Documentacion rest api springboot

Esteban Herrera Escofet

## Introducción

En esta documentación vamos a ver el funcionamiento y contenido de una rest API con autenticación y autorización con token temporales que contiene los modelos user para poder acceder y obtener los permisos permisos necesarios para poder acceder al modelo customers, en el modelo customers tendremos diversas opciones a realizar, como listar, crear, modificar o borrar customers.

## Tablas

La API consta de dos tablas en mysql que son las siguientes:

Diagrama

Descripción generada automáticamente

## Versión y dependencias

La versión de Java que usamos para el proyecto será la 17 y la de SpringBoot será la 3.0.6

Las dependencias que usaremos serán las siguientes:

spring-boot-starter-data-jpa

spring-boot-starter-security

spring-boot-starter-web

spring-boot-devtools

mysql-connector-j

spring-boot-starter-test

h2

spring-security-test

lombok

jjwt-api

jjwt-impl

jjwt-jackson

## Conexión a BBDD

En el archivo application.properties podemos ver la conexión a la base de datos:

Texto

Descripción generada automáticamente

## Modelos

Captura de pantalla de computadora

Descripción generada automáticamente

## Controller

Aquí podemos observar el controlador para el modelo customer que nos permite realizar las diferentes acciones, con GET, POST, PUT y DELETE, cada una con su path correspondiente, el path raíz será /CRUDRepo:

Texto

Descripción generada automáticamente

Para poder llevar a cabo estas acciones necesitaremos los siguientes archivos de service que se comunicara con su respectivo repositorio:

Texto

Descripción generada automáticamente

## Repositorios

Los repositorios necesarios serán los siguientes:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Seguridad

Para la seguridad usaremos JWT, donde crearemos tokens para autenticación de usuarios y autorización.

Los archivos que usaremos para ello serán:

Texto

Descripción generada automáticamente

Aquí podemos ver userdetailsImpl.java y el webSecurityConfig.java

Texto

Descripción generada automáticamente

En userdetailsImpl implementamos userdetails con sus propiedades y en websecurityconfig podemos ver como creamos un filterchain que se comunicará con el autenticador y el autorizador, asignara el path /login y hará uso de las clases :

UsernamePasswordAuthenticationFilter.class

AuthenticationManagerBuilder.class

Del SecurityFilterChain y del AuthenticacionManager, seguidamente observaremos un Bean para crear un passwordEncoder.

Seguidamente en TokenUtils y en UserDetailService podremos ver el siguiente código:

Texto

Descripción generada automáticamente

En token utils podemos ver como hacemos uso de un ACCESS\_TOKEN\_SECRET:

    private final static String ACCESS\_TOKEN\_SECRET = "4qhq8LrEBfYcaRHxhdb9zURb2rf8e7Ud";

y de un ACCES\_TOKEN\_VALIDITY\_SECONDS, que en nuestro caso será de 24 horas

 private final static Long ACCES\_TOKEN\_VALIDITY\_SECONDS = 86\_400L;// 24Horas

Seguidamente en la creación del token lo multiplicaremos por 1000 al estar en milisegundos y crearemos un map llamado extra para el caso que haya que añadir valores.

Finalmente devolveremos con un return un Jwts con las siguientes propiedades:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Luego pasamos al UsernamePasswordAuthenticationToken donde haremos un try/catch donde haremos un claim que será un parserBuilder, un builder a la inversa y lo añadiremos al collections al retornar el usernamepasswordAutentcationToken que lo asignará a email.

En UserDetailServiceImpl podemos ver como hará uso del userdetailsservice, secomunicará con el repositorio del usuario y haremos uso del findOneByEmail en caso de lo encuentre, en caso contrario nos devolverá un mensaje de error.

Seguidamente veremos los archivos JWTAuthenticationFilter y JWTAuthorizationFilter :

Texto

Descripción generada automáticamente

En JWTAuthenticationFilter podemos ver como hacemos un extend UsernamePasswordAuthenticationFilter, también haremos uso del authcredentials que mostraremos posteriormente, para finalmente crear el usernamepasswordauthenticationtoken .

Luego Veremos el successfulAuthentication que se encargará de crear el token con el header “bearer “ en caso de que la autenticación se haya completado correctamente.

En JWTAuthorizationFilter pasamos a la fase de autorización donde crearemos un if que hará la comprobación de que en caso del que el bearerToken no este vacio y que empiece con el header “Bearer “ le quitara el bearer y se lo asignará a la variable usernamePAT haciendo uso de tokenutills.getauthentication, seguidamente se lo asignaremos al securityContextHolder.

En authcredentials.java solo veremos la clase que contiene los atributos necesarios para poder loguearse, que en nuestro caso es email y password

Accediendo a la API

En la base de datos tabla user tenemos los siguientes usuarios, con sus passwords ya codificadas:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

En Web Security config hemos creado un system.oup.prinln para poder ver como seria nuestra contraseña codificada, que es el siguiente:

    public static void main(String[] args) {

        System.out.println("pass: " + new BCryptPasswordEncoder().encode("jose"));

    }

Tenemos que el usuario jose tiene un email que es [jose@jose.com](mailto:jose@jose.com) con la contraseña jose encriptada.

Accedemos a Postman para realizar el loguin después de haber arrancado la aplicación con un mvn spring-boot:run

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Aquí podemos ver que nos da un codigo 200 ok

Seguidamente copiamos el bearer y lo añadimos en la pestaña authorization > bearertoken

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Volvemos a la pestaña del CRUDpara realizar alguna acción sobre customers, por ejemplo listarla:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Y nos deja realizar la acción dado que seguimos teniendo el código 200, luego si quisiéramos añadir un nuevo customer usaríamos el siguiente path con POST:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para Modificar un customer lo haríamos con PUT de la siguiente forma:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para Buscar un Customer usaríamos su ID con el GET:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Y finalmente para eliminar un customer usaríamos el DELETE y su ID:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente