ultimo creamos el siguiente paquete para el controlador ClienteRestController



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** eclipse-workspace - springboot-apirest/src/main/java/com/formacionjava/springboot/apirest/controllers/ClienteRestController.java - Eclipse IDE
- Menu Bar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Toolbar:** Standard Eclipse toolbar with various icons for file operations.
- Project Explorer:** Shows the project structure:
 - api-rest-formacion [boot] [devtools]
 - maven_proyect
 - primero [boot] [devtools]
 - projeto_web
 - Refresh
 - repaso
 - segundo [boot] [devtools]
 - springboot-apirest [boot] [devtools]
 - src/main/java
 - com.formacionjava.springboot.apirest
 - com.formacionjava.springboot.apirest.controllers
 - ClienteRestController.java (highlighted with a red box)
 - com.formacionjava.springboot.apirest.models.dao
 - com.formacionjava.springboot.apirest.models.entity
 - com.formacionjava.springboot.apirest.models.services
 - src/main/resources
 - static
 - templates
 - application.properties
 - import.sql
 - src/test/java
 - JRE System Library [JavaSE-11]
 - Maven Dependencies
 - src
 - target
 - HELP.md
 - mvnw
- Editor:** The code for `ClienteRestController.java` is displayed:

```
1 package com.formacionjava.springboot.controllers;
2
3 import java.util.List;
4
5 @RestController
6 @RequestMapping("/api")
7 public class ClienteRestController {
8
9     @Autowired
10    private ClienteService clienteService;
11
12
13    @GetMapping("/clientes")
14    public List<Cliente> index(){
15        return clienteService.findAll();
16    }
17
18    @GetMapping("/clientes/{id}")
19    public Cliente show(@PathVariable Long id) {
20        return clienteService.findById(id);
21    }
22
23    @PostMapping("/clientes")
24    @ResponseStatus(HttpStatus.CREATED)
25    public Cliente create(@RequestBody Cliente cliente) {
26        return clienteService.save(cliente);
27    }
28}
```

eclipse-workspace - springboot-apirest/src/main/java/com/formacionjava/springboot/apirest/controllers/ClienteRestController.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer x ClienteRestController.java x

```
35     return clienteService.findById(id);  
36 }  
37  
38 @PostMapping("/clientes")  
39 @ResponseStatus(HttpStatus.CREATED)  
40 public Cliente create(@RequestBody Cliente cliente) {  
41     return clienteService.save(cliente);  
42 }  
43  
44 @PutMapping("/clientes/{id}")  
45 @ResponseStatus(HttpStatus.CREATED)  
46 public Cliente update(@RequestBody Cliente cliente, @PathVariable Long id) {  
47     Cliente clienteUpdate = clienteService.findById(id);  
48  
49     clienteUpdate.setApellido(cliente.getApellido());  
50     clienteUpdate.setNombre(cliente.getNombre());  
51     clienteUpdate.setEmail(cliente.getEmail());  
52  
53     return clienteService.save(clienteUpdate);  
54 }  
55  
56 @DeleteMapping("clientes/{id}")  
57 public void delete(@PathVariable Long id) {  
58     clienteService.delete(id);  
59 }  
60 }  
61
```

Markers Properties Servers Data Source Explorer Snippets Console x

Anotaciones utilizadas

- ▶ **@Controller** . Esto es simplemente una especialización de la clase `@Component` , que nos permite detectar automáticamente las clases de implementación a través del escaneo de classpath. Por lo general, usamos `@Controller` en combinación con una anotación `@RequestMapping` para los métodos de manejo de solicitudes.
- ▶ **@RequestMapping** En pocas palabras, la anotación se utiliza para asignar solicitudes web a los métodos de Spring Controller.
- ▶ **@GetMapping** es una versión especializada de `@RequestMapping` anotación actúa como un atajo para `@RequestMapping(method = RequestMethod.GET)`
- ▶ **@PostMapping** es una versión de `@RequestMapping` anotación que actúa como un acceso directo a `@RequestMapping(method = RequestMethod.POST)`
- ▶ **@PathVariable** se puede usar para manejar variables de plantilla en la asignación de URI de solicitud
- ▶ **@RequestBody** asigna el cuerpo `HttpRequest` a un objeto de transferencia o dominio, lo que permite la deserialización automática del cuerpo `HttpRequest` entrante en un objeto Java.
- ▶ **@ResponseStatus** marca un método o clase de excepción con el código de estado y el mensaje de motivo de devolverse. El código de estado se aplica a la respuesta HTTP cuando se invoca el método controlador o siempre que se lanza la excepción específica.

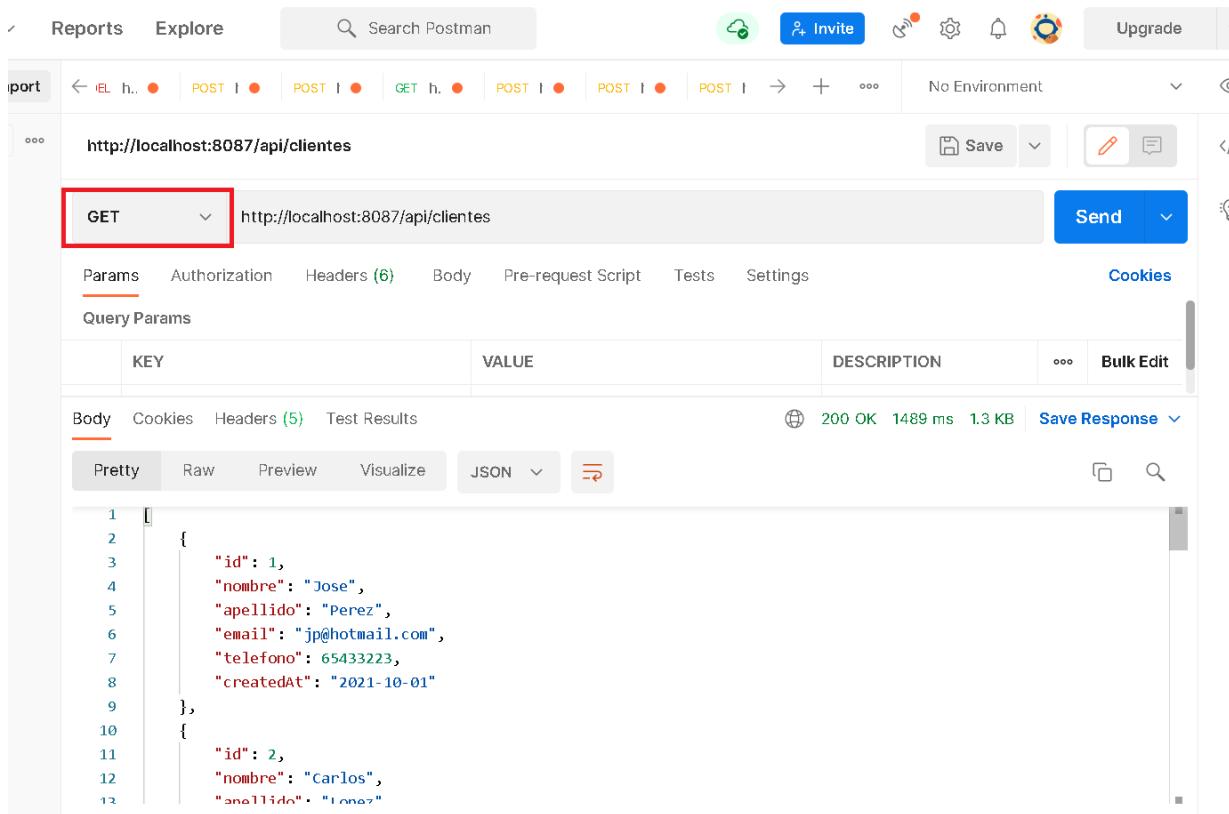
Las url de las apis que hemos creado

- ▶ GET <http://localhost:8087/api/clientes>
- ▶ POST <http://localhost:8087/api/clientes>
- ▶ GET <http://localhost:8087/api/clientes/{id}>
- ▶ DELETE <http://localhost:8087/api/clientes/{id}>

- ▶ OBSERVACION: EL PUERTO SERIA EL QUE ESTE FUNCIONANDO PARA USTEDES

Lo probamos con Postman

► PROBAR API GET



The screenshot displays the Postman application interface. At the top, there are navigation links for 'Reports' and 'Explore', a search bar, and various status indicators. Below the header, a list of recent requests is shown, followed by a 'No Environment' message. The main workspace shows a 'GET' request to 'http://localhost:8087/api/clientes'. The 'Params' tab is selected, showing a table for 'Query Params'. The 'Body' tab is also visible. The response section shows a 200 OK status with a response time of 1489 ms and a size of 1.3 KB. The response body is displayed in a JSONpretty-printed format:

```
1 [ { 2   "id": 1, 3   "nombre": "Jose", 4   "apellido": "Perez", 5   "email": "jp@hotmail.com", 6   "telefono": 65433223, 7   "createdAt": "2021-10-01" 8 }, 9 { 10    "id": 2, 11    "nombre": "Carlos", 12    "apellido": "Lopez" 13 } ]
```

Búsqueda por id

The screenshot shows the Postman application interface. At the top, there are tabs for 'Reports' and 'Explore'. A search bar contains the text 'Search Postman'. On the right side of the header are icons for cloud sync, invite, notifications, settings, and upgrade. Below the header, a navigation bar shows a sequence of requests: GET, POST, POST, GET, POST, POST, followed by a plus sign and three dots, indicating more requests. To the right of this is a dropdown for 'No Environment' and a眼睛 icon.

