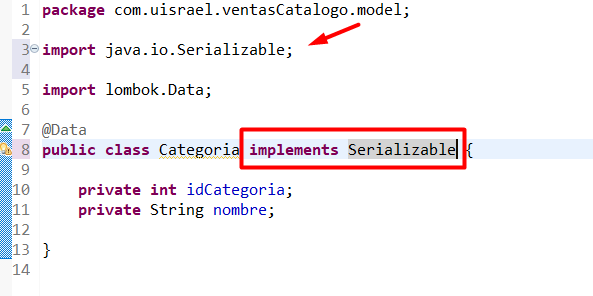
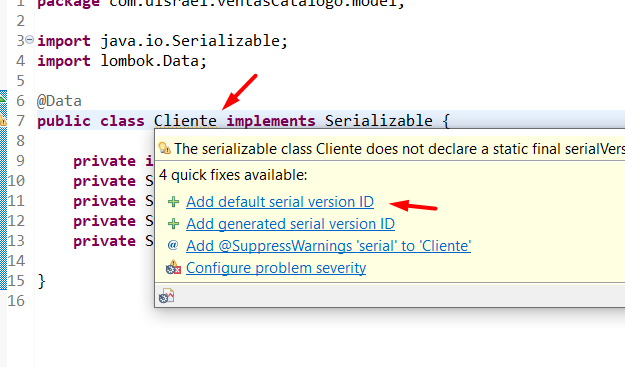
Requisitos:

Haber creado la arquitectura del proyecto  
Haber creado la arquitectura MVC

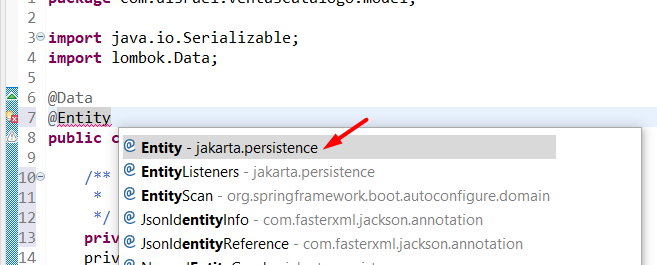
**Serialización:** Sirve para el paso de información para que pase desde el cliente al servidor. Abrimos el modelo y hacemos:



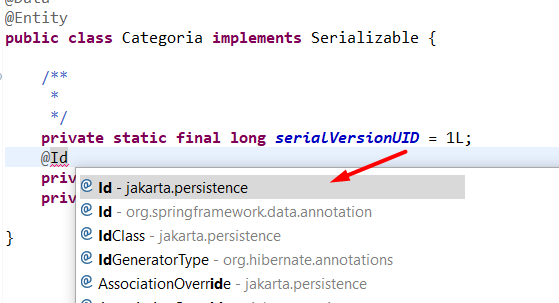
Luego escogemos el generar el ID. Esto genera un id único



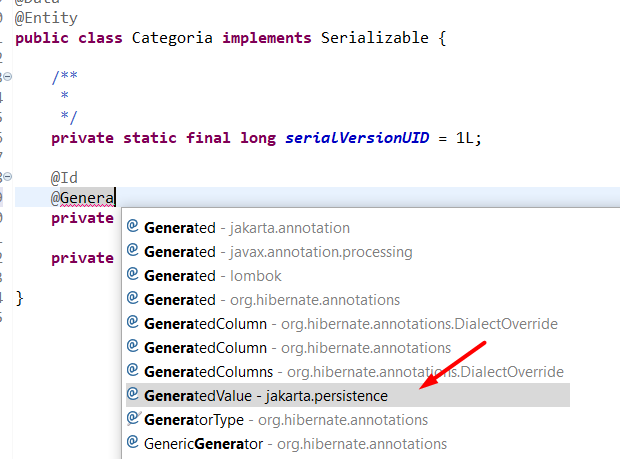
**Entity:** Ahora la hacemos transformar en entidad, hacemos:



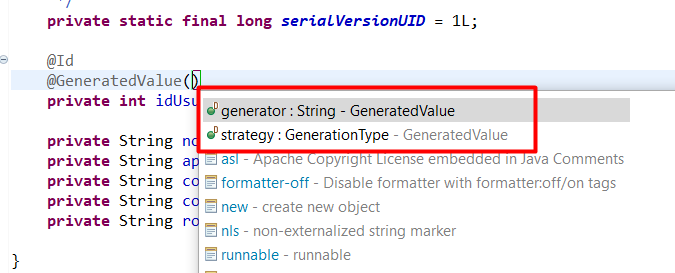
**Único:** Ahora hacemos la PK para que sea único



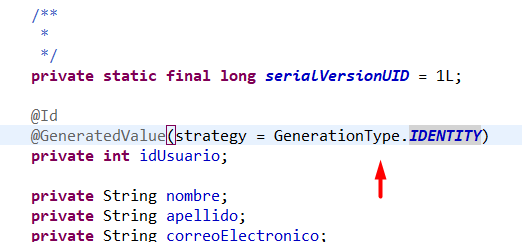
Hacer la **PK automática**



Ahora generamos   
Si la lógica del sistema obliga a usar relaciones dependientes, utilizamos ***generated***, sino, usamos ***strategic***

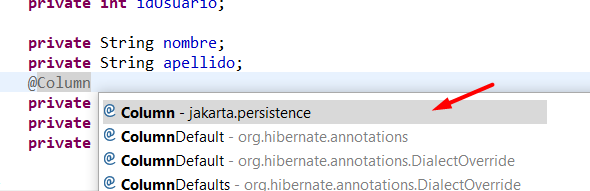


En nuestro caso, usamos **strategic e identity:** Para generar registros de 1 en 1 autoincrementable

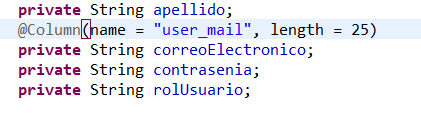


**OPCIONALES:**

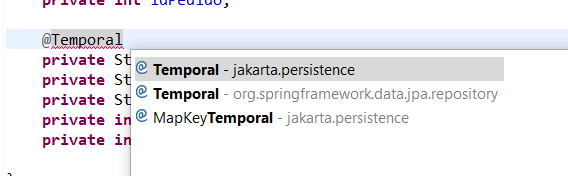
Hay ocasiones en las que necesito mapear mis objetos, mis clases, mis columnas a las columnas de la tabla que tienen nombres diferentes, podemos hacer lo siguiente:

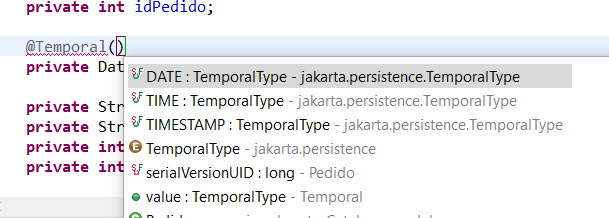


Se hace una personalización del atributo correoElectrónico, de tal forma que a lo que se mapee con el ORM la tabla tendrá en el atributo el nombre de user\_mail, incluso podemos darle una longitud:

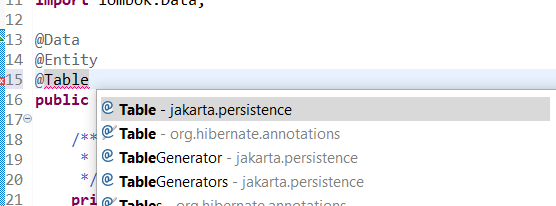


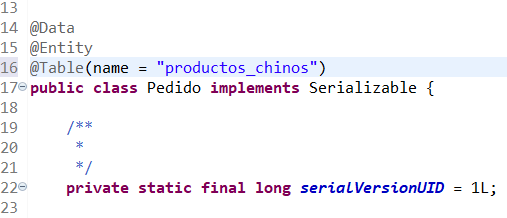
Hay veces en las que trabajar con fechas es un poco confuso porque no sabemos si la fecha tiene formato 12-11-2023 otras que es 12/11/2024, entonces lo mejor es delegar al framework que busque el collection que está configurado en la base de datos, y que haga ese trabajo por nosotros:





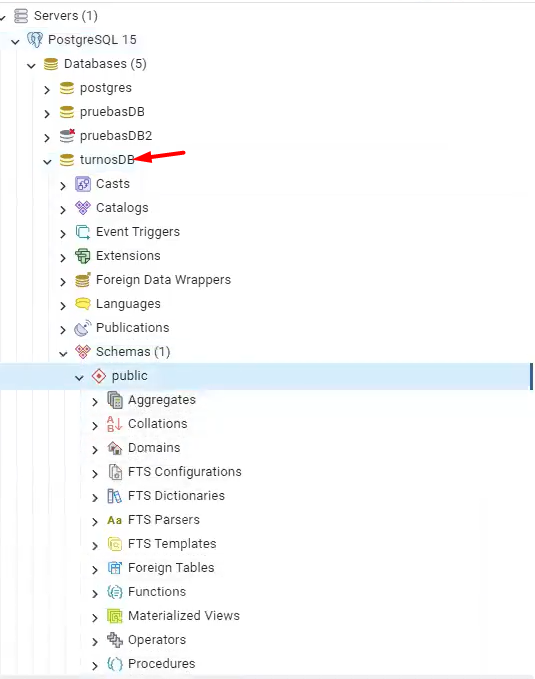
Del mismo modo podemos acoplar nuestro objeto a una tabla que ya exista, podemos por ejemplo, DIGAMOS que en la base de datos existe una tabla productos\_chinos



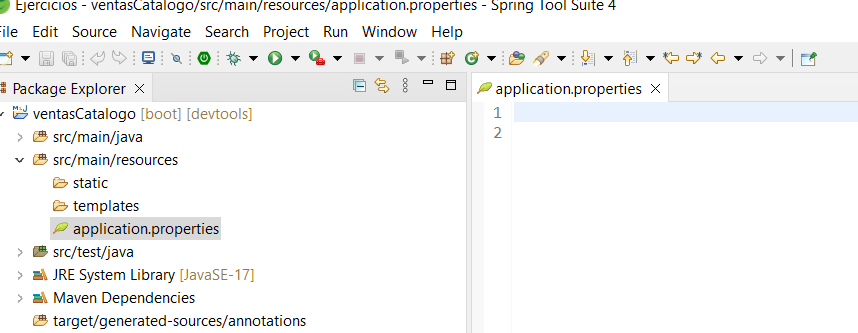


**CONEXIÓN A LA BDD**

Cremos una base de datos en postgres



Vamos a ***resources/application/properties:***



Y colocamos el siguiente código:

