CURSO DE SPRING

**Introducción a SpringBoot**

**Qué es Spring y que usaremos en el curso**

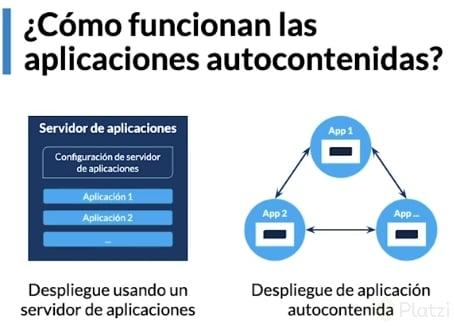
Spring es el framework más usado de Java. Nos ofrece herramientas que nos permiten crear proyectos más avanzados, con mejores prácticas y en menor tiempo. También posee una gran comunidad, lo que nos brinda muchísima documentación y ayuda.

Spring tiene una estructura modular y flexible, lo que nos hace usar solo lo que necesitemos.

Vamos a usar 4 subproyectos de Spring:

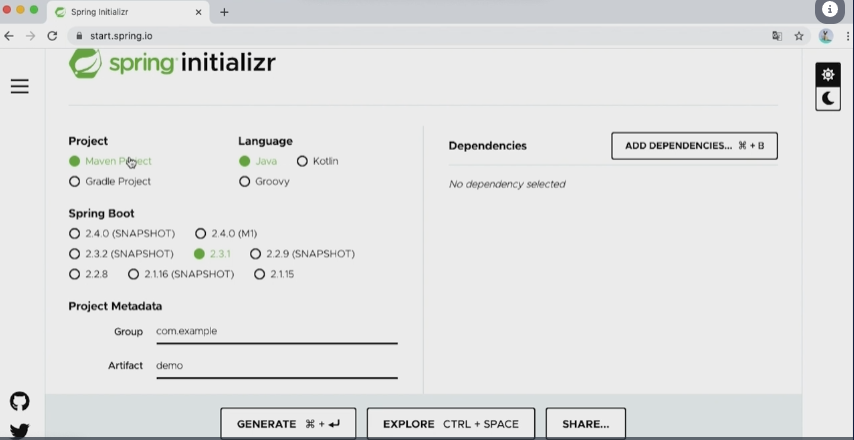
* Spring Framework: Permite crear aplicaciones empresariales. Es transversal, ya que todos lo usan.
* Spring Boot: Con el que podemos crear aplicaciones auto contenidas y auto configurables.
* Spring Data: Gestionar e integrar bases de datos.
* Spring Security: Gestionar la seguridad de la aplicación

**Que es una aplicación contenida**



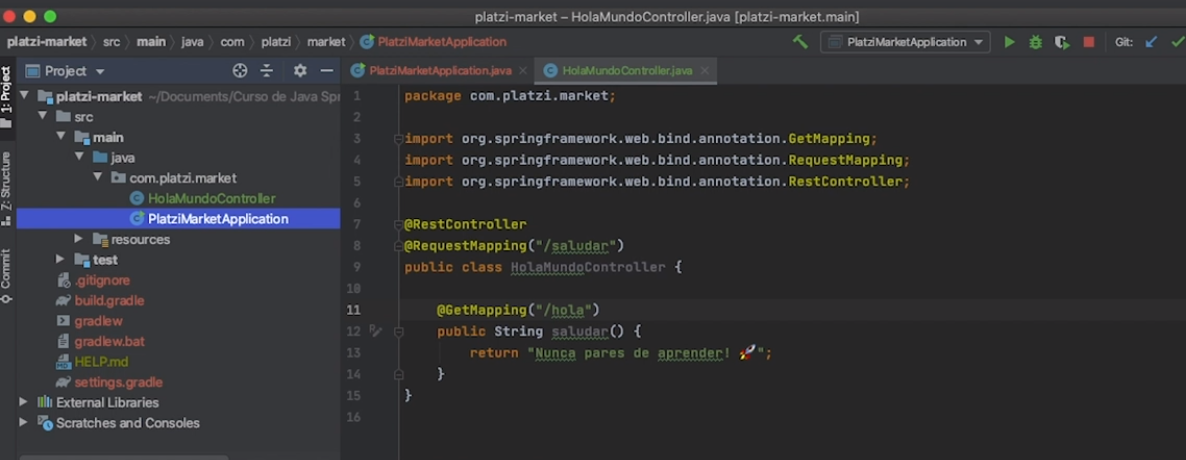
Hasta hace algún tiempo las aplicaciones web empresariales lucias como el grafico de la izquierda, teníamos un servidor de aplicaciones que contenían su configuración donde van configuración de bases de datos, servicios externos, variables, etc. allí también ibas desplegadas todas las aplicaciones que quisiéramos que interactuaran o colaborativamente entre sí, ahora las tendencias han cambiado las arquitecturas modernas nos sugieren tener algo como el diagrama de la derecha donde tengamos pequeñas aplicaciones o pequeños servicios que interactúen entre sí, en vez de una gran aplicación. esto nos da mucha facilidad al momento de desarrollo y al momento de mantener nuestra aplicación. Cada aplicación internamente contiene su propio servidor de aplicaciones, con una configuración totalmente independiente una de la otra. como vimos en la clase anterior sprint Boot es el proyecto de spring para crear aplicaciones auto contenidas esto nos permite olvidar completamente de la arquitectura y enfocarnos únicamente en desarrollo, delegadle a sprint boot labores como configuración de dependencias o desplegar nuestros servicios o aplicación a un servidores de aplicaciones y enfocarnos únicamente en crear el mejor código posible. Para eso sprint boot utiliza internamente un servidor de aplicaciones embebido o un contenedor de aplicaciones embebido, por defecto sprint boot utiliza TOMCAT pero podemos hacerlo también con JETTY o con UNDERTOW.

**Creando aplicación con inizializer**

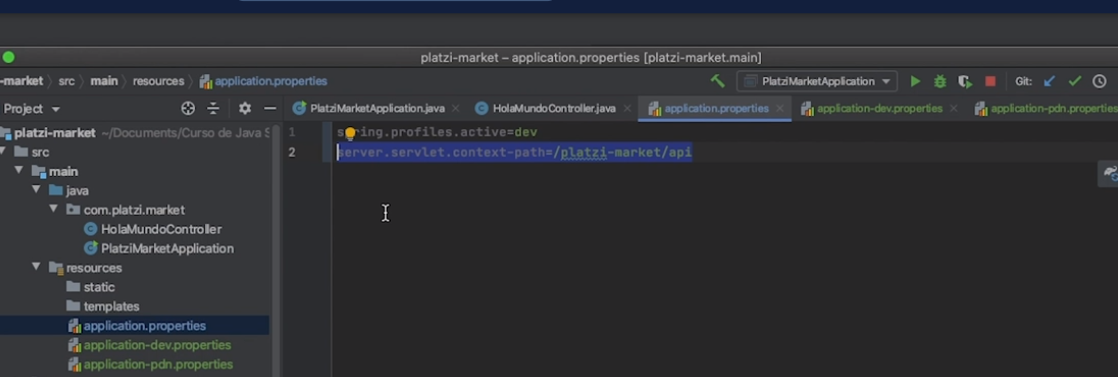


**Desde INTELIJ IDEA.**[**https://www.adictosaltrabajo.com/2016/07/29/arranca-con-spring-initializr/**](https://www.adictosaltrabajo.com/2016/07/29/arranca-con-spring-initializr/)

**Hola mundo con spring**



Para poder poner a ejecutar nuestra plataforma en Spring podemos realizar la actividad teniendo en cuenta la creación del controlador el cual se encargará de tener las salidas de nuestro proyecto, para esto creamos una anotación en la línea 8 llamada request mapping donde damos la salida del endpoint saludar y hola.

**Configurar SpringBoot**

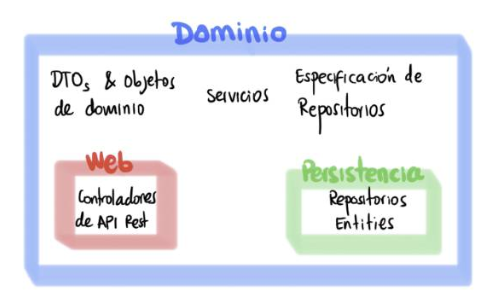
Para este caso se puede estructurar dentro de nuestro proyecto la creación de perfiles , dando como preferencia según en resouces properties la configuración establecida para ejecutar nuestro proyecto.