The main workspace displays a single request: `http://localhost:8087/api/clientes/3`. The method is set to `GET`, which is highlighted with a red box. The URL is also displayed in the input field. To the right of the URL are 'Save' and 'Edit' buttons. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. The 'Headers (6)' tab is currently selected. Under 'Query Params', there is a table with columns: KEY, VALUE, and DESCRIPTION. The table is currently empty. Below the table, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Body' tab is selected. To the right of the body tabs, there is a status indicator showing `200 OK`, `47 ms`, and `281 B`, along with a 'Save Response' button. At the bottom of the body section, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. The 'JSON' tab is selected and has a dropdown arrow. The response body is shown as a JSON object:

```
1 {  
2   "id": 3,  
3   "nombre": "Maria",  
4   "apellido": "Orillana",  
5   "email": "mo@hotmail.com",  
6   "telefono": 65433223,  
7   "createdAt": "2021-02-01"  
8 }
```

At the very bottom of the interface, there are navigation icons for 'Postman', 'Profile', 'Logout', and a help icon.

Probamos el método POST

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Reports' and 'Explore' tabs, a search bar, and various icons for authentication, invites, and settings. Below the header, a toolbar shows recent requests: GET h., DEL h., GET U, POST h. (highlighted in orange), and DEL h. The status bar indicates 'No Environment'. The main workspace shows a POST request to 'http://localhost:8087/api/clientes/'. The 'Body' tab is selected, showing the raw JSON payload:

```
1 {  
2   "nombre": "Joaquin",  
3   "apellido": "Rosales",  
4   "email": "jrosales@hotmail.com",  
5   "telefono": 9132211  
6 }
```

The 'Body' tab also includes tabs for 'Pretty', 'Raw', 'Preview', and 'JSON' (which is currently selected). Below the body, the response is displayed in a table:

Body	Cookies	Headers (5)	Test Results	Save Response
1 { 2 "id": 11, 3 "nombre": "Joaquin", 4 "apellido": "Rosales", 5 "email": "jrosales@hotmail.com", 6 "telefono": 9132211, 7 "createdAt": "2021-11-21T15:55:52.129+00:00" 8 }				

At the bottom, there are buttons for 'Bootcamp', 'Runner', 'Trash', and a help icon.

Método PUT

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Reports', 'Explore', a search bar ('Search Postman'), and various icons for cloud sync, invite, settings, and notifications. Below the navigation is a toolbar with icons for back, forward, and search, followed by a dropdown menu for environments ('No Environment') and a save button.

The main workspace displays a request card for a PUT operation:

- Method:** PUT (highlighted with a red box)
- URL:** http://localhost:8087/api/clientes/11
- Body:** (highlighted with a red box) - Contains the following JSON payload:

```
1 {  
2     "nombre": "Rolando",  
3     "apellido": "Lopez",  
4     "email": "rlopez@hotmail.com",  
5     "telefono": 6432332  
6 }
```
- Content Type:** raw (highlighted with a red box)
- Format:** JSON (highlighted with a red box)

Below the request card, the response section shows the results of the PUT request:

- Status:** 201 Created
- Time:** 41 ms
- Size:** 289 B
- Save Response:** (button)
- Body:** (highlighted with a red box) - Shows the updated client data:

```
1 {  
2     "id": 11,  
3     "nombre": "Rolando",  
4     "apellido": "Lopez",  
5     "email": "rlopez@hotmail.com",  
6     "telefono": 9132211,  
7     "createdAt": "2021-11-21"
```
- Format:** Pretty (highlighted with a red box)

Método DELETE

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Reports' and 'Explore' tabs, a search bar 'Search Postman', and various icons for cloud, invite, settings, and notifications. Below the header, a toolbar shows recent requests: GET, DEL, GET, PUT, and another DEL. The main area displays a request configuration for a DELETE method:

- Method:** DELETE
- URL:** http://localhost:8087/api/clientes/1
- Params:** Authorization, Headers (6), Body, Pre-request Script, Tests, Settings, Cookies
- Query Params:** A table with columns KEY, VALUE, DESCRIPTION, and Bulk Edit. It contains one row with Key, Value, and Description.
- Body:** Options: Pretty, Raw, Preview, Visualize, Text (selected), and a copy icon.
- Headers:** (4) - This tab is selected.
- Test Results:** This tab is visible but not selected.
- Status:** 200 OK, 56 ms, 123 B
- Save Response:** A dropdown menu.