# PostgreSQL properties

spring.datasource.url=jdbc:postgresql://localhost:5432/test3

spring.datasource.username=postgres

spring.datasource.password=123123

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

logging.level.org.hibernatee.sql=debug

spring.jpa.show-sql=true

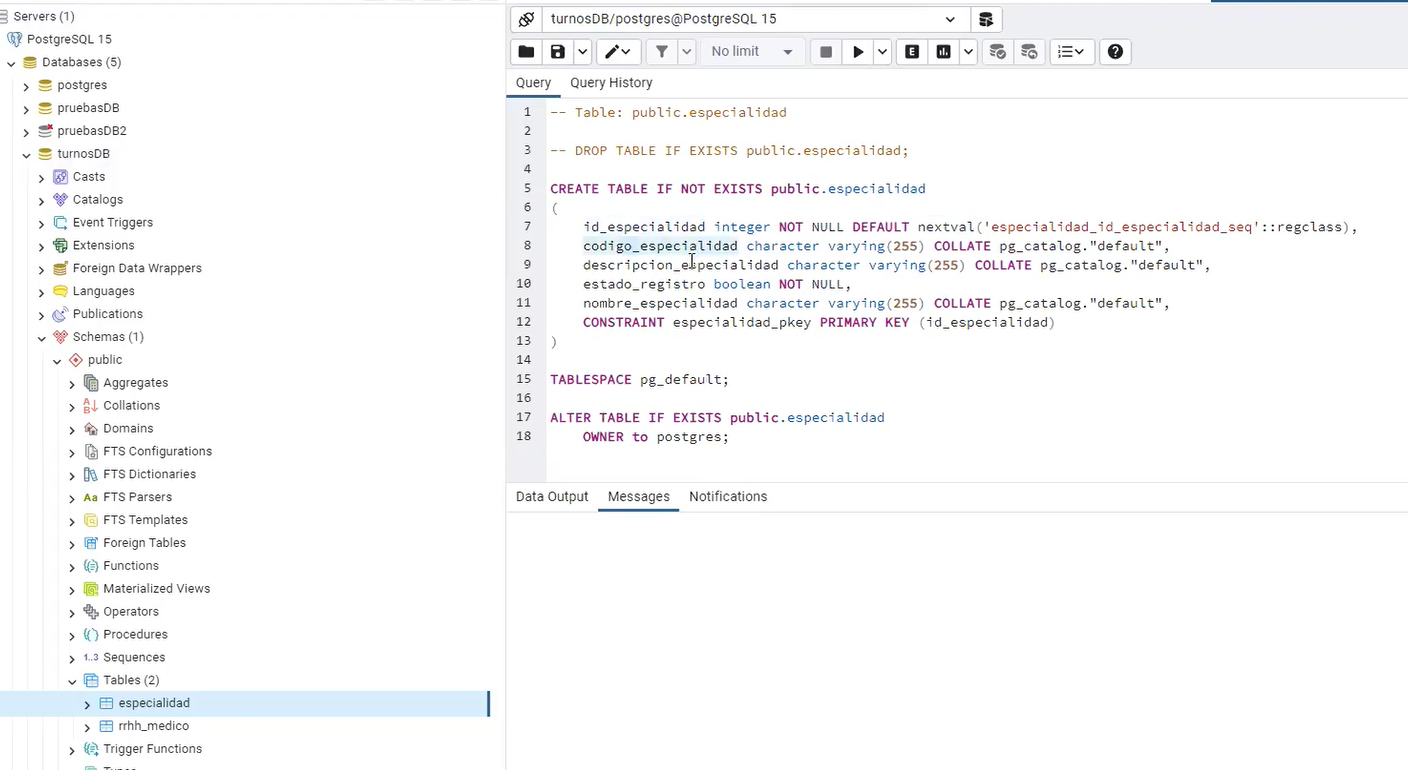
spring.jpa.hibernate.ddl-auto=update#use update si está en entrono no producción, es decir desarrollo

Por último hacemos la prueba de conexión, vamos a *src/test/java*

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Verificamos en la base de datos lo creado:



**PREGUNTA: ¿Cúándo se que debo esgcoger generator y cuándo strategy?¿Puede ser el caso de que en 1 proyecto lleguemos a usar ambas?¿Qué quiere decir si nos obligan a ausar relaciones dependientes?**

Cuando usamos el acoplamiento de tabla, en ley que los atributos de mi programación sean los mismos en número, nombre y atributo a los que existen en la tabla? SI correcto, deben ser el mismo número.

Luego de haber hecho el mapeo de entidades hacia la base, digamos que quiero incorporar una nueva tabla. Se me ocurre borrar la tabla y hacer de nuevo todo el Run Test?

**CRUD: Hacemos herencia**

Ya construido la base de datos, vamos a el repository y usamos herencia:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Escogemos el atributo Medico, e integer porque así es el tipo de dato de la PK

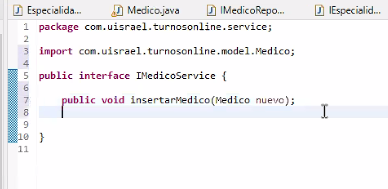
Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Listo sería todo para el crud Repítase para todas las demás entidades