Podemos encontrar más información en la documentación oficial de todo lo que podemos trabajar en properties.

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

**Crear la estructura del proyecto**

La estructura del proyecto está enfocada en el marco de dominio, es decir la arquitectura por capas orientada al dominio.



**Objetos de Dominio (DTO):** Son objetos que hacen parte del contexto de nuestra aplicación. En este caso hacen parte de un contexto de un supermercado.

**Servicios:** Van a ser los encargados de servir como puente entre los controladores de la API y la capa de persistencia o el repositorio, que es quien interviene con la base de datos.

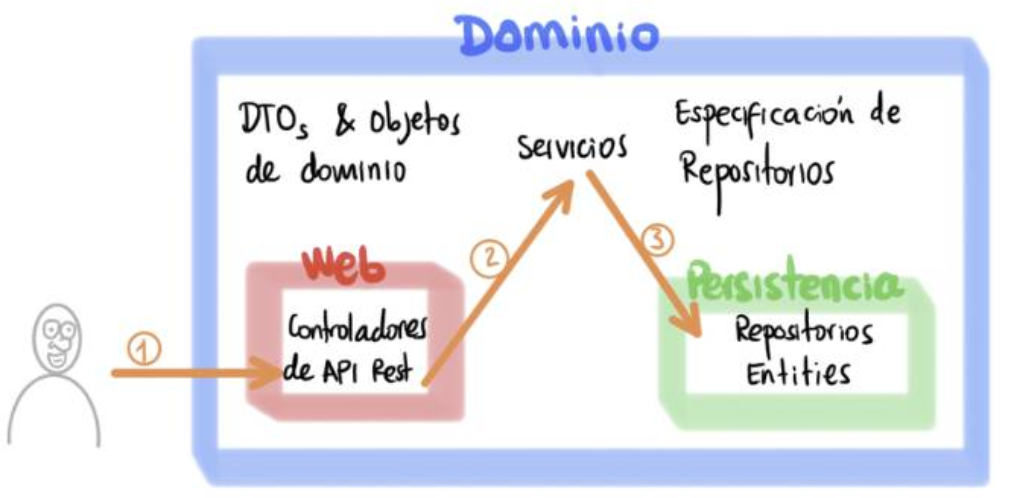
**Especificación de los repositorios:** Interfaces que determinan las reglas que debe cumplir la persistencia para actuar entre los objetos de dominio y la DB.

**Capa de persistencia:** Es la capa que tiene la obligación de interactuar con la base de datos.

**Repositorios:** Son los que van a implementar las especificaciones que tenemos definidas en la capa de dominio.

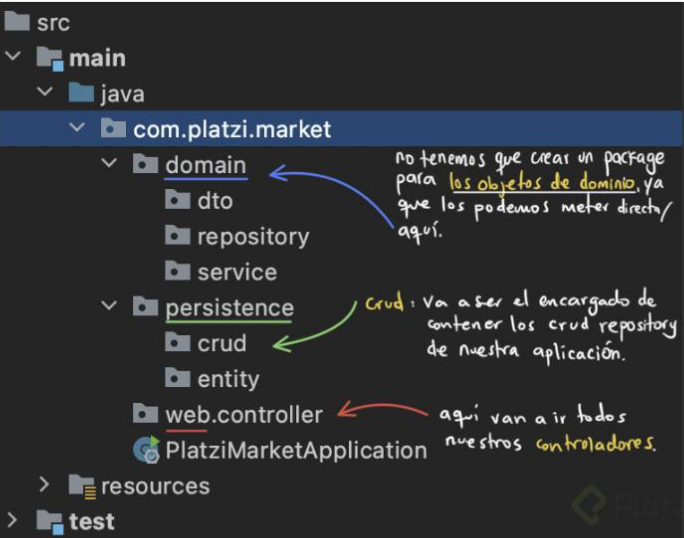
**Entities:** son las clases que mapean y hacen las veces de las tablas que están en nuestra base de datos.

**EL FLUJO NORMAL DE NUESTRA APP SERIA ASI:**



**PASOS DE NUESTRA LOGICA DEL NEGOCIO**

1. **Un cliente hace un llamado a un controlador de nuestra API**
2. **El controlador de nuestra API va a un servicio que tiene todo lo necesario para intervenir en esta operación.**
3. **El servicio va a ir al repositorio especifico que necesite y es allí donde si tenemos que hacer algún movimiento o gestión en la base de datos, es donde ocurre.**



**SPRING DATA**

**Que es JPA**

**JPA : Es una especificacion de java ( un estandar ) para un framework ORM.**

**Interactuar con las tablas de la base de datos en forma de objetos.**

**Algunas de sus implementaciones son: Hibernate , EclipseLink , topLink , ObjectDB**

**Anotaciones de JPA**

**JPA utiliza anotaciones para conectar clases a tablas de la BD y así evitar hacerlo de manera nativa en SQL.**

**@Entity. indica a una clase de java que está representando una tabla de nuestra base de datos.**

**@Table. Recibe el nombre de la tabla a la cual esta mapeando la clase.**

**@Column. se le pone a los atributos de la clase , no es obligatoria , se indica solo cuando el nombre de la columna es diferente al nombre del atributo de la tabla.**

**@id and @EmbededID es el atributo como la clave primaria de la tabla dentro de la clase. @id se utiliza cuando es clave primaria sencilla y @EmbededID cuando es una clave primaria compuesta**

**@GeneratedValue permite generar automáticamente valores para las clases primarias de nuestras clases**

**@OneToMany.**

**@ManyToOne representan los tipos de relaciones uno a mucho y muchos a uno.**

**Que es Sprint data**

Spring Data NO es una implementación de JPA , sino más bien es un proyecto que usa JPA para ofrecer funcionalidades extra en la gestión de tareas desde JAVA a las bases de datos .

Spring Data internamente tiene varios subproyectos , entre ellos: Spring data JPA y Spring Data jdbc para conectarnos a las bases de datos relacionales (SQL) .

SpringData MongoDB y Spring Data Cassandra son proyectos para conectarnos a BD no relacionales.

La tarea principal de Spring Data es optimizar tareas repetitivas .

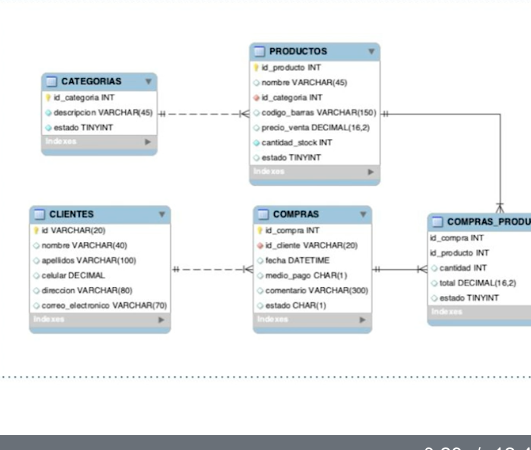
Spring Data nos provee de repositorios sin código, nos permite hace todo tipo de operaciones en BD (CRUD) sin utilizar una línea de código , JPARepository , CrudRepository , PagingAndSortingRepository

Vamos a ingresar nuestro spring data a nuestro proyecto.

También nos provee de auditorías transparentes por ello posee un motor de auditorías que nos permiten saber cuando se inserto un registro , cuando se borró , cuando se actualizó en la BD etc.

**creando la base de datos de la aplicación**

Primero debemos verificar la estructura de la base de datos para la construcción de la misma



luego vamos a realizar la creación de las tablas y registros en nuestra base de datos para esto vamos a crear nuestra base de datos en postgres luego alli crearemos la base de datos e importamos los script sql.

vamos a dar clic derecho en el nombre de la base de datos y CREATE SCRIPT luego vamos a realizar la importación de los datos .

Después de importar los datos vamos a crear la dependencia de conexión con postgres.

esto lo hacemos en el archivo pom.xml

<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->

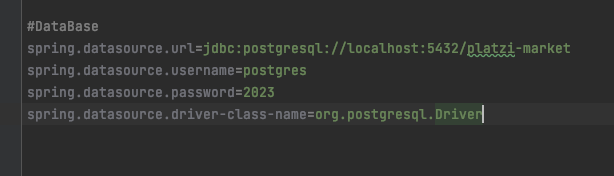
<dependency>

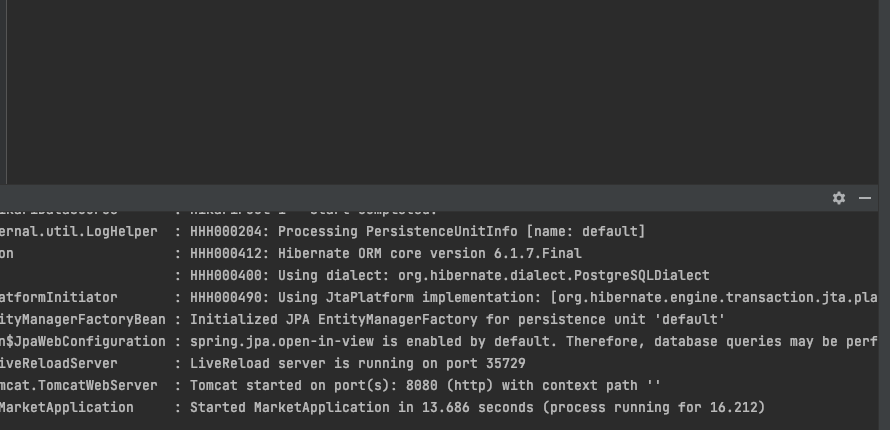
<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.6.0</version>

</dependency>





Si deseamos usar la conexión de mysql usamos las siguientes líneas de código en el pom y en el properties.

implementation 'mysql:mysql-connector-java'

spring.datasource.url=jdbc:mysql://localhost/java?serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

**mapear las tablas con clases**

**Para este paso debemos identificar las clases con sus respectivos atributos estos atributos se denominan con sus clases con las anotaciones correspondientes como veremos a continuación.**

****

**El nombre de la clase es el nombre de la tabla**

**si el nombre de la tabla de la base de datos cambia es necesario usar la anotacion @table que se encargara de conectarse al nombre correcto de la base de datos.**

**Hacemos el procedimiento de colocar las columnas que en este caso serían los atributos de la clase o de la tabla y posteriormente los getter y setter.**

****

**Hacemos lo mismo para las diferentes tablas**

**Crear Entity cuando su clave primaria es compuesta**

estas entidades son establecidas para la unión de las tablas establecidas como claves primaras y claves foreing key

estas tablas como lo son comprasproducto y compras producto pk deben ser creadas como clases y tener los atributos correspondientes para establecer las conexiones .



Esta tiene una anotación embeddedID que permite establecer que será embebida o establecida para un conexión y así pueda ser incorporado de otras tablas.

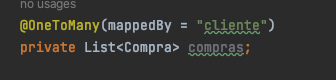
**Mapear relaciones entre clases**

Cuando tenemos más de dos tablas con primary key esta es una tabla compuesta para esto usaremos una anotacion específica para realizar las tablas correspondientes, para este caso realizaremos las anotaciones correspondientes a las relaciones uno a muchos y muchos a uno.

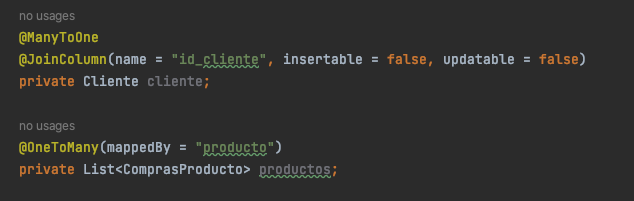
@ManyToOne

@JoinColumn(name = "id\_categoria", insertable = false, updatable = false)

iniciamos con las entidades de categoria donde le decimos cual es la relación en este caso mucho a uno, y luego nos vamos a la tabla de categorias que sera la que se incorpore dentro de las tablas.

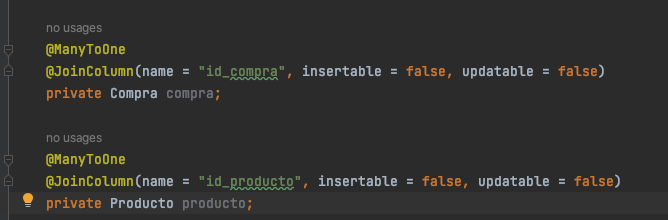


de esta manera creamos las relaciones de categorias producto y productos y clientes



adicional a esto le agregamos una lista de mapeo donde encontrará los productos asociados a esta compra

por último agregamos los valores correspondientes de la tabla compras productos.



de esta forma tenemos nuestra relación de muchos a uno en la tabla compras producto.

**Usar la interface CrudRepository**

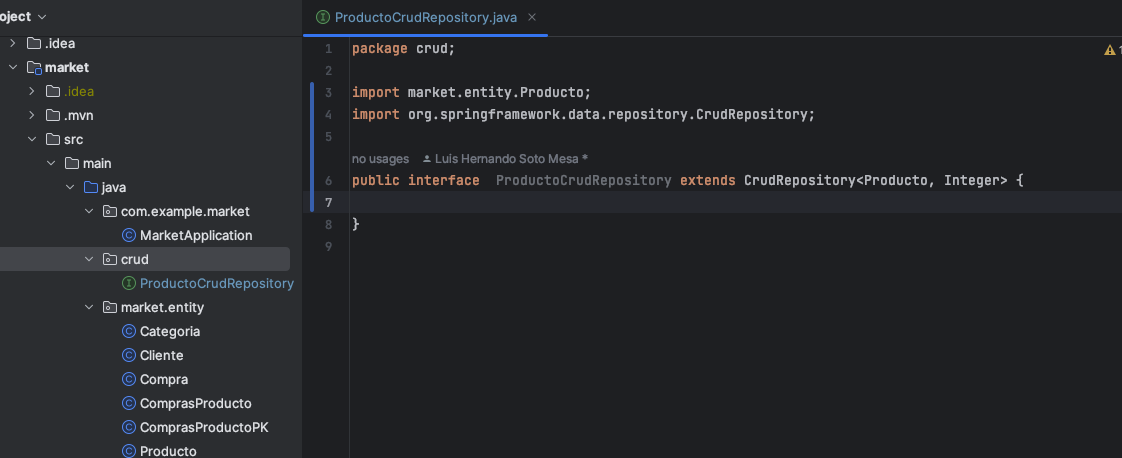
Spring Data repositories

* ahorra un montón de código y tiempo de implementación
* hace operaciones sin código en la base de datos.
* los repositorios de spring data son:
  + CrudReposittory
  + PagingAndSortingRepository
  + JPARepository

**CrudRepository**: Permite hacer las operaciones de CRUD, igual que el patrón de diseño DAO.

**PagingAndSortingRepository**: Nos permite hacer todo lo que hace el CrudRepository pero adicionalmente nos permite hacer tareas de ordenación y paginamiento de nuestro repositorio.

**JPARepository**: Nos permite hacer lo mismo que los dos anteriores, pero además nos permite hacer tareas de JPA específicas como Flush que combina o guarda todo en memoria sin que otras entidades o entornos vean esos cambios en la base de datos.

ahora vamos a crear un paquete nuevo llamado crud y a este le asignemos una interface de nombre ProductoCrudRepository 

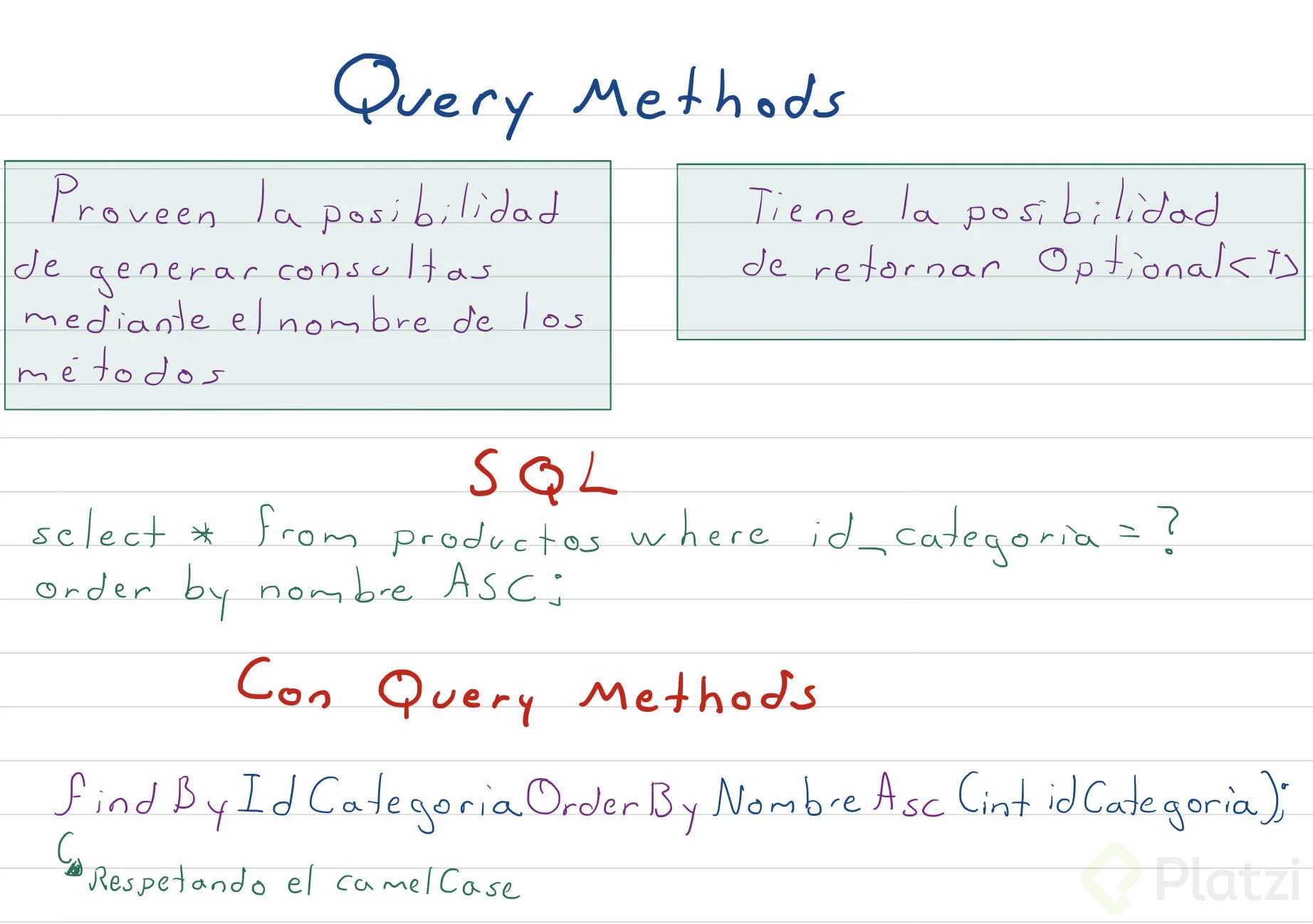
**Query Methods**

**CONSTRUYENDO NUESTRA API**

Después de crear el crudrepository creamos la interface , producto repository.

**Query method**

**Los Query se usan con el fin de realizar la configuración del proyecto con el finde no realizar consultas SQL .**

****

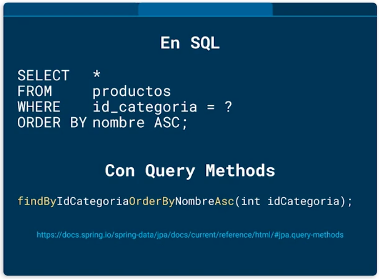
**USOS DE QUERY METHODS**

* En ocasiones necesitamos consultas que el repository de spring data no nos puede ofrecer.
* Los Query Methods proveen la posibilidad de generar consultas mediante el nombre de los métodos.
* Tienen la posibilidad de retornar Opcional <T>

Los query method son muy potentes. Además de los explicado, permiten realizar múltiples operaciones de comparación con:

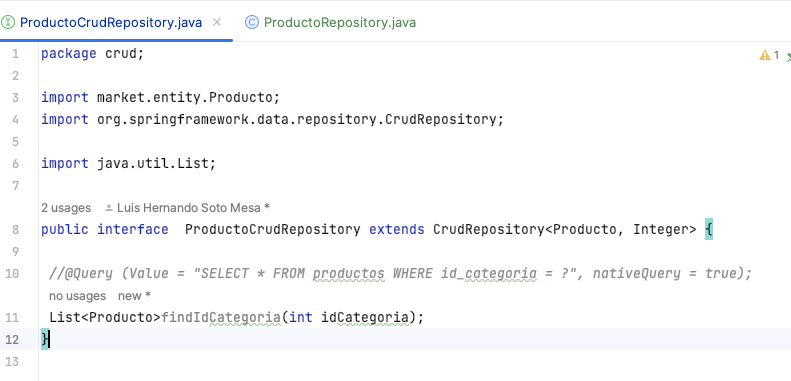
* **Números**: mayores, menores, iguales…
* **Textos**: contiene cierta porción de texto, empieza o termina con una porción de texto, ignora case sensitive…
* **Fechas**: Antes de cierta fecha, después de cierta fecha, entre cierta fecha…
* **Joins entre entidades**: Si tenemos una entidad que se relaciona con otra, es posible realizar “joins” con esa relación para tener queries más específicas según nuestra necesidad. Por ejemplo, si tengo una relación de Producto y Categoría y quiero tener todos los productos de cierta categoría podría hacer: findAllByCategoriasId(Integer categoriaId) y así poder llegar a esta relación. Esto puede mezclarse con múltiples relaciones en simultáneo
* **Comparación entre un conjunto de datos**: Si por ejemplo quiero traerme los productos con varias categorías, podría escribir findAllByCategoriasIdIn(List<Integer> categoriaIds); y así trabajar bajo un conjunto de Id de categorías

Existen más funcionalidades ✌🏼. Pueden ver más detalle acá:<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation>



Para ver el código en nuestro proyecto realizamos lo siguiente.

para realizar las consultas podemos usar los query methods de forma nativa de la siguiente forma .



**ahora nos vamos a la clase productorepository y creamos una nueva lista**

**CONSTRUYENDO NUESTRA API**

**@Repository: le indicamos a la clase que es la encarga de interactuar con la base de datos.  
 @Component: le indicamos que es un componente de spring.**

**El primero es mejor porque indicamos el tipo de componente que eS.**

**Inyectamos ProductCrudRepository en el Repository y lo usamos directamente porque extiende de CrudRepository, que es una funcionalidad que nos brinda Spring Data para interactuar con la base de datos sin necesidad de implementar. ¡Únicamente tenemos que utilizarla! Genial, no?**

****

**Que es el patrón Data Mapper**

**DATA MAPPER**

* **Convertir o traducir dos objetos que pueden hacer una misma labor**
* **No exponer directamente la base datos medianta la API**
* **Esto garantiza que ningun agente externo, vizualice la forma del diseño de la base de datos**
* **Desacoplar la API de una base de datos puntual**
* **En el caso que se desee integrar una nueva base de datos con otros campos, pero que sea para el mismo proyecto, no es necesario cambiar todo el código, simplemente se crea otro traductor que sirva para traducir la nueva tabla al dominio**
* **Evita tener campos innecesarios en la API**
* **Evitar mezclar idiomas en el dominio**

## DataMappers - Mapeando los datos

**Nos permite desacoplar la persistencia de la aplicación. Consiste en convertir o traducir varios objetos que pueden cumplir la misma labor. De esta forma podemos:**

* **Independizar la base de datos de la API, desacoplanto la capa de persistencia o de negocio.**
* **Desacoplarnos de una base de datos puntua, así no tendríamos que refactorizar todo el código si la capa de persistencia cambia.**
* **Evitar campos innecesarios en la API.**
* **Evitar mezclar idiomas en la aplicación.**