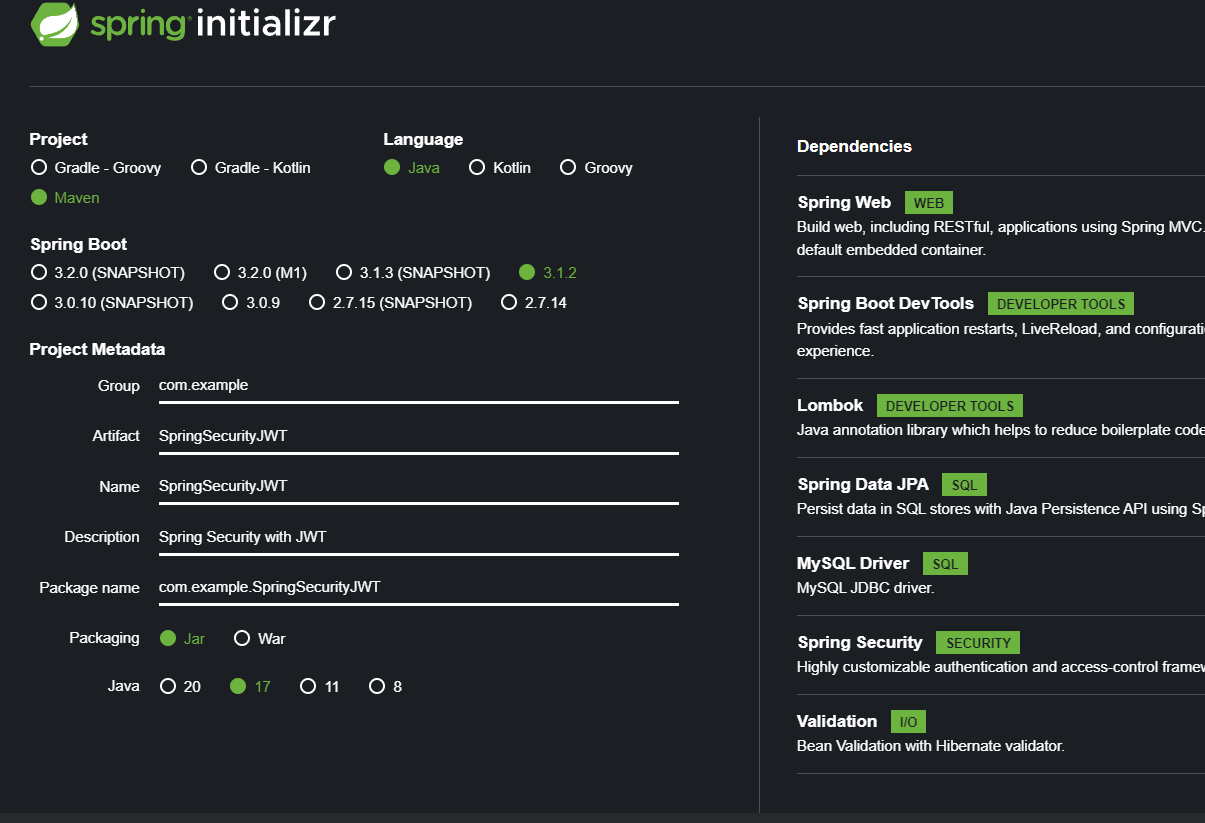
Spring Security with JWT

1 - Vamos a generar el Proyecto en spring initializr



2 – Una vez importado el Proyecto buscamos Spring boot banner en inet y generamos el siguiente banner

,---. ,--. ,---. ,--. ,--. | | ,--.,--. ,--.,--------.

' .-' ,---. ,--.--.`--',--,--, ,---. ' .-' ,---. ,---.,--.,--.,--.--.`--',-' '-.,--. ,--. ,---| |---. | || | | |'--. .--'

`. `-.| .-. || .--',--.| \| .-. |`. `-.| .-. :| .--'| || || .--',--.'-. .-' \ ' / '---| |---' ,--. | || |.'.| | | |

.-' | '-' '| | | || || |' '-' '.-' \ --.\ `--.' '' '| | | | | | \ ' | | | '-' /| ,'. | | |

`-----'| |-' `--' `--'`--''--'.`- / `-----' `----' `---' `----' `--' `--' `--' .-' / `--' `-----' '--' '--' `--'

“Spring Security + JWT”

3 – Ahora vamos a las dependencias y comentamos la siguiente (y luego recargamos el maven/proyecto)

<!-- <dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency> -->

4 – Ahora vamos a configurar la DB

Vamos a application.properties y escribimos la siguiente configuracion

spring.datasource.url=jdbc:mysql://localhost:3306/securitydb

spring.datasource.username=root

spring.datasource.password=Asdf7928

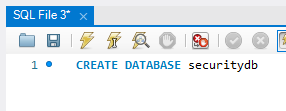
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver

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

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create

Vamos a mySQL y creamos la table securitydb



Una vez hecho esto, la app ya funciona y se puede comenzar a trabajar en ella

5 – Dentro del paquete com.example.SpringSecurityJWT creamos el paquete models donde iran los modelos de las clases

La primer clase a crear es UserEntity

*@Entity*

*@Data*

*@AllArgsConstructor*

*@NoArgsConstructor*

*@Builder*

*@Table*(name="users")

public class UserEntity {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

*@Email*

*@NotBlank*

*@Size*(max = 80)

private String email;

*@NotBlank*

*@Size*(max = 30)

private String userName;

*@NotBlank*

private String password;

}

Para la proxima clase que se va a llamar RoleEntity, primero creamos un enum llamado ERole que indicara el tipo de rol para cada usuario

public enum *ERole* {

***ADMIN***, ***USER***, ***GUEST***

}

Una vez creado el enum, creamos la clase RoleEntity

*@Data*

*@AllArgsConstructor*

*@NoArgsConstructor*

*@Builder*

*@Entity*

*@Table*(name = "roles")

public class RoleEntity {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

*@Enumerated*(*EnumType*.***STRING***)

private *ERole* name;

Tener en cuenta que la anotacion Enumerated lleva dentro de los args un EnumType.String ya que si bien es un enum, aquello que esta dentro del mismo, al fin y al cabo es un String

6 – Es necesario configurar la relacion entre las entidades mediante una tabla intermedia ya que la relacion que existe entre UserEntity y RoleEntity es de muchos a muchos; y, dado que la clase que nos interesa desarollar es UserEntity, solo en ella vamos a declarer lo necesario

Empezamos por agregar un atributo privado del tipo . Esto es ya que Set, a diferencia de List u otro tipo de arreglo, no permite duplicados (esto es para que el usuario no tenga 2 veces declarado un mismo valor de tipo de usuario)

Ahora le agregamos las anotaciones:

*@ManyToMany*(fetch = *FetchType*.***EAGER***)

Establece el tipo de relacion entre las clases. Se seleciona FetchType.Eager ya que esta eleccion nos va a traer todos los roles que contiene la clase; a diferencia de Lazy que solo nos trae lo solicitado

*@ManyToMany*(fetch = *FetchType*.***EAGER***, targetEntity = RoleEntity.class

Esto va a declarar la clase con la que se va a generar la relacion

*@ManyToMany*(fetch = *FetchType*.***EAGER***, targetEntity = RoleEntity.class, cascade = *CascadeType*.***PERSIST***)

El .PERSIST hace que al momento de eliminar un usuario, no se borre también el Rol. (esto romperia la db y dejaria inhabilitados a los demas usuarios con roles invalidos)

Ahora creamos la table intermedia:

*@JoinTable*(name = "user\_roles"

Indica el nombre de la table intermedia

*@JoinTable*(name = "user\_roles", joinColumns = *@JoinColumn*(name = "user\_id")

El joinColumn indica cual es la clave foranea de la clase UserEntity

*@JoinTable*(name = "user\_roles", joinColumns = *@JoinColumn*(name = "user\_id"), inverseJoinColumns = *@JoinColumn*(name = "role\_id"))

El inverseJoinColumns indica la clave foranea de la clase RoleEntity

Si volvemos a correr la app, se ejecutaran directamente las querys de creacion de las tablas y las relaciones entre las mismas y al refrescar la DB deberian aparecer las nuevas tablas