Definimos los métodos que queremos que utilice el usuario

Abrimos los services y escribirmos los métodos

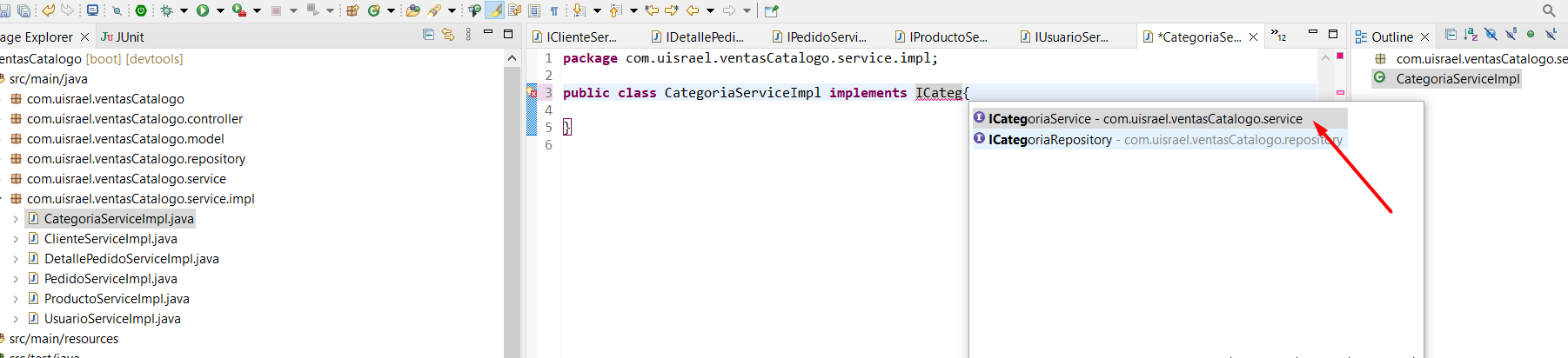


Interfaz de usuario gráfica, Texto, Aplicación

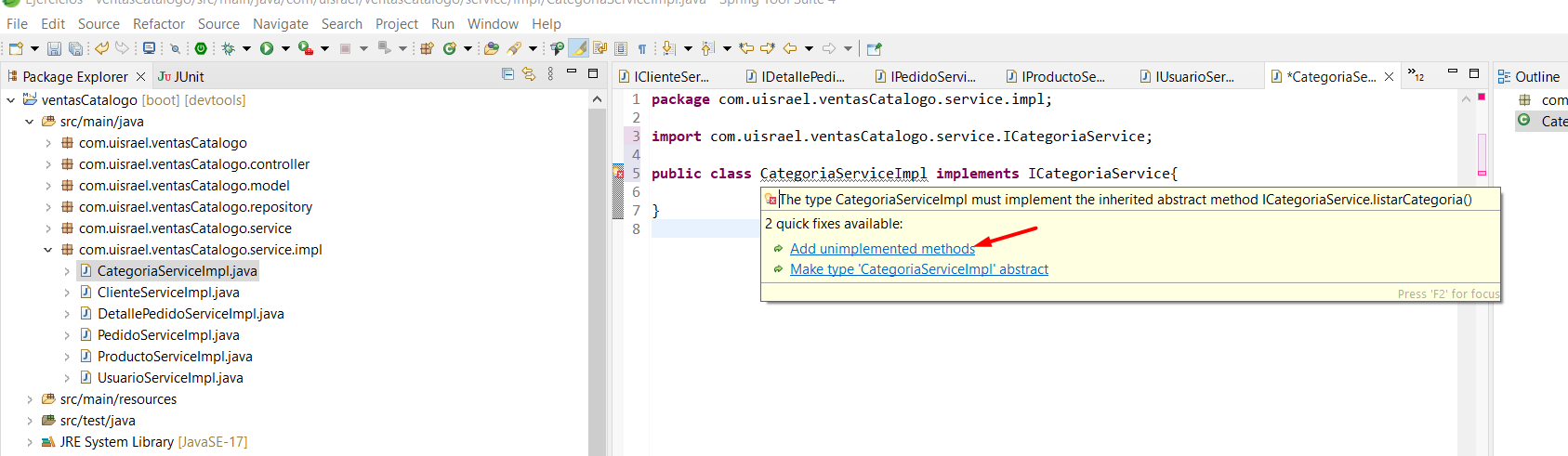
Descripción generada automáticamente

Y adicional los n métodos que falten. En cada una de las entidades.

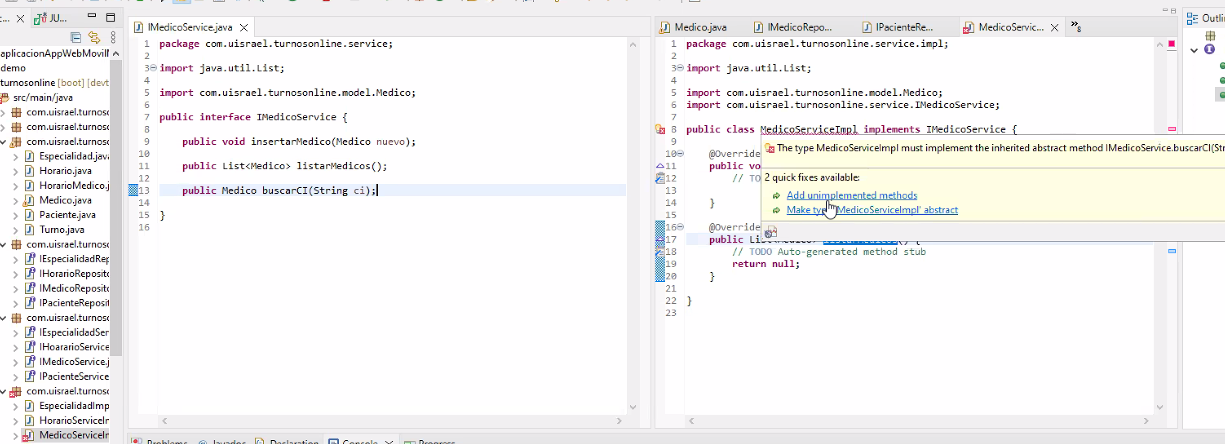
Ahora los implementamos: Vamos al service.impl



Va a pintarse un error, lo que debemos es añadir los métodos

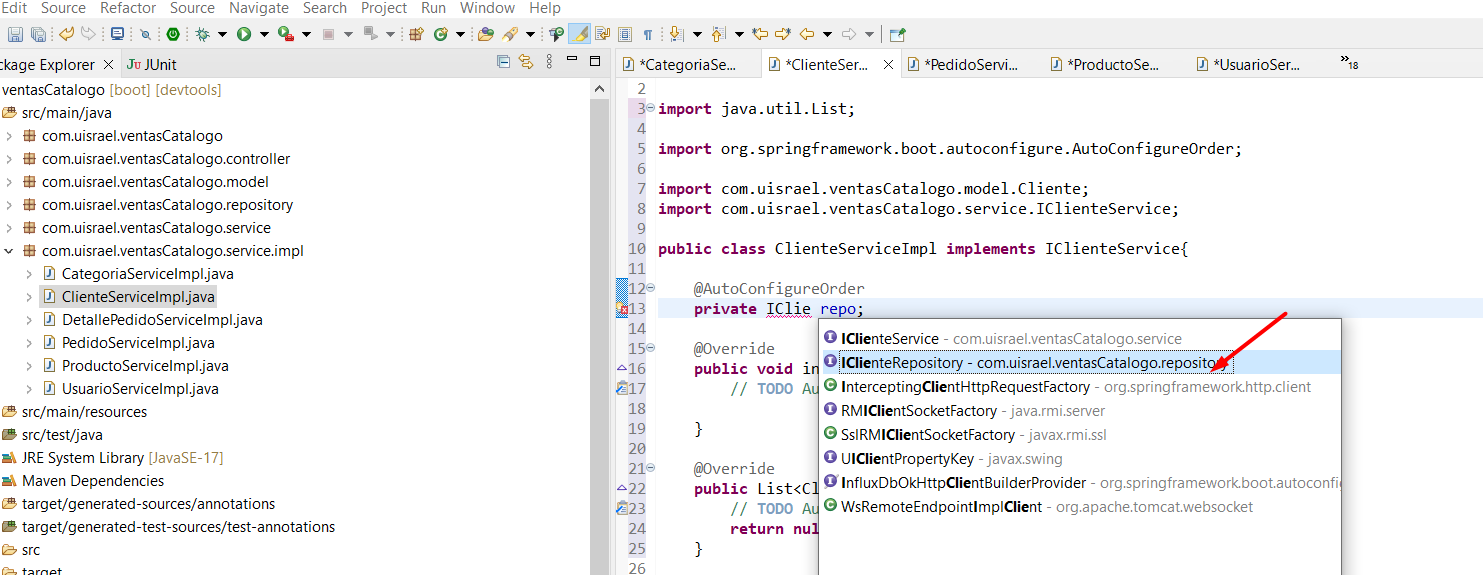


Estos métodos corresponden a los que definidos en el servicio

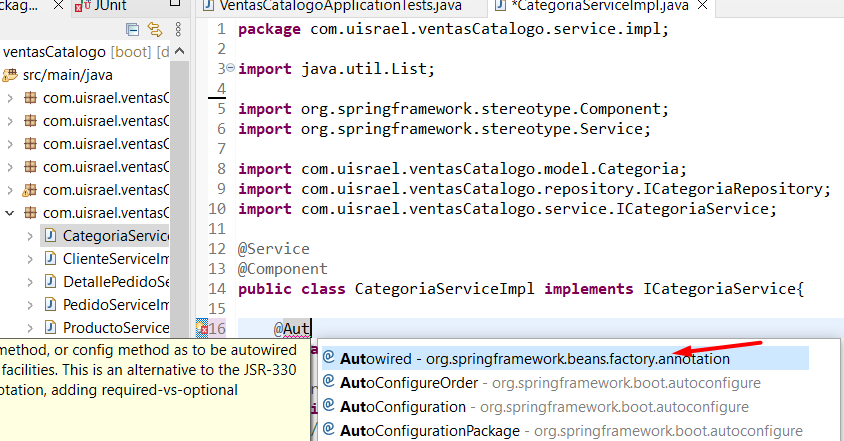


Repítase para cada entidad.

Luego vamos a hacer una instancia.



Hacemos inyección de dependencia con @**Autowired** para delegar la creación de objetos al framework, aquí mismo en el implements



Repítase para cada entidad.

Y Luego

Y para hacer el insert:

Texto

Descripción generada automáticamente

Hacer el Listar todos

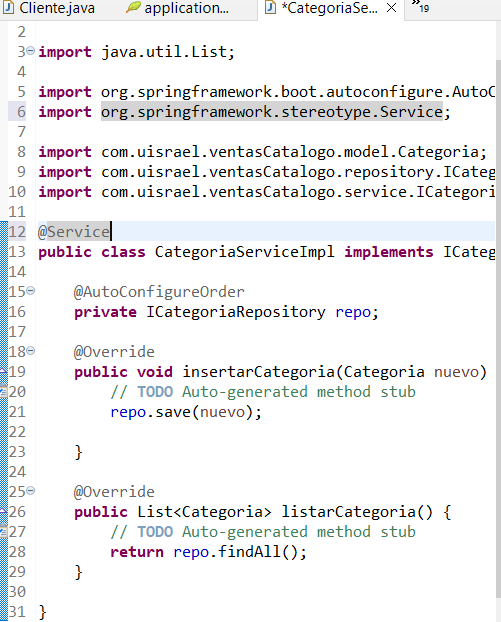
Texto

Descripción generada automáticamente

Ahora usamos estereotipo para decirle al framework que trarte a ala lase comounservicio.Vamos al implements

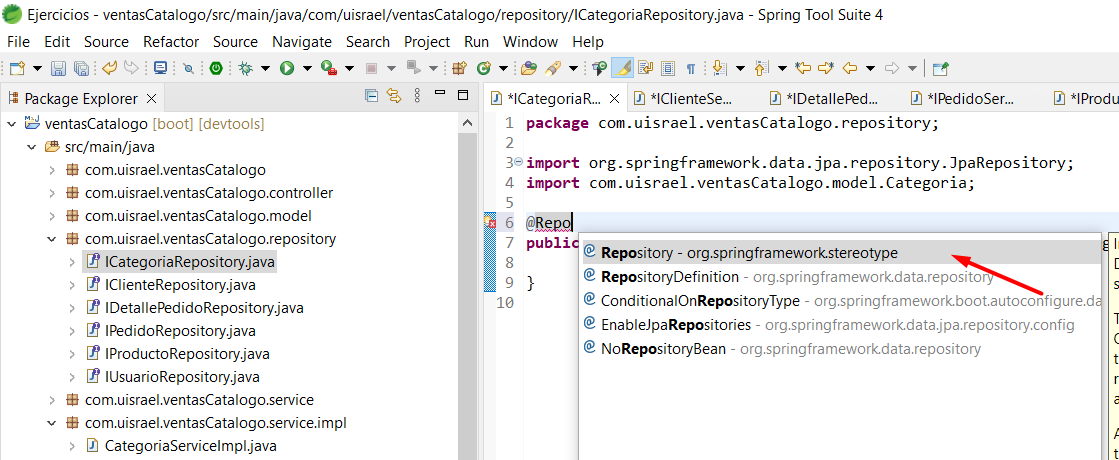
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente



Repítase para todos los demás implements

Del mismo modo que trate al repository como repository



Y si queremos garantizar que se comporte como compientne, también le ponemos component Importante debaje del @Service

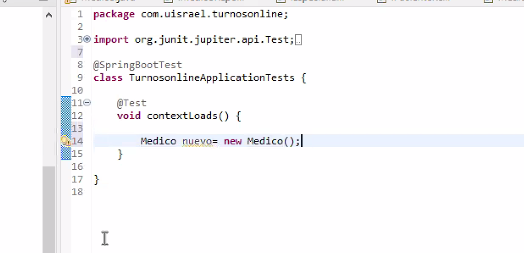
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Vamos a probar todo esto:

Vamos a Junit del test

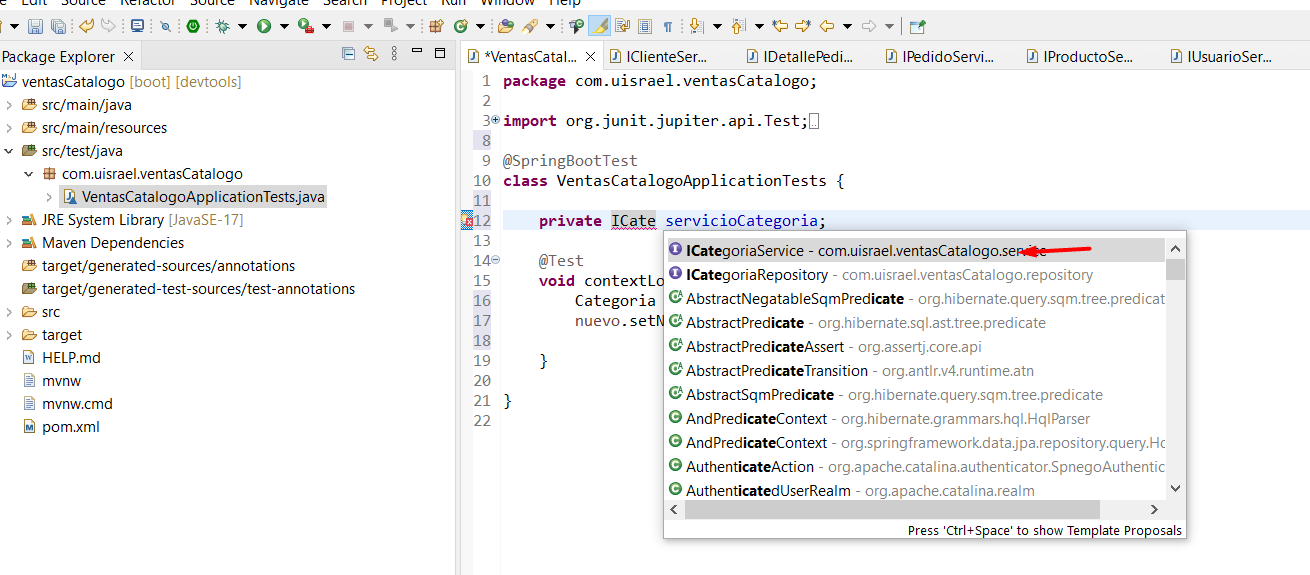
Creamos un objeto nuevo



Vamos a simular que el usuario llenó un formulario



Instanciamos el servicio:



Hacemos inyección de dependencia para que interactue el controlador con el servicio\_

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Por último mandamos el objeto

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ver en consola

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente