La base de datos de nuestro proyecto para este casó será postgres y podremos descargarlo en el siguiente enlace:

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

<https://get.enterprisedb.com/postgresql/postgresql-14.12-2-windows-x64.exe>

**Por motivos de compatibilidad nosotros vamos a instalar una versión antigua**

  
ADVERTENCIA

NO INSTALAR UNA VERSIÓN MUY RECIENTE

NO OLVIDAR LA CONTRASEÑA

NO OLVIDAR EL NÚMERO DE PUERTO

NO INSTALAR LAS HERRAMIENTAS ADICIONALES

por defecto el usuario es **postgres**

**reiniciar el PC**

PASS: kmzwa8awaa

PORT: 5432

LANG: Spanish, Colombia

Permitir todas las conexiones

Ir a la carpeta de instalación: C:\Program Files\PostgreSQL\14\data y vamos a modificar los archivos:

* postgresql.conf  
  
* pg\_hba.conf  
  

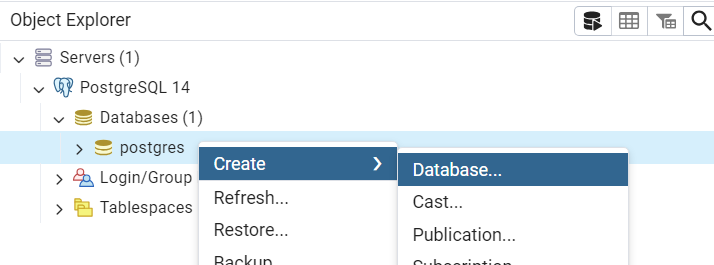
El el archivo postgresql.conf vamos a buscar listen\_addresses y le vamos a poner '\*'

Y para el archivo pg\_hba.conf vamos a agregar al final:

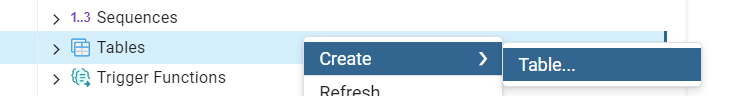
host all all 0.0.0.0/0 trust

**Creación de la base de datos**

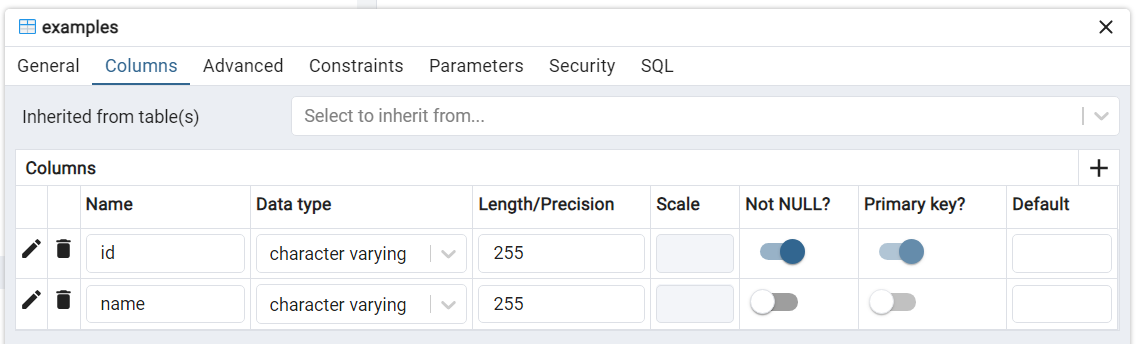
Una vez instalado vamos a dirigiros al inicio y buscar el PGAdmin y lo abrimos y nos autenticamos con la contraseña… y vamos a crear la base de datos de ejemplo y la tabla examples:



La base de nosotros se va a llamar “ejemplo” y luego de crearla vamos a ir a  y allí vamos a crear la tabla “examples” la cual es la representación de nuestra entidad.



Y vamos a crear la siguientes columnas:

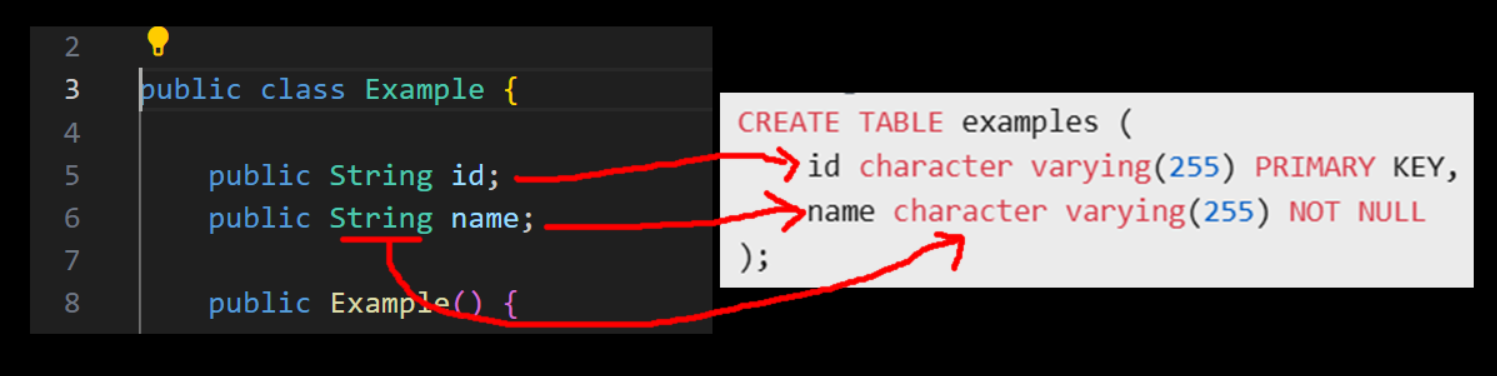


CREATE TABLE examples (

id character varying(255) PRIMARY KEY,

name character varying(255) NOT NULL

);



Vamos a crear 10 datos de ejemplo:

INSERT INTO examples (id, name) VALUES

('1', 'a'),

('2', 'b'),

('3', 'c'),

('4', 'd'),

('5', 'e'),

('6', 'f'),

('7', 'g'),

('8', 'h'),

('9', 'i');

COMMIT;

***Conectar nuestra base de datos con nuestro backend de Spring boot***

**Antes de empezar vamos a contextualizarlos**

Nuestra conexión será establecida mediante librerías y un patrón de diseño llamado DAO



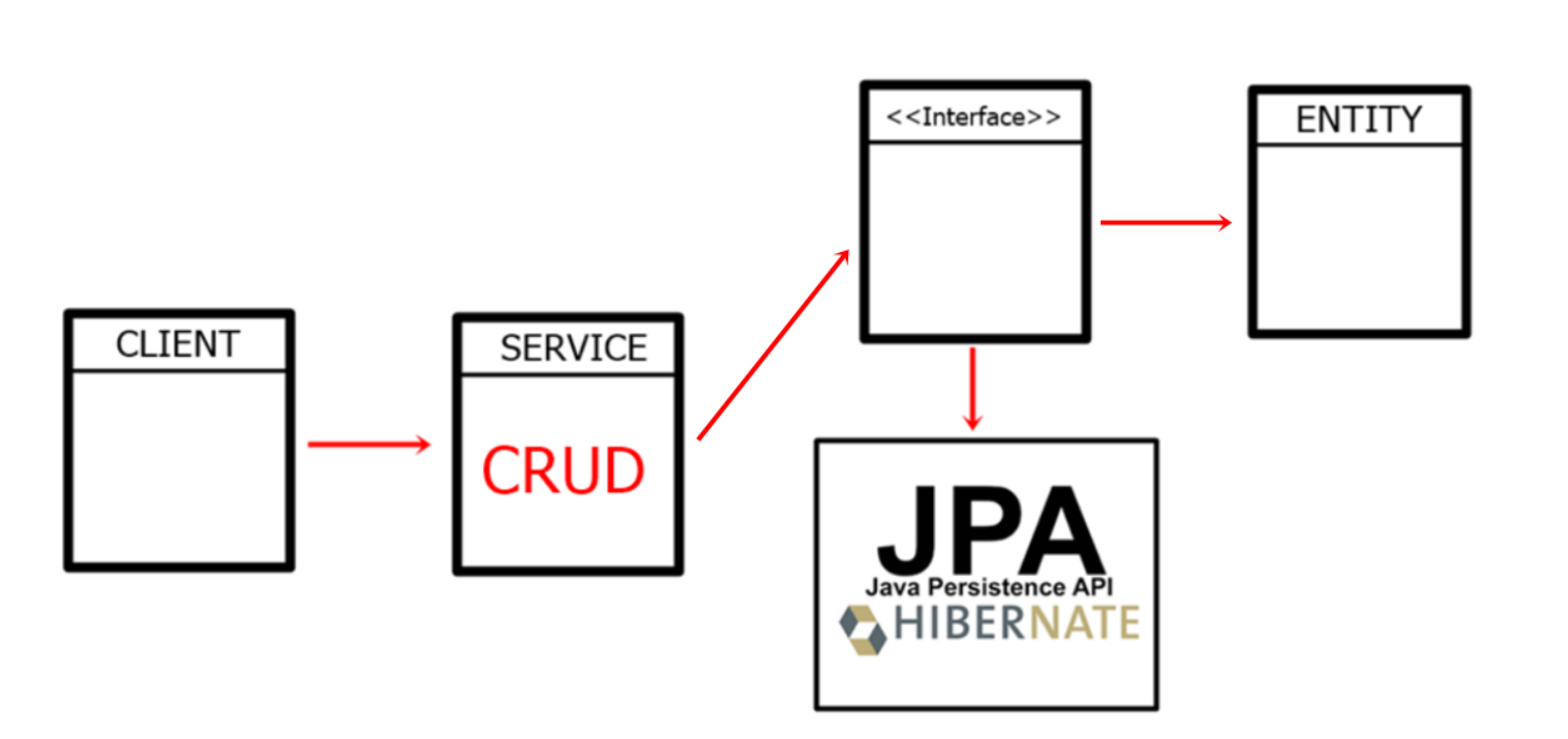
Jakarta es una librería que nos permite mapear una entidad .java de nuestro código hacia la base de datos (Es capaz de convertir un objeto de java a uno de base de datos)



Es un conjunto de reglas y estándares para poder persistir los datos, solo se trata de un facilitador entre java y la base de datos.



Lombok es una librería capaz de escribir código por nosotros usando @ para ahorrar tiempo de desarrollo. por ejemplo si nosotros queremos un constructor con todos los parámetros solo basta con poner @AllArgsConstructor en el principio de nuestra clase y para poner los setters y getters basta con poner @Data



El patrón de diseño DAO (Data Access Object) será lo que nosotros vamos a usar para conectarnos a la base de datos y dicho patrón sirve para separar las consultas de la base de datos de la lógica de negocio. El patrón DAO se encarga de introducir una interfaz que va a contener los métodos de acceso a la base de datos.

**Tutorial para conectar nuestra base de datos POSTGRESQL con Spring BOOT**

**1 - Realizar todas las importaciones necesarias:**

implementation 'jakarta.persistence:jakarta.persistence-api'

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

implementation 'org.springframework.boot:spring-boot-starter-jdbc'

implementation 'org.postgresql:postgresql'

runtimeOnly 'com.h2database:h2'

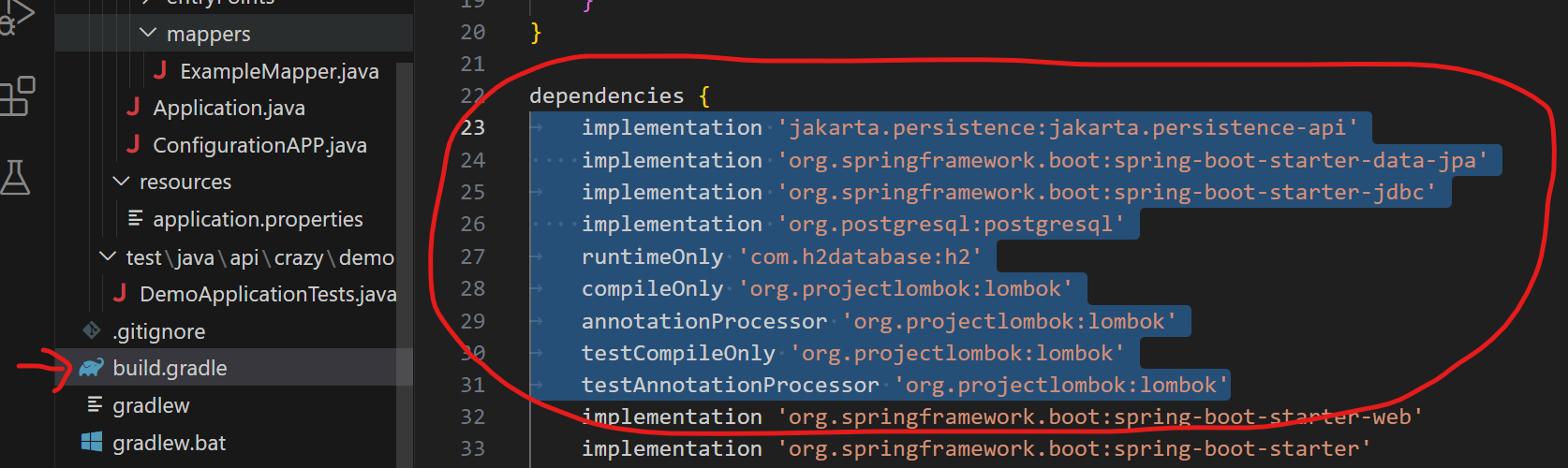
compileOnly 'org.projectlombok:lombok'

annotationProcessor 'org.projectlombok:lombok'

testCompileOnly 'org.projectlombok:lombok'

testAnnotationProcessor 'org.projectlombok:lombok'

Vamos a ir a nuestro archivo gradle (demo\build.gradle) para instalar todas las librerías necesarias:



**2 - Poner las credenciales de la base de datos y la configuración de JPA**

demo\src\main\resources\application.properties

# Configurar POSTGRESQL

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

spring.datasource.username=postgres

spring.datasource.password=TuContraseña

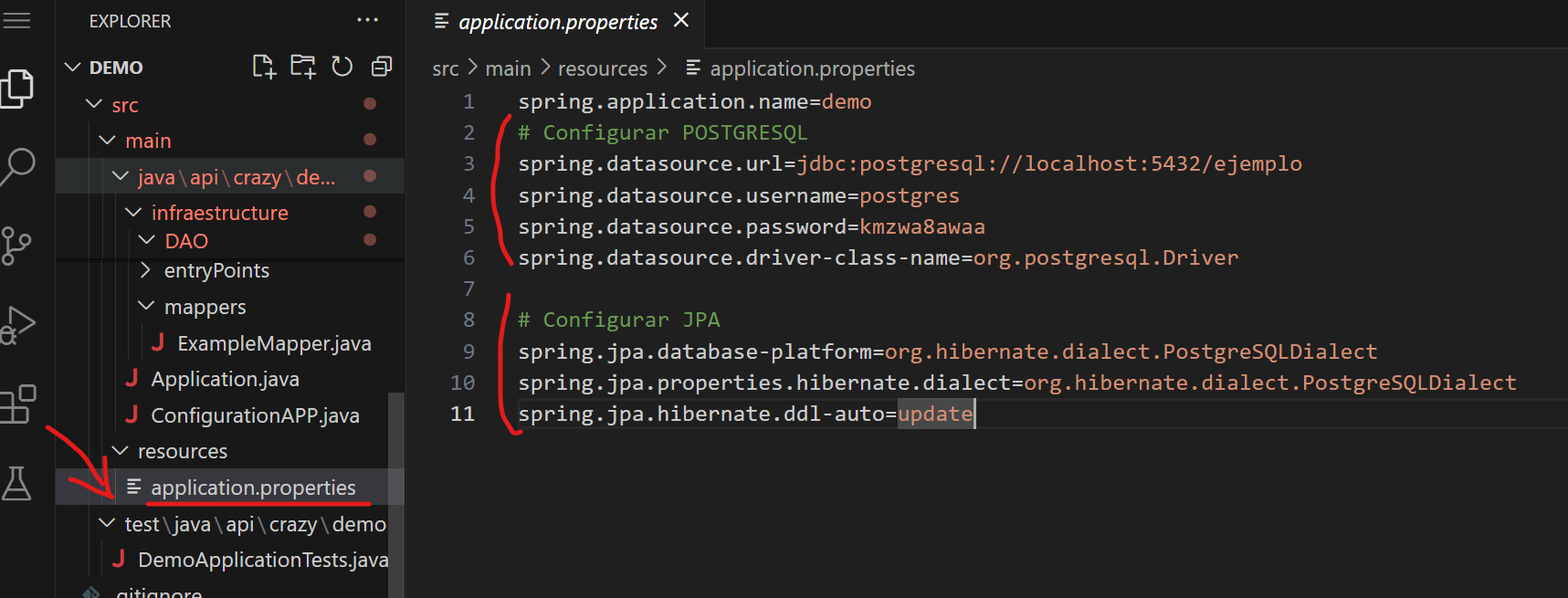
spring.datasource.driver-class-name=org.postgresql.Driver

# Configurar JPA

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

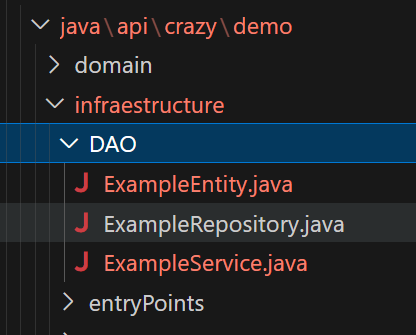
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto=update



**3 - Vamos a crear una nueva capa llamada DAO en infraestructure**

demo\src\main\java\api\crazy\demo\infraestructure\DAO



Advertencia: “Nuestros archivos van a estar en la raíz DAO eso está mal nosotros tenemos que crear una carpeta DAO para cada entidad ExampleDAO”

Allí nosotros vamos a crear 3 archivos:

***Entidad***

package api.crazy.demo.infraestructure.DAO;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

import jakarta.persistence.Column;

import lombok.AllArgsConstructor;

import lombok.NoArgsConstructor;

import lombok.Builder;

import lombok.Data;

@Entity

@Table(name = "examples")

@AllArgsConstructor

@NoArgsConstructor

@Builder

@Data

public class ExampleEntity {

@Id

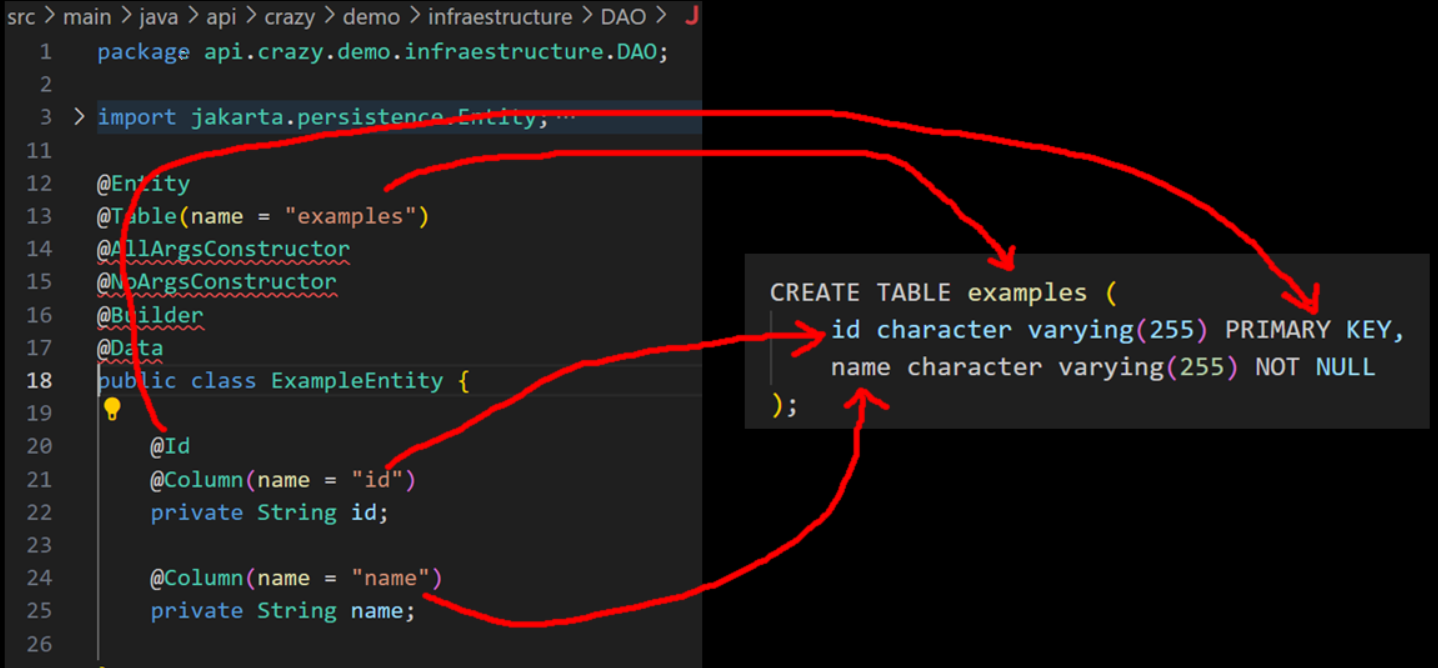
@Column(name = "id")

private String id;

@Column(name = "name")

private String name;

}  
  
Es el encargado de traducir de java a base de datos y todo ello lo tenemos que hacer a través de anotaciones:



| **Anotación Jakarta** | **Descripción** |
| --- | --- |
| @Entity | Le dice a Java que está en base de datos. |
| @Table(name = "examples") | Le dice a JPA como se llama la tabla. |
| @Id | Le dice a JPA cual es el id de la tabla. |

| **Anotación Lombok** | **Descripción** |
| --- | --- |
| @AllArgsConstructor | Crea un constructor con todos los parámetros y su instancia. |
| @NoArgsConstructor | Crea un constructor vacío. |
| @Builder | Crear todos los setters y getters. |
| @Data | Crea instancias del objeto “.build” |

**Contrato de base de datos**

package api.crazy.demo.infraestructure.DAO;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

public interface ExampleRepository extends JpaRepository<ExampleEntity, String> {

@Query(

value = "select \* from examples",

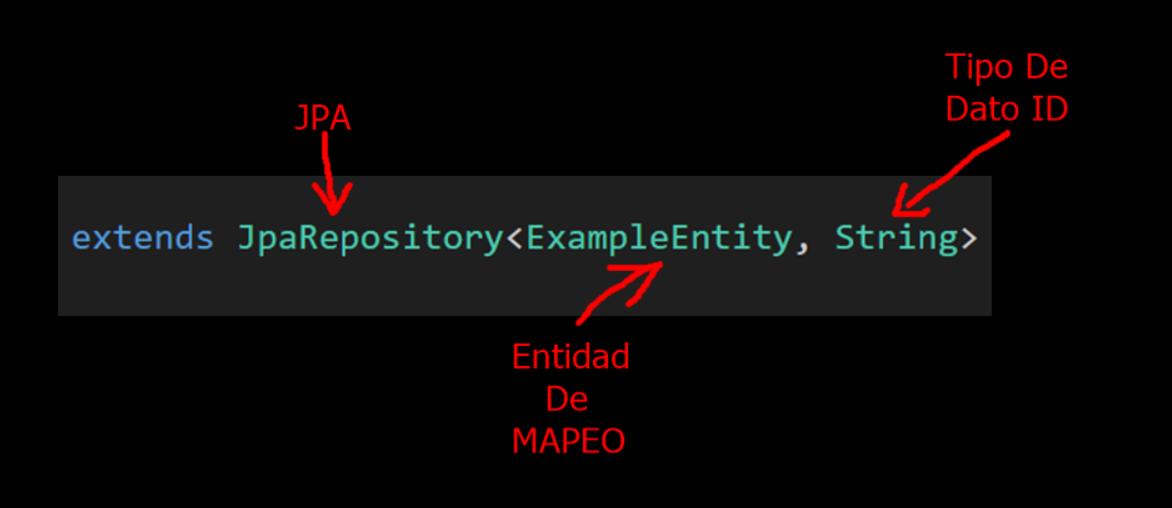
nativeQuery = true

)

List<ExampleEntity> getAll();

}

Es el encargado de conectarse hacia la base de datos y todo ello porque es una extensión del JPA allí vamos a poner todas las consultas personalizadas. y todo ello se hace con



**Servicio que va a consumir el cliente**

package api.crazy.demo.infraestructure.DAO;

import java.util.List;

import org.springframework.stereotype.Service;

import lombok.AllArgsConstructor;

@Service

@AllArgsConstructor

public class ExampleService {

private final ExampleRepository exampleRepository;

public List<ExampleEntity> getAll(){

var x = exampleRepository.findAll();

return x;

}

}

Podemos observar varias cosas:

* Service: significa que puede ser inyectado y no tiene que ser instanciado.
* Tiene un constructor LOMBOK que inyecta el contrato.

Pero nada de esto nos sirve debido a que nosotros necesitamos indicarle a spring boot que todo lo que nosotros tenemos puede ser inyectado.

**4 - Registrar el JAP para que pueda ser utilizado**

Nosotros tenemos que indicarle al spring boot donde está el DAO. Tenemos que indicarle donde están los contratos de repositorios y donde están las entidades de mapeo a bases de datos. y ello lo hacemos agregando @ a el archivo principal de spring boot (demo\Application.java)

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

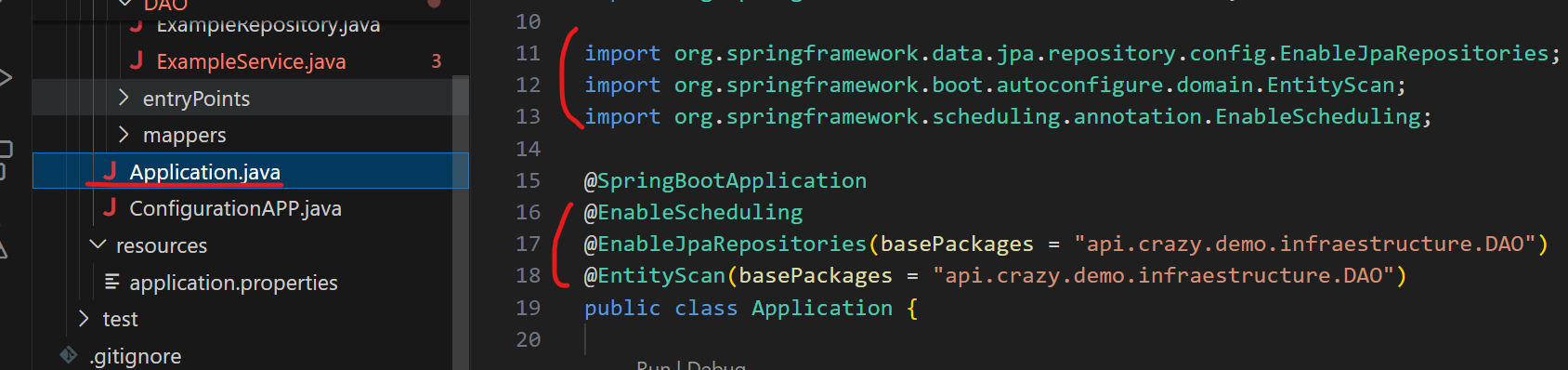
import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.scheduling.annotation.EnableScheduling;

@EnableScheduling

@EnableJpaRepositories(basePackages = "api.crazy.demo.infraestructure.DAO")

@EntityScan(basePackages = "api.crazy.demo.infraestructure.DAO")



**5 - habilitar la inyección de casos de uso en todo el proyecto:**

Tenemos que crear una nueva clase que esté al mismo nivel de (demo\Application.java) la vamos a llamar ConfigurationAPP.java y va a tener el siguiente código:

package api.crazy.demo;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.FilterType;

@Configuration

@ComponentScan(

basePackages = "api.crazy.demo.domain.usecases",

includeFilters = {

@ComponentScan.Filter(type = FilterType.REGEX, pattern = "^.+UseCase$"),

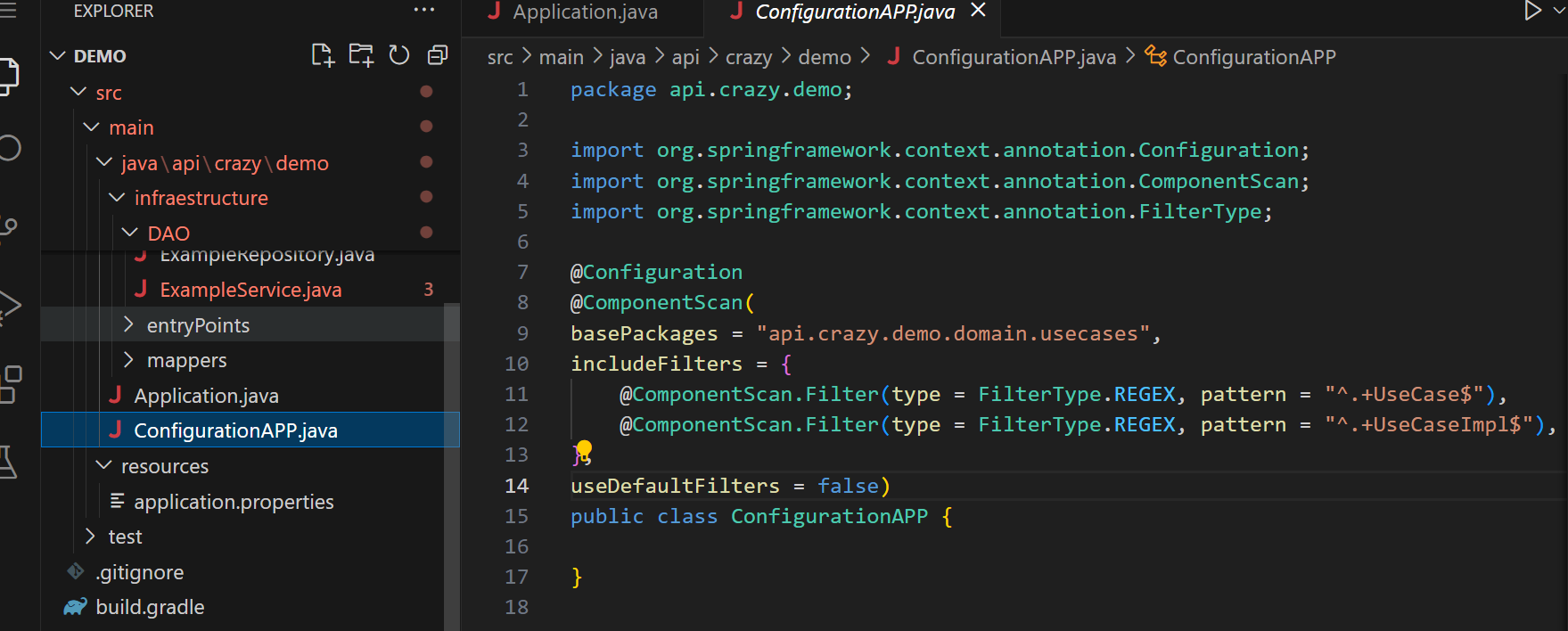
@ComponentScan.Filter(type = FilterType.REGEX, pattern = "^.+UseCaseImpl$"),

},

useDefaultFilters = false)

public class ConfigurationAPP {

}



**6 - Mapear de base de datos a nuestra clase de java**

package api.crazy.demo.infraestructure.mappers;

import api.crazy.demo.domain.entities.Example;

import api.crazy.demo.infraestructure.DAO.ExampleEntity;

public class ExampleMapper {

public static ExampleEntity toEntity(Example example) {

return new ExampleEntity(example.getId(), example.getName());

}

public static Example toDomain(ExampleEntity entity) {

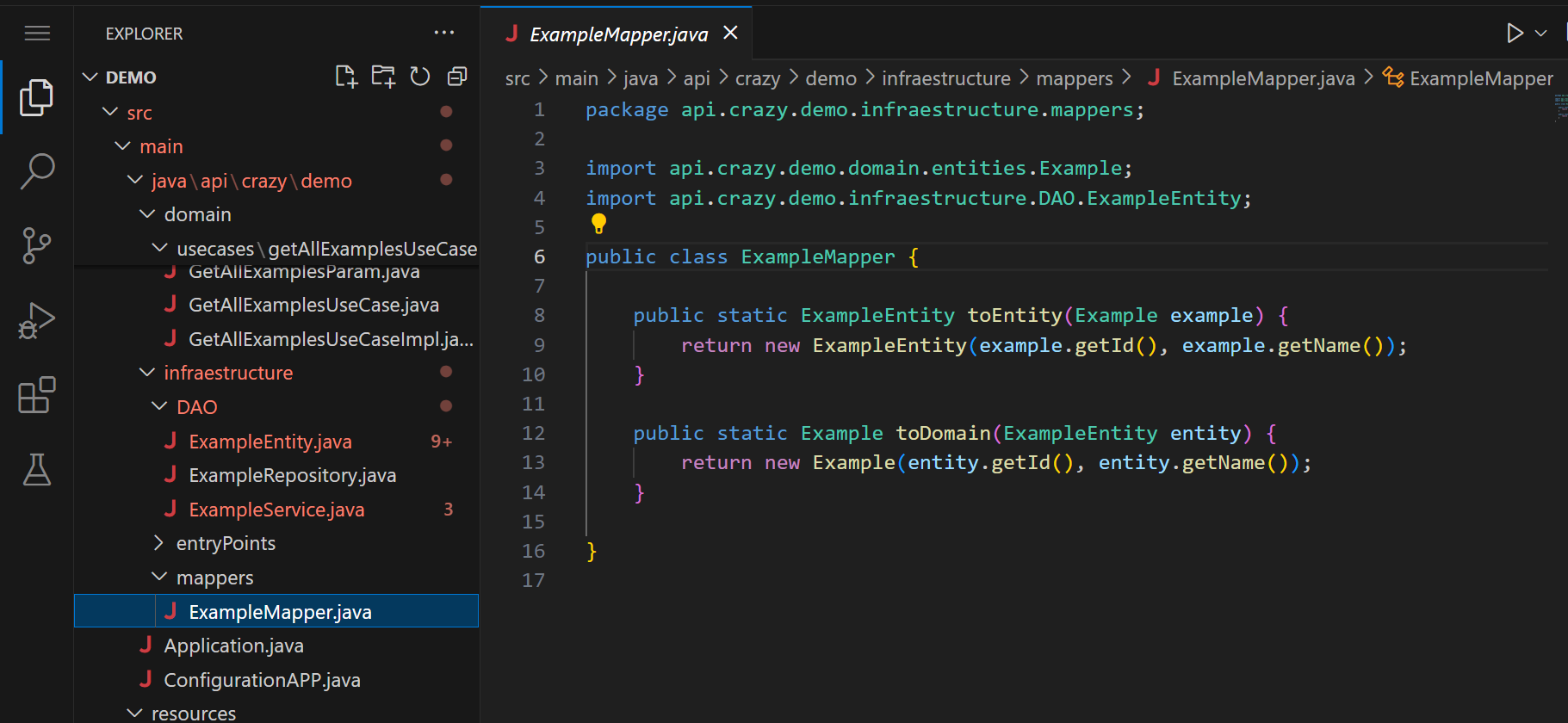
return new Example(entity.getId(), entity.getName());

}

}

Resulta que lo único que hemos hecho hasta el momento es ir hasta la base de datos con nuestra Entidad pero esa entidad no tiene nada que ver con nuestro dominio… tenemo que crear un convertidor que convierta de JPA a Java y eso lo hacemos creando una nueva capa (en infraestructure) que se va a llamar mappers (demo\infraestructure\mappers)

y es una clase muy simple que va a retornar la instancia de un objeto o el otro



**7 - Asociar nuestro JPA con nuestro caso de USO**

package api.crazy.demo.domain.usecases.getAllExamplesUseCase;

import java.util.List;

import java.util.ArrayList;

import org.springframework.stereotype.Component;

import api.crazy.demo.domain.entities.Example;

// LOAD DATA FROM DATABASE

import api.crazy.demo.infraestructure.DAO.ExampleEntity;

import api.crazy.demo.infraestructure.DAO.ExampleService;

//Convert database info in Java Class

import api.crazy.demo.infraestructure.mappers.ExampleMapper;

@Component //This decorator permit injection

public class GetAllExamplesUseCaseImpl extends GetAllExamplesUseCase {

private final ExampleService exampleService;

public GetAllExamplesUseCaseImpl(ExampleService exampleService){

this.exampleService = exampleService;

}

@Override

public List<Example> call(GetAllExamplesParam param) {

List<Example> x = new ArrayList<>();

List<ExampleEntity> y = exampleService.getAll();

for (int i = 0; i < y.size(); i++) {

x.add(ExampleMapper.toDomain(y.get(i)));

}

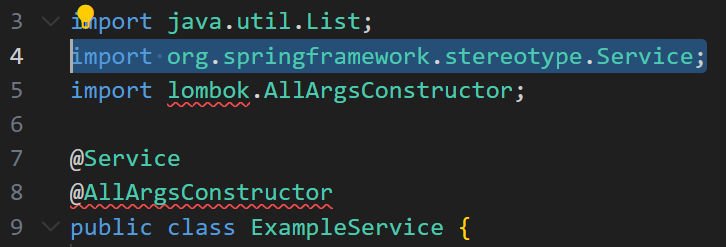
return x;

}

}

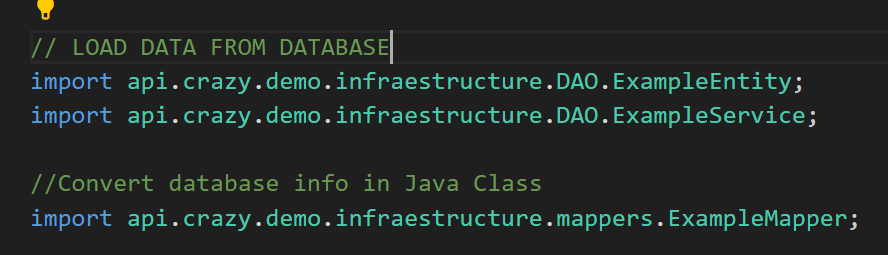
Lo único que tenemos que hacer es inyectar el servicio en nuestro caso de uso y utilizar el mapper para ver todas las entidades.

*“Recuerden que ya configuramos el servicio para ser injectable”*

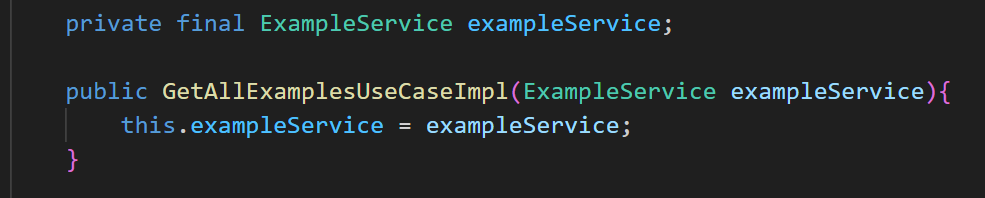


Solo tenemos que importarlo e indicarlo en el constructor del caso de uso.

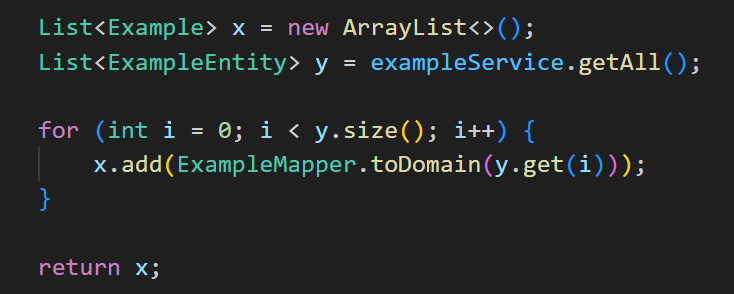
Importaciones necesarias



Inyectar el caso el servicio

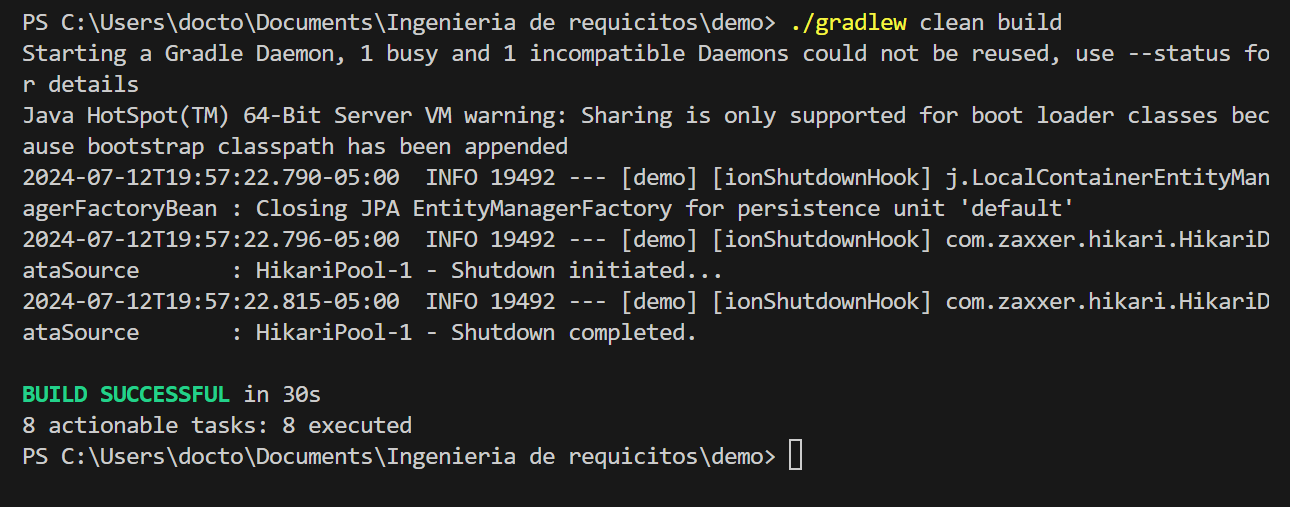


Usar:



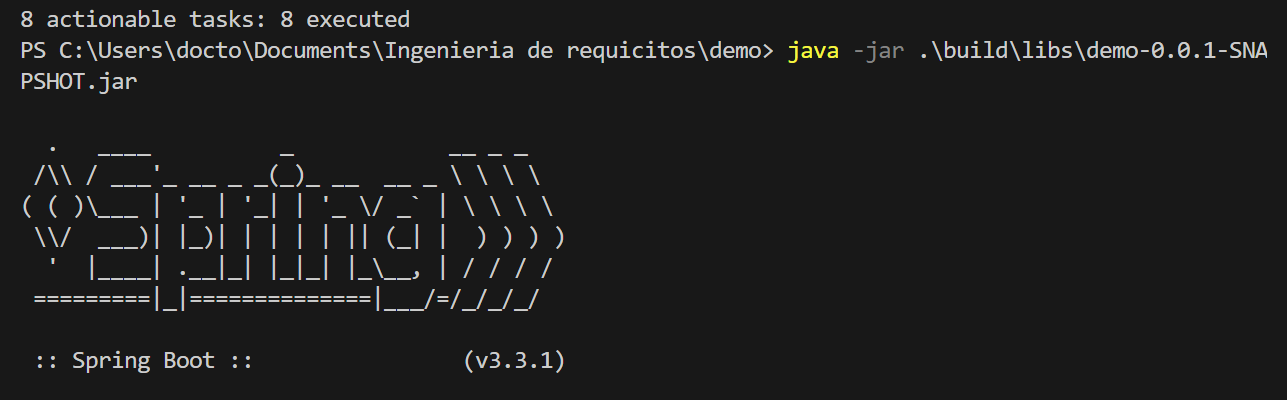
Y ahora vamosa a compilar el proyecto:

./gradlew clean build

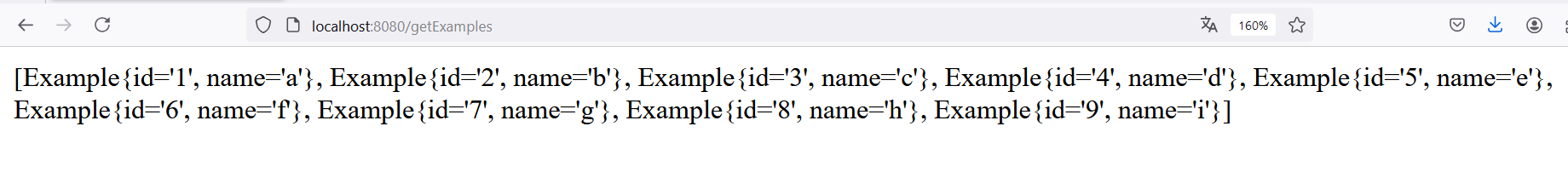


Y luego vamos a levantarlo:

java -jar .\build\libs\demo-0.0.1-SNAPSHOT.jar



Y vamos a acceder al API para ver nuestras entidades de la base de datos:



Nota: yo puse un stringToString() en el entity