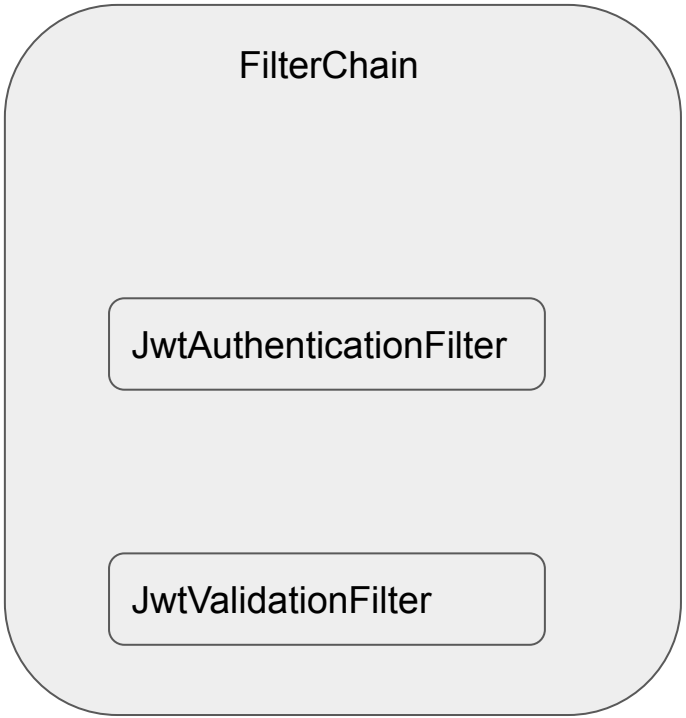
Protección de rutas

Obtener token en /login

Acceso a recursos protegidos



Encriptar password

```
public User save(User user)
```

Login con Jpa

```
@Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username)
```

Token Jwt

JwtAuthenticationFilter

```
void successfulAuthentication
```

JwtValidationFilter

```
void doFilterInternal
```

Protección de rutas

Se agregó la dependencia

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
```

crear SpringSecurityConfig

Vamos a crear el método `filterChain`
que devuelve una cadena de filtros de springSecurity, `SecurityFilterChain`
con la reglas necesarias
Se debe anotar con @Bean , lo que devuelve el método se guarda en el contexto de spring por estar anotado como `@Configuration`

El csrf se desactiva, ya que es la seguridad para cuando se usa mvc, (front desde java)

Configuramos la sesión , en nuestro caso no se guarda, si se guarda el token, la configuramos sin estados, esto lo manejamos del lado de react.

SpringSecurityConfig

Protección de rutas

```
→ DefaultSecurityFilterChain
@Configuration
public class SpringSecurityConfig {


    → 1 bean | ← 1 bean
    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        return http.authorizeHttpRequests()
                .requestMatchers(HttpMethod.GET, "/users").permitAll()
                .anyRequest().authenticated()
                .and()
                .csrf(config -> config.disable())
                .sessionManagement(managment -> managment.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .build();

    }
}
```

Con versiones superiores de sprinboot 3.0.4 el método `authorizeHttpRequests()` esta deprecade

Con el siguiente código si levantamos el proyecto , se debería tener solo acceso a GET /users

Obtener Token

Vamos a crear la clase JwtAuthenticationFilter para realizar el esquema siguiente , para eso esa clase se la debemos pasar al filterChain

JwtAuthenticationFilter

Obtener Token

Vamos a generar una nueva clase y extender de extend UsernamePasswordAuthenticationFilter

y vamos a implementar tres metodos , `attemptAuthentication`
se va llamar cuando existe un POST /login

```java
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {


    private AuthenticationManager authenticationManager;

    public JwtAuthenticationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }



    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
            throws AuthenticationException {


        }


    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
            Authentication authResult) throws IOException, ServletException {

        }

    protected void unsuccessfulAuthentication(HttpServletRequest request, HttpServletResponse response,
    AuthenticationException failed) throws IOException, ServletException {
```

JwtAuthenticationFilter

Obtener Token

aca extraemos de request el username y password y luego lo autenticamos , para la autenticación hay que implementar el UserDetailService

```
Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
 throws AuthenticationException {

    User user = null;
    String username = null;
    String password = null;

    try {
        user = new ObjectMapper().readValue(request.getInputStream(), valueType:User.class);
        username = user.getUsername();
        password = user.getPassword();

        logger.info("Username desde request InputStream (raw) " + username);
        logger.info("Password desde request InputStream (raw) " + password);

    } catch (StreamReadException e) {
        e.printStackTrace();
    } catch (DatabindException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(username, password);
    return authenticationManager.authenticate(authToken);
}
```

estas son los dos métodos ,que luedo de la auth se disparan según resultado

Obtener Token

```java
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
        Authentication authResult) throws IOException, ServletException {

    String username = ((org.springframework.security.core.userdetails.User) authResult.getPrincipal())
            .getUsername();
    String originalInput = "algun_token_con_alguna_frase_secreta." + username;
    String token = Base64.getEncoder().encodeToString(originalInput.getBytes());

    response.addHeader("Authorization", "Bearer " + token);

    Map<String, Object> body = new HashMap(null)<>();
    body.put("token", token);
    body.put("message", String.format("Hola %s, has iniciado sesion con exito!", username));
    body.put("username", username);
    response.getWriter().write(new ObjectMapper().writeValueAsString(body));
    response.setStatus(200);
    response.setContentType("application/json");
}

protected void unsuccessfulAuthentication(HttpServletRequest request, HttpServletResponse response,
AuthenticationException failed) throws IOException, ServletException {

    Map<String, Object> body = new HashMap<>();
    body.put("message", "Error en la autenticacion username o password incorrecto!");
    body.put("error", failed.getMessage());

    response.getWriter().write(new ObjectMapper().writeValueAsString(body));
    response.setStatus(401);
    response.setContentType("application/json");
}


}
```

aca se pueden ver los metodos respectivos en el proceso de autenticación

Obtener Token

una vez creada la clase con sus métodos debemos pasarla al filterChain

Obtener Token

```
→ DefaultSecurityFilterChain
@Configuration
public class SpringSecurityConfig {

    @Autowired
    private AuthenticationConfiguration authenticationConfiguration;

    → 1 bean | ← 1 bean
    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        return http.authorizeHttpRequests()
                .requestMatchers(HttpMethod.GET, "/users").permitAll()
                .anyRequest().authenticated()
                .and()
                .csrf(config -> config.disable())
                .addFilter(new JwtAuthenticationFilter(authenticationConfiguration.getAuthenticationManager()))
                .sessionManagement(managment -> managment.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .build();

    }
}
```

Para poder realizar la autenticacion que se llama en `authenticationManager.authenticate(authToken);`

se debe implementar la interface `UserDetailsService`
Primero vamos a harcodear la verificación para luego hacerlo con la base de datos.

```java
@Service
public class JpaUserDetailsService implements UserDetailsService{

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        if (!username.equals("admin")) {
            throw new UsernameNotFoundException(String.format("Username %s no existe en el sistema!", username));
        }

        List<GrantedAuthority> authorities = new ArrayList<>();
        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));

        return new User(username,
        "12345",
        true,
        true,
        true,
        true,
        authorities);

    }
```

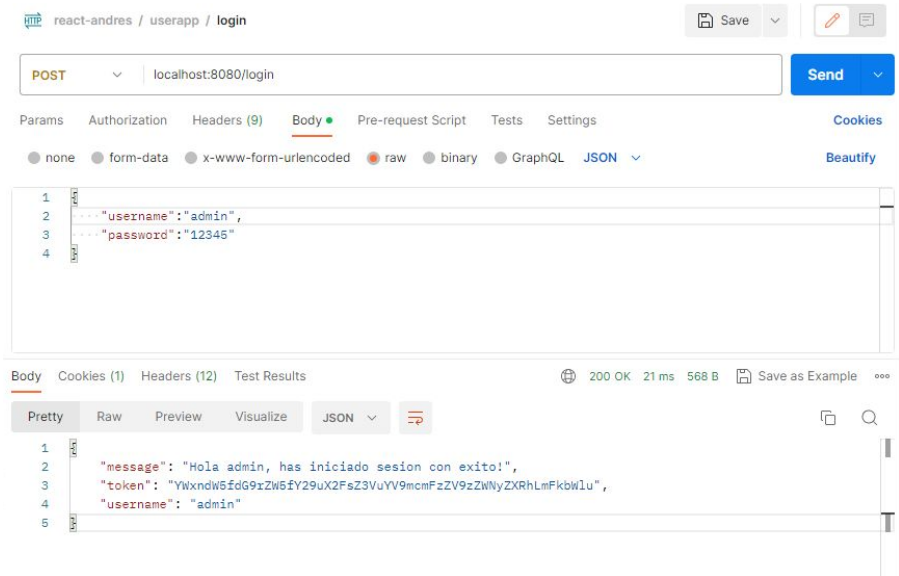
Importante:
este es el password esperado

Antes de probar debemos agregar al SpringSecurityConfig un bean que luego va a codificar el TOKEN por ahora es `NoOpPasswordEncoder`

```
  @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }
```

ACá ya podemos probar ,debe devolver el token

Obtener Token



BCryptPasswordEncoder es un encriptacion en una sola via, irreversible, esto lo maneja al crear este bean SpringSecurity lo usa por detrás, en este primera prueba devolvemos sin encriptar

para encripatar la contraseña agregamos al SpringSecurityConfig lo siguiente

```
@Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    AuthenticationManager authenticationManager() throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }
```

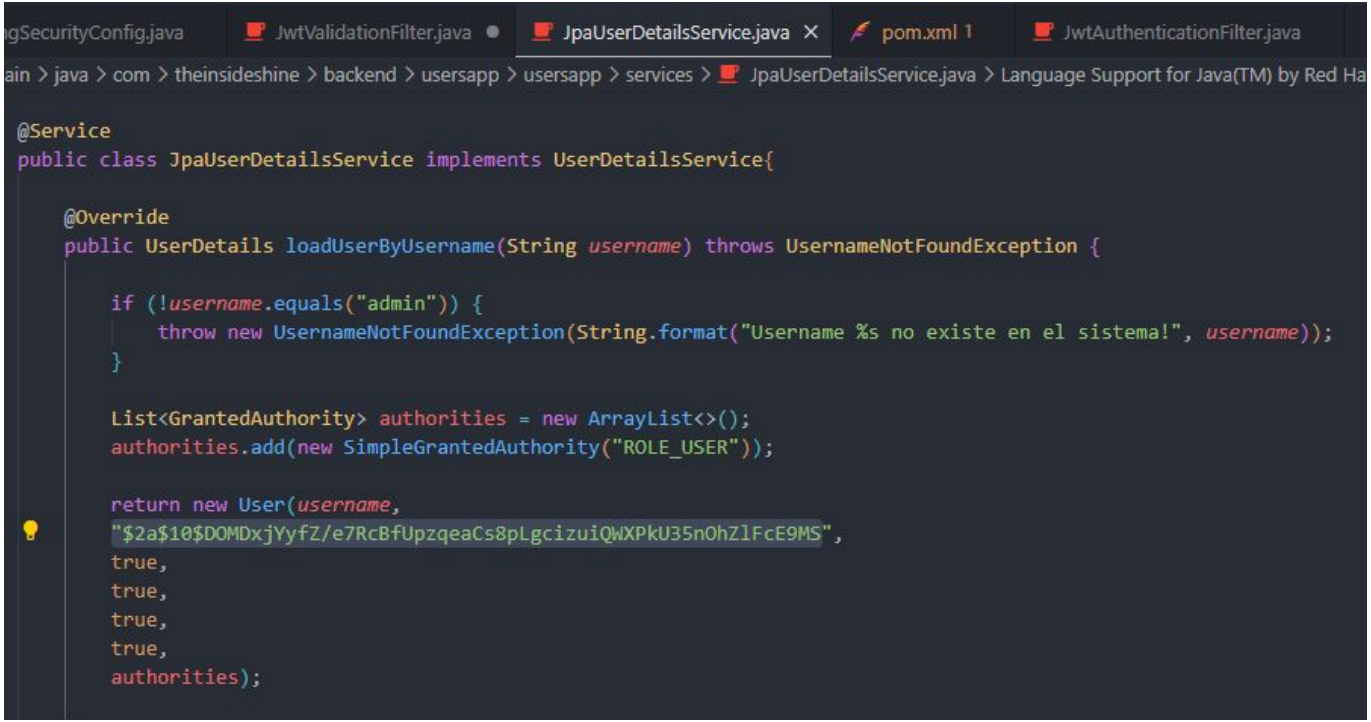
BCryptPasswordEncoder es un encriptacion en una sola via, irreversible, esto lo maneja al crear este bean SpringSecurity lo usa por detrás

Obtener Token

```
Configuration
ublic class SpringSecurityConfig {

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    AuthenticationManager authenticationManager() throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    ← 1 bean
    @Autowired
    private AuthenticationConfiguration authenticationConfiguration;

    → 1 bean | ← 1 bean
    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        return http.authorizeHttpRequests()
                .requestMatchers(HttpMethod.GET, "/users").permitAll()
                .anyRequest().authenticated()
                .and()
                .csrf(config -> config.disable())
                .addFilter(new JwtAuthenticationFilter(authenticationConfiguration.getAuthenticationManager()))
                .sessionManagement(managment -> managment.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .build();

    }
```

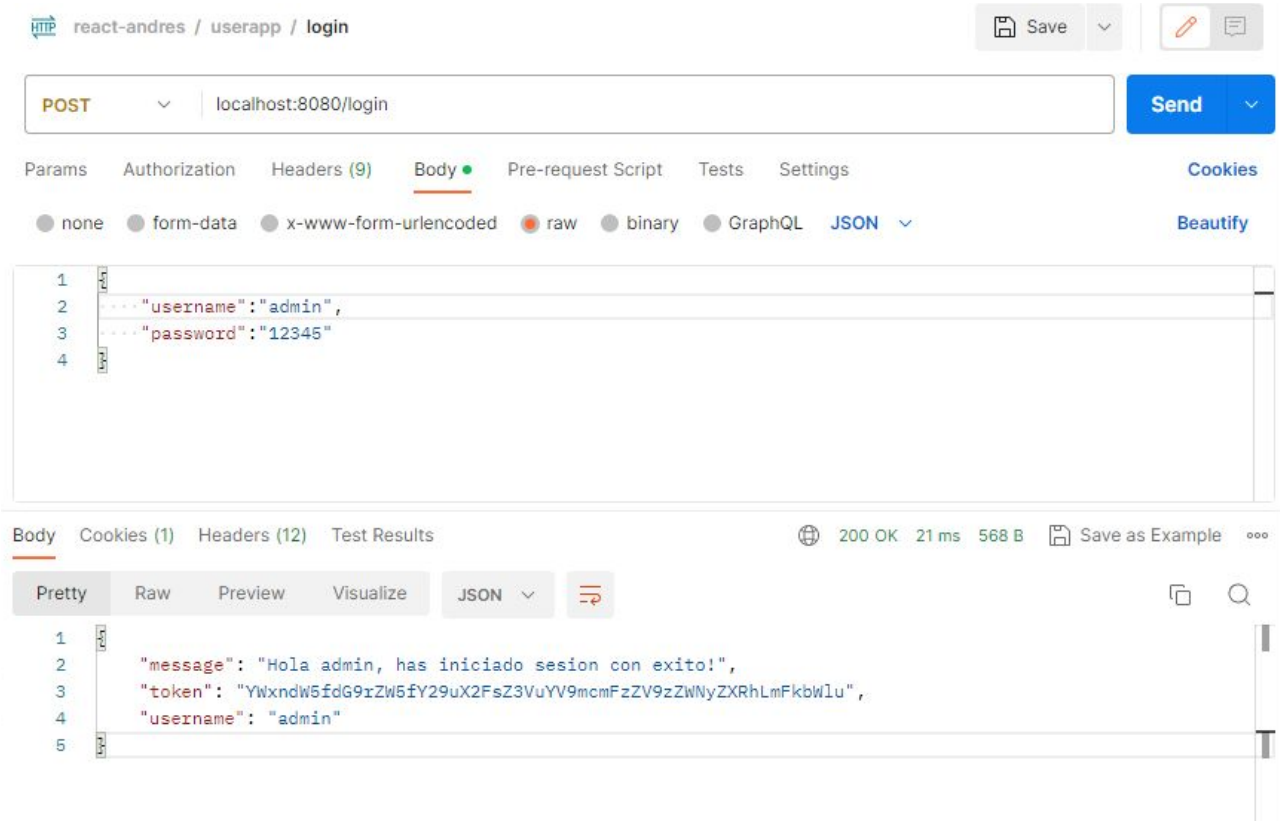
Debemos cambiar el password esperado, ya que ahora está encriptado

Obtener Token

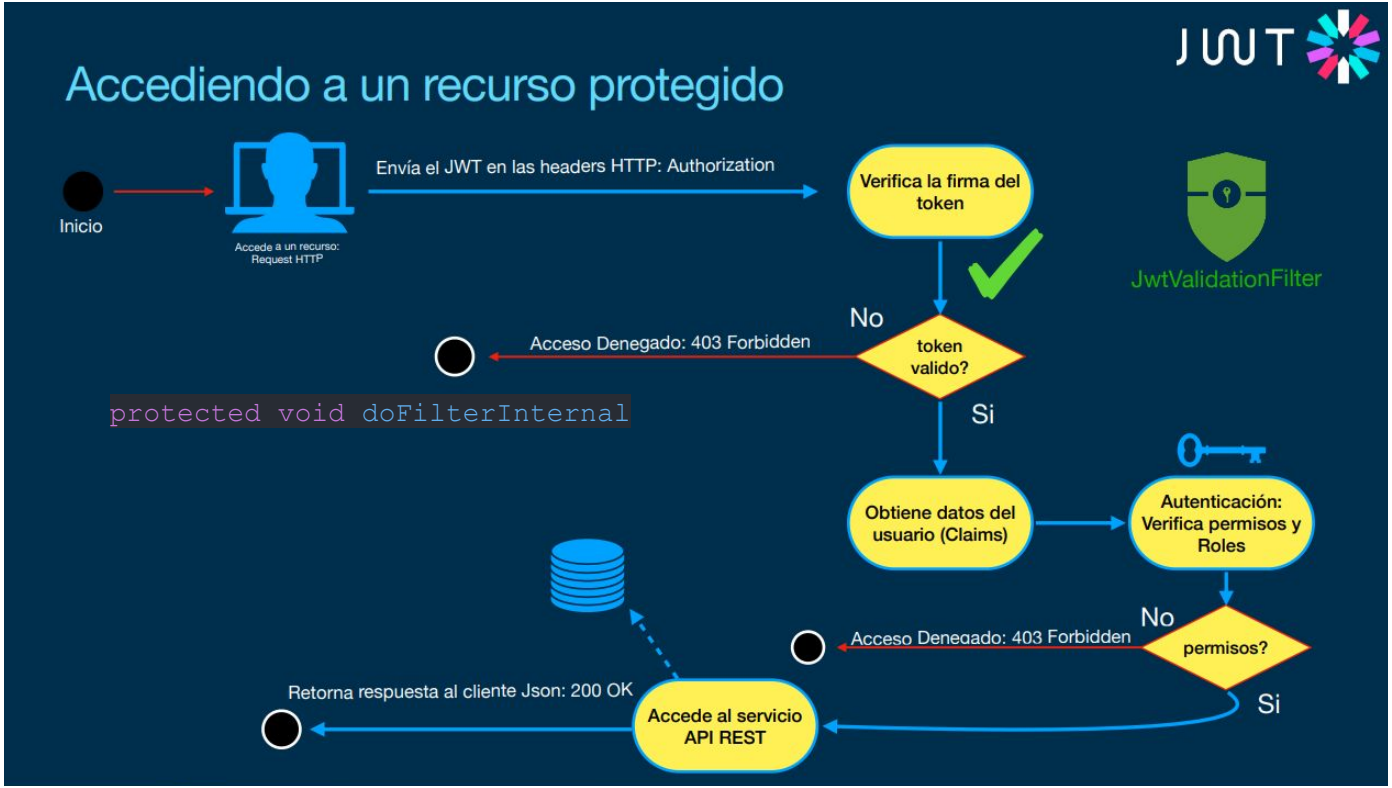


La misma prueba debe ser válida, pero por detrás el token está encriptado

Obtener Token

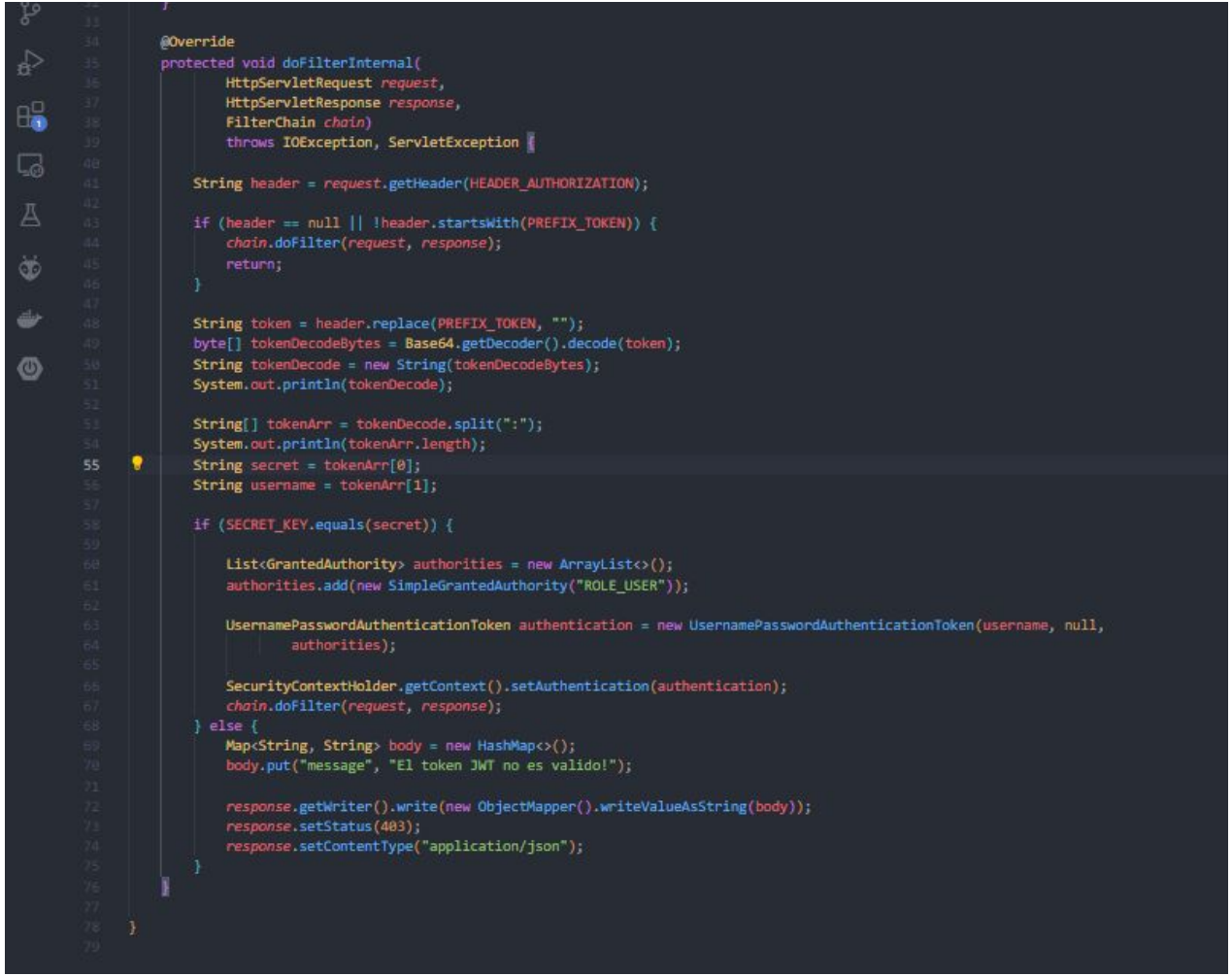


Acceso a recursos protegidos



Acceso a recursos protegidos

Vamos a crear otro filtro JwtValidationFilter y el metodo doFilterInternal



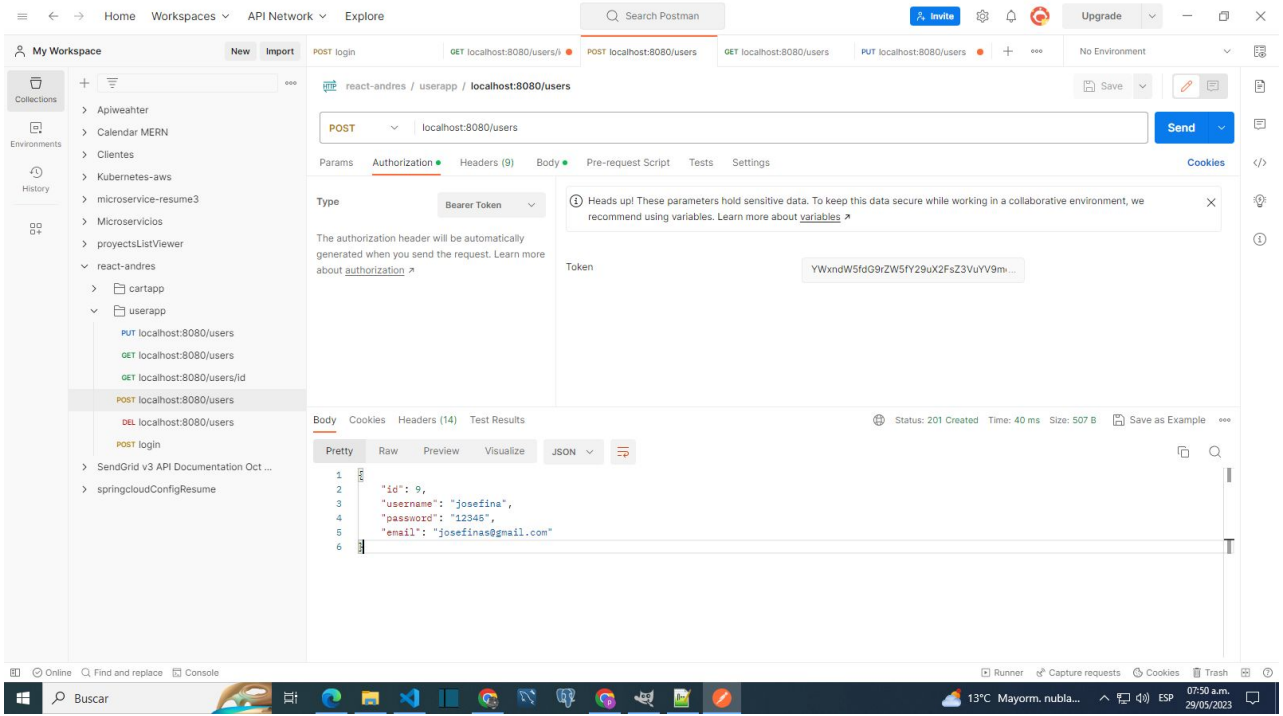
```java
@Override
protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {

    String header = request.getHeader(HEADER_AUTHORIZATION);

    if (header == null || !header.startsWith(PREFIX_TOKEN)) {
        chain.doFilter(request, response);
        return;
    }

    String token = header.replace(PREFIX_TOKEN, "");
    byte[] tokenDecodeBytes = Base64.getDecoder().decode(token);
    String tokenDecode = new String(tokenDecodeBytes);
    System.out.println(tokenDecode);

    String[] tokenArr = tokenDecode.split(":");
    System.out.println(tokenArr.length);
    String secret = tokenArr[0];
    String username = tokenArr[1];

    if (SECRET_KEY.equals(secret)) {

        List<GrantedAuthority> authorities = new ArrayList<>();
        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));

        UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(username, null,
                authorities);

        SecurityContextHolder.getContext().setAuthentication(authentication);
        chain.doFilter(request, response);
    } else {
        Map<String, String> body = new HashMap<>();
        body.put("message", "El token JWT no es valido!");

        response.getWriter().write(new ObjectMapper().writeValueAsString(body));
        response.setStatus(403);
        response.setContentType("application/json");
    }
}

}
```

Acceso a recursos protegidos

Hay que agregar el filtro a la cadena de rutas

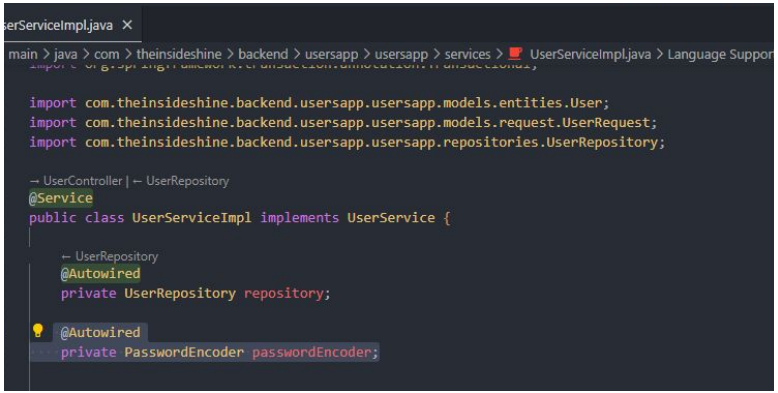
Acceso a recursos protegidos

ahora debe tener acceso a los recursos protegido si se pasa el token obtenido en login



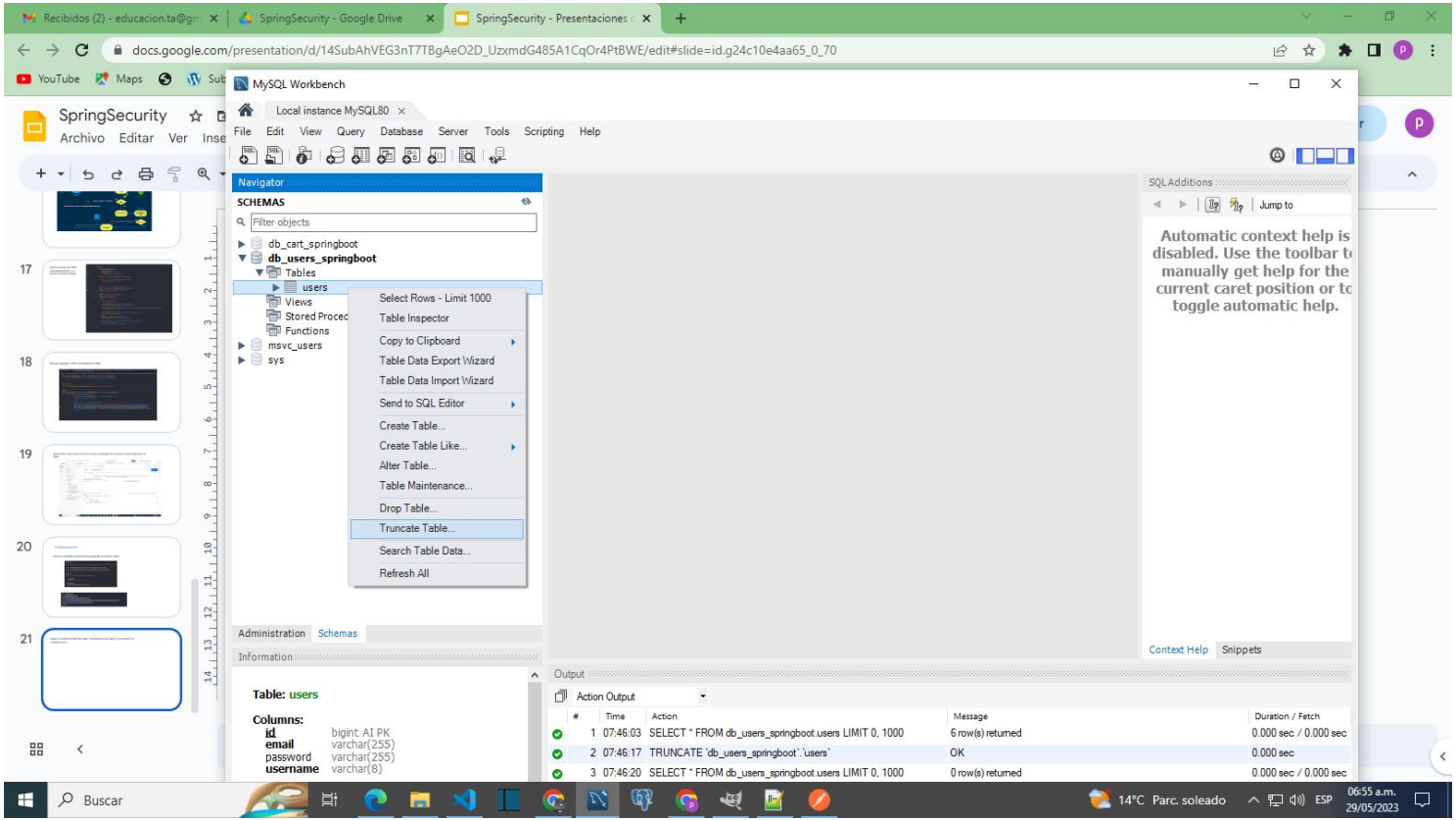
Encriptar password

vamos a encriptar el password al guardar en la base datos



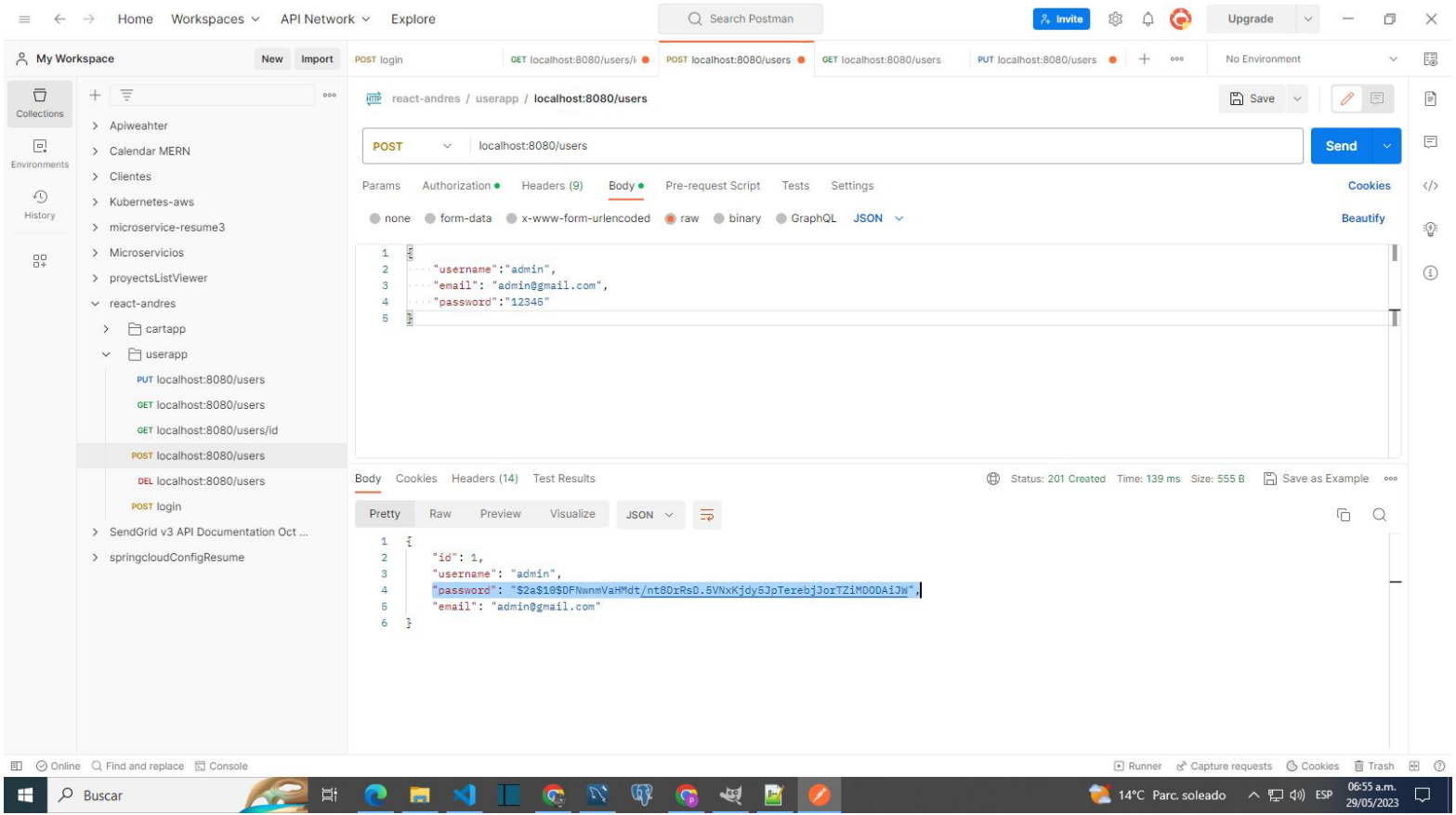
desde el worbechn truncate table, asi borramos los datos y reseteamos el contador de id

Encriptar password

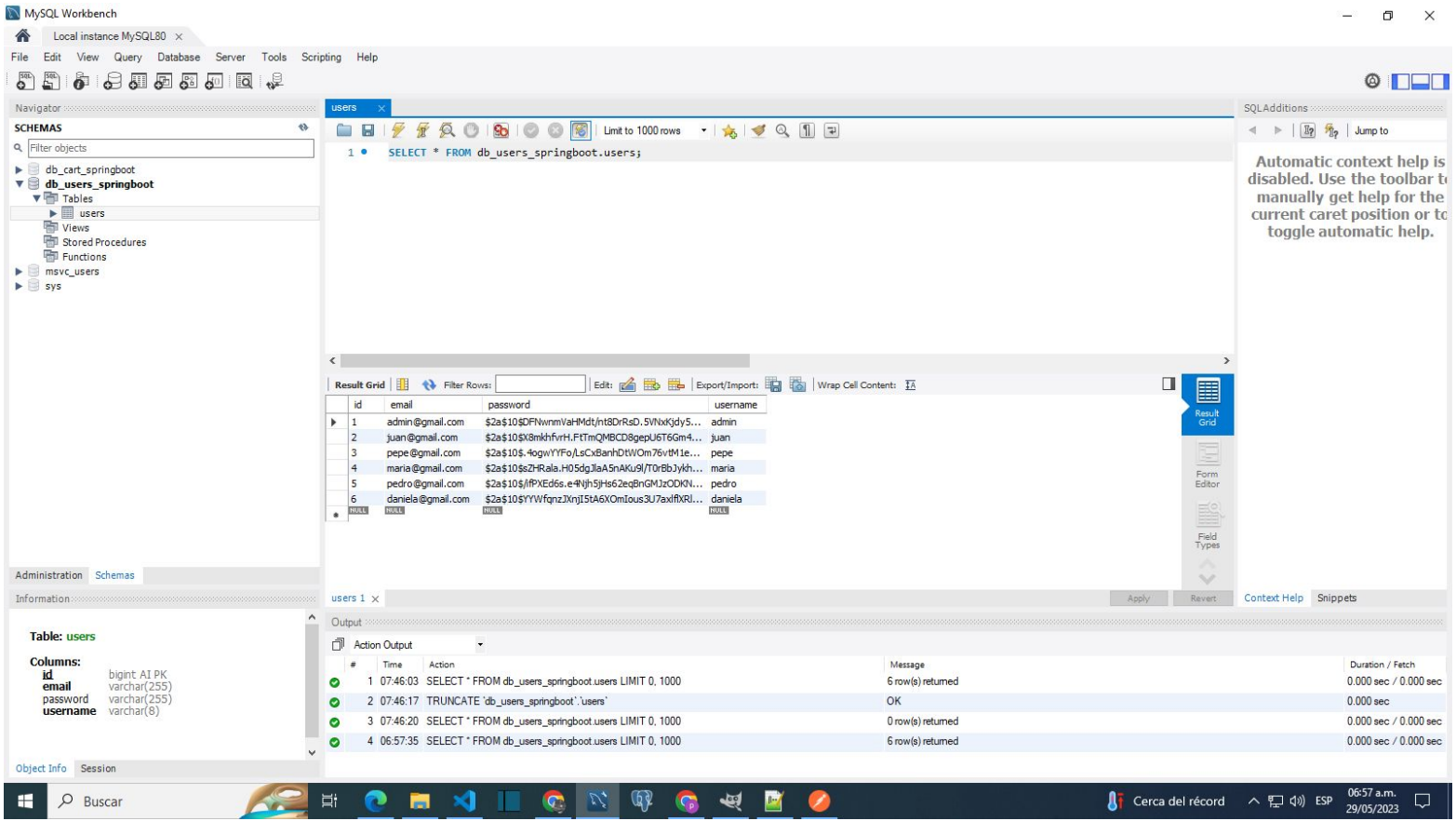


Encriptar password

volver a generar los usuarios, ahora el password debe verse encriptado



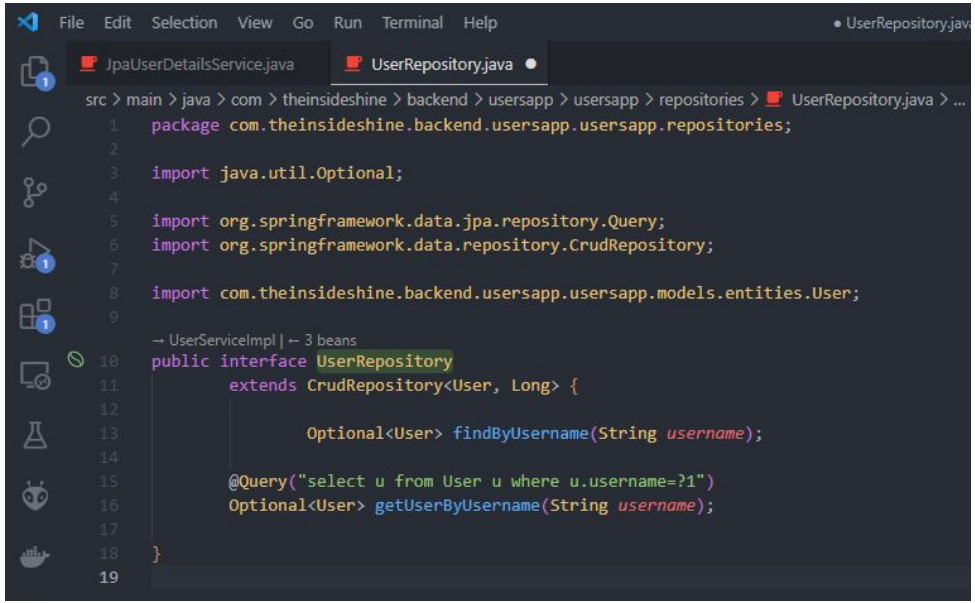
Encriptar password



Login con Jpa

vamos a hacer el login contra la BBDD,
IMPORTANTE: la comparación Con BCryp la hace por debajo SpringSecurity

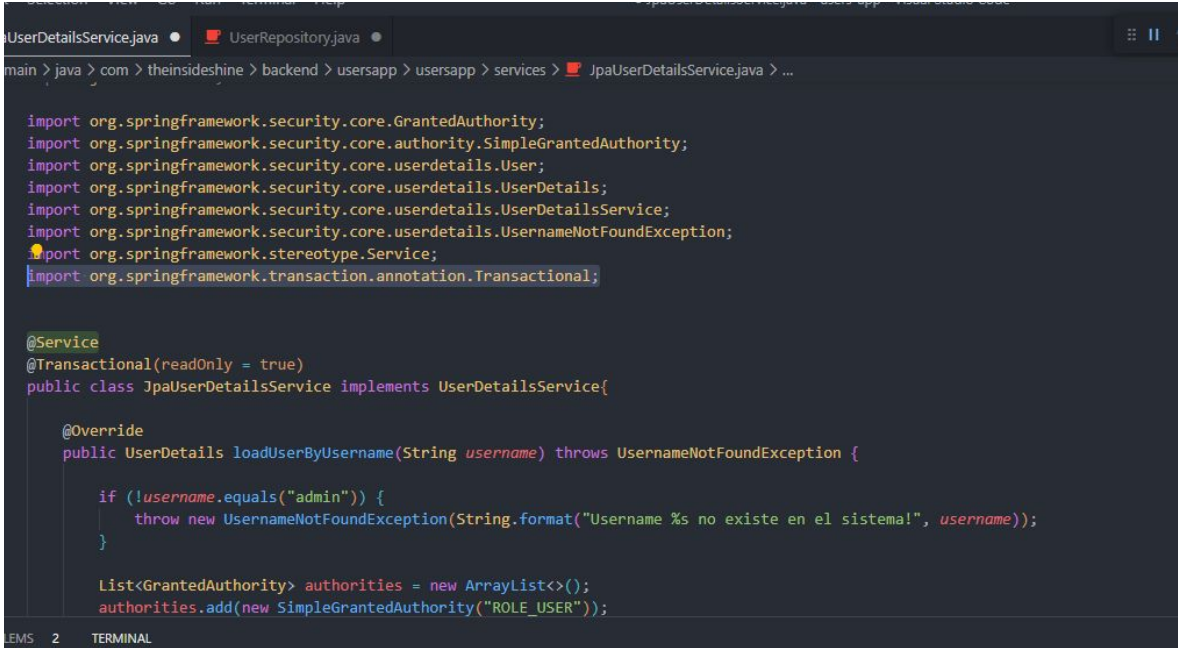
agregamos al busqueda por usuario en el repository



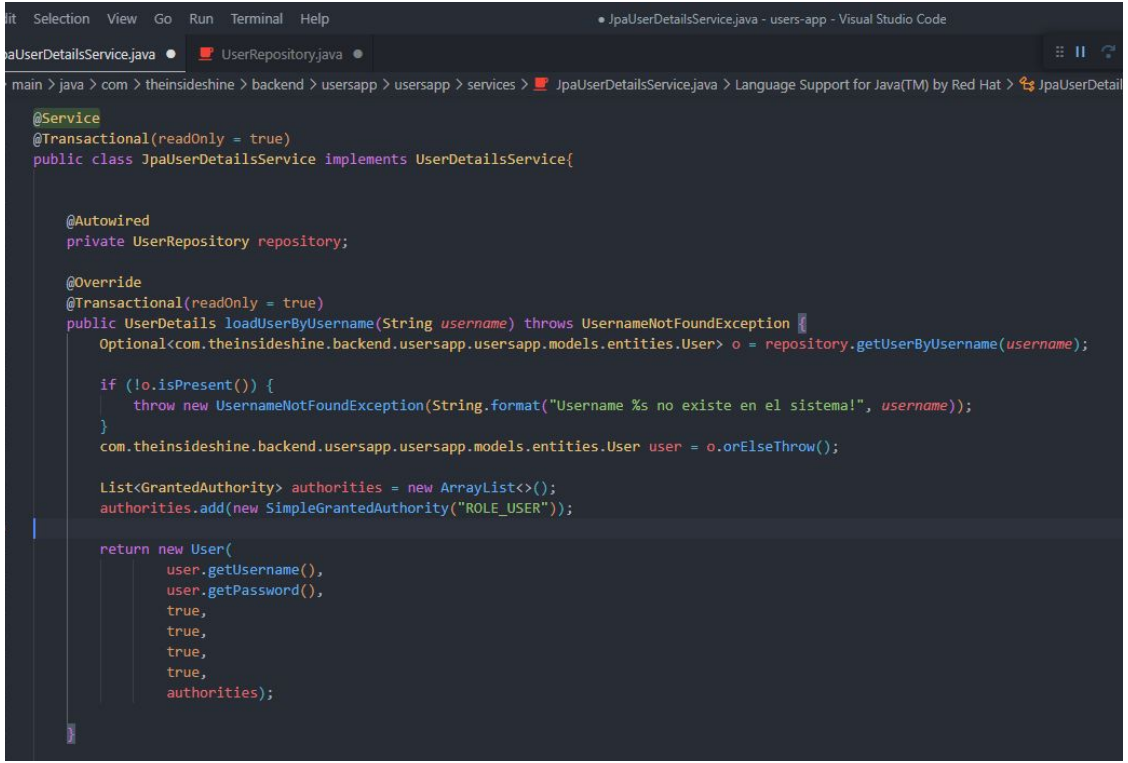
Login con Jpa

agregar a UserDetailsService el transaccional importado de springframework

agregamos al busqueda por usuario en el repository



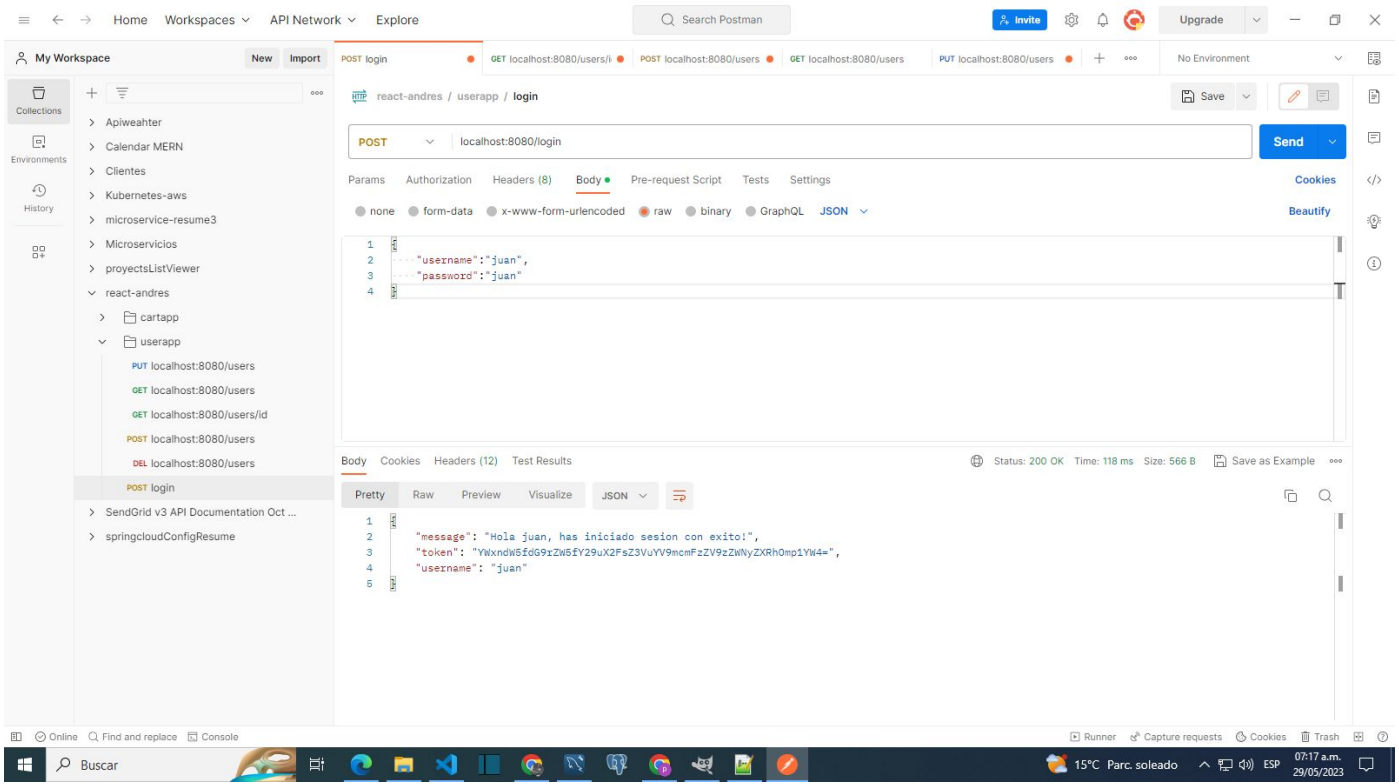
Login con Jpa



Nota: hay dos objetos que se llaman User, un es de SpringSecurity el otro del proyectos

Login con Jpa

Ahora debe hacerse el login con cualquier usuario en la BBDD



Token jwt

hasta ahora el token los encriptamos con base 64,

```java
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
        Authentication authResult) throws IOException, ServletException {

    String username = ((org.springframework.security.core.userdetails.User) authResult.getPrincipal())
            .getUsername();
    String originalInput = "algun_token_con_alguna_frase_secreta." + username;
    String token = Base64.getEncoder().encodeToString(originalInput.getBytes());

    response.addHeader("Authorization", "Bearer " + token);

    Map<String, Object> body = new HashMap(null)<>();
    body.put("token", token);
    body.put("message", String.format("Hola %s, has iniciado sesion con exito!", username));
    body.put("username", username);
    response.getWriter().write(new ObjectMapper().writeValueAsString(body));
    response.setStatus(200);
    response.setContentType("application/json");
}
```

Vamos a implemetar la encryptacion con jwt

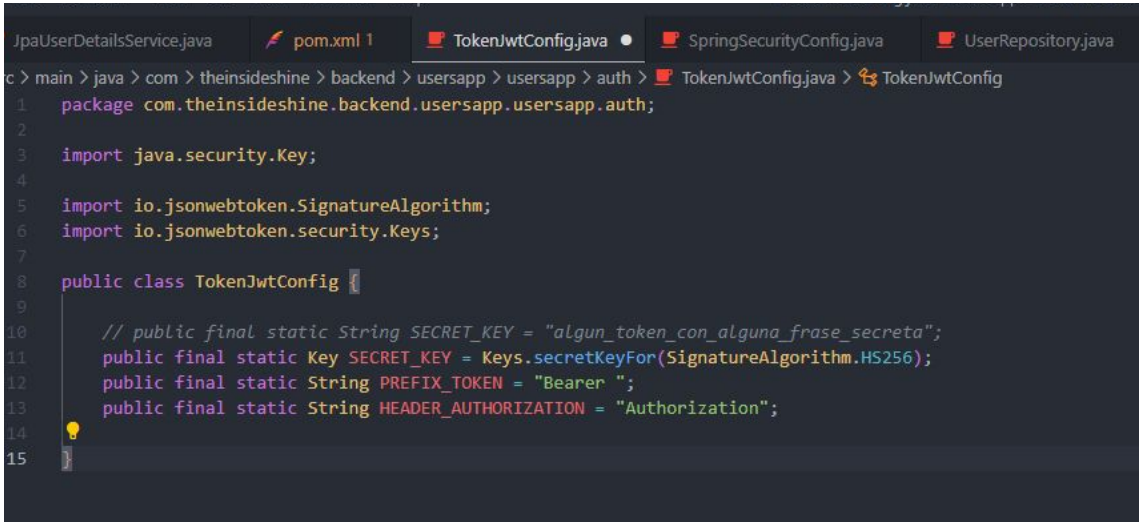
Token jwt

Agregar dependencia

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

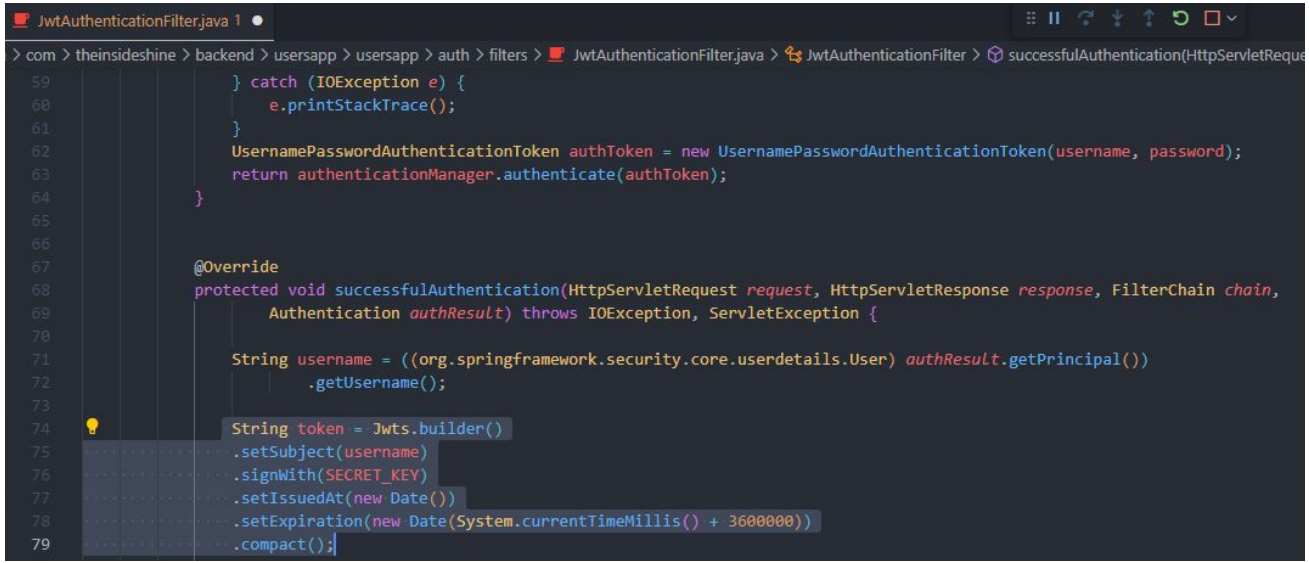
Token jwt

Vamos a cambiar la secret por la encriptación que propone jwt



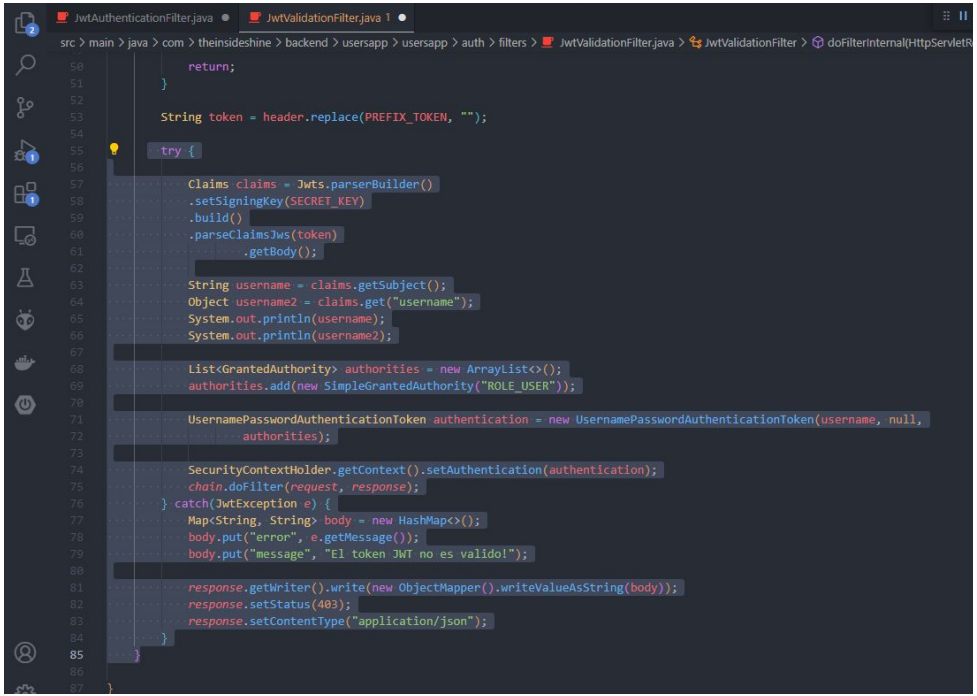
Token jwt

Vamos a cambiar la secret en JwTAuthenticationFilter vamos a cambiar el metodo con la generacion de jwy `protected void successfulAuthentication`



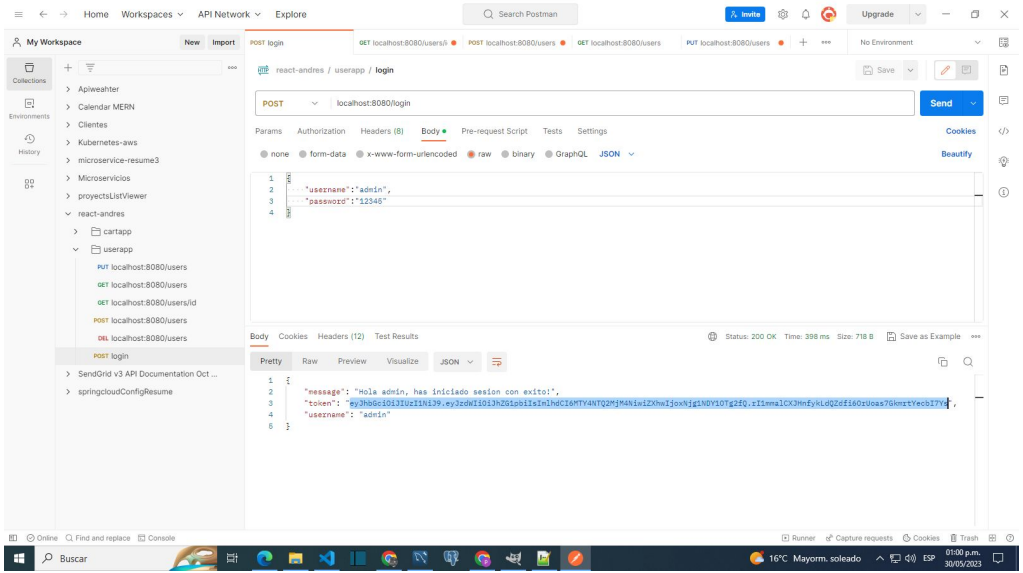
Token jwt

Vamos a cambiar la secret en JwtValidartion Filter vamos a cambiar el metodo con la generacion de jwy `protected void doFilterInternal`



Token jwt

Ahora deberíamos tener en el token el Jwt



nota:Cada vez que se reinicia spring genera una secret key

Crear entity role

Role.java

Obtener token en /login

Acceso a recursos protegidos

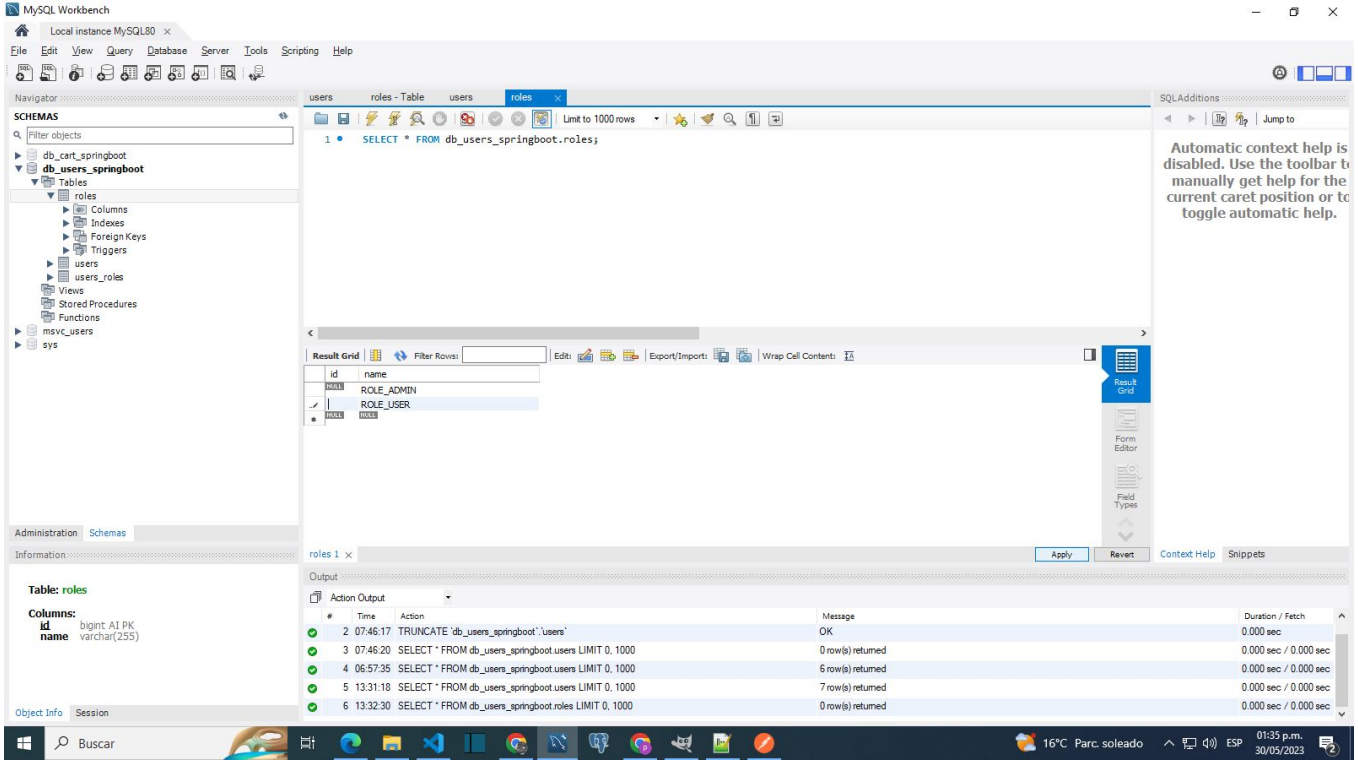
JwtValidationFilter

Crear Role.java

```java
@Entity
@Table(name = "roles")
public class Role {

    public Role() {
    }

    public Role(String name) {
        this.name = name;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String name;


    @ManyToMany
    @JoinTable(
        name = "users_roles",
        joinColumns = @JoinColumn(name="user_id"),
        inverseJoinColumns = @JoinColumn(name="role_id"),
        uniqueConstraints = { @UniqueConstraint(columnNames = {"user_id", "role_id"})})
    private List<Role> roles;
```

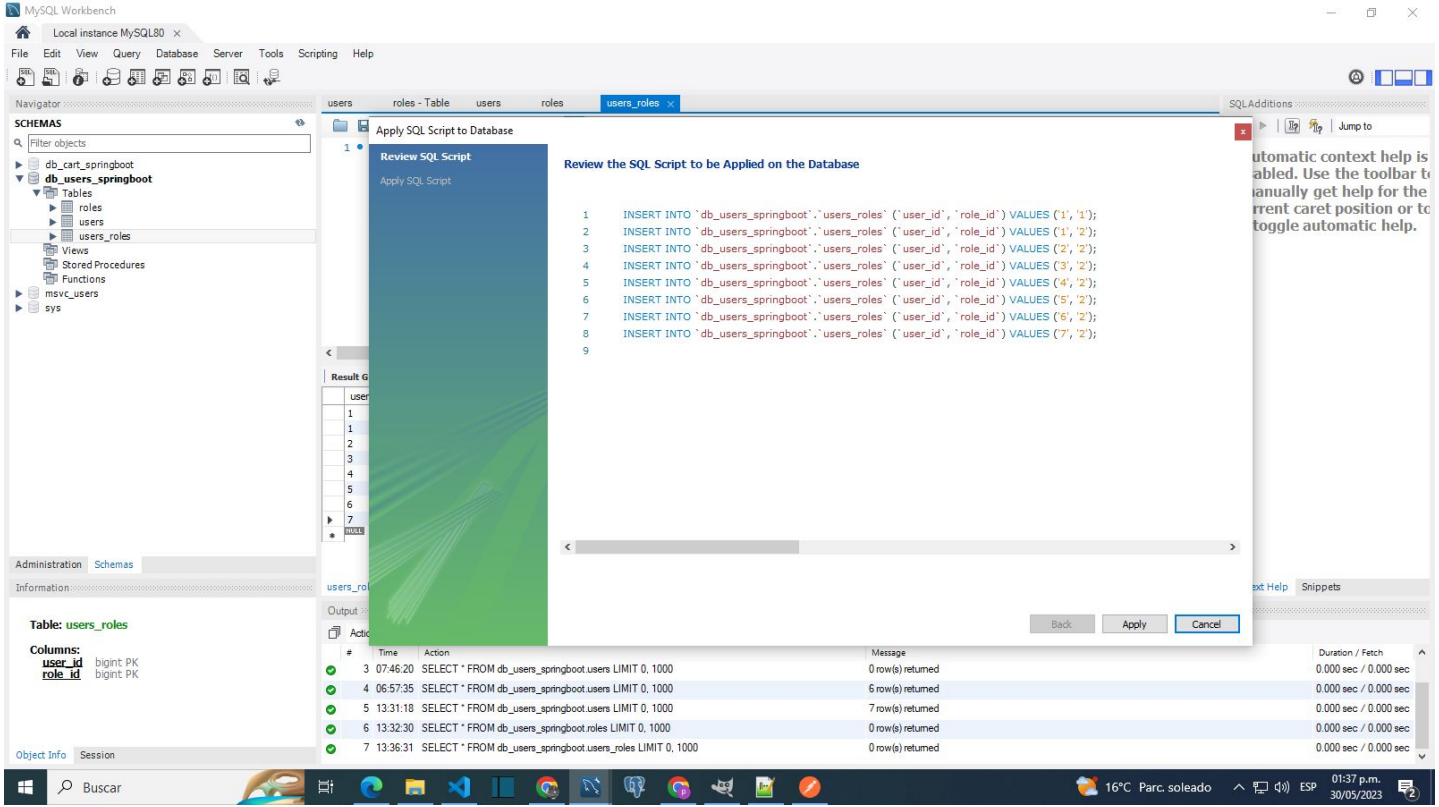
Crear Role.java

Al levantar el proyecto crea la tabla role, y la talba intermedia, a mano crearmos los roles y llenamos la tabla intermedia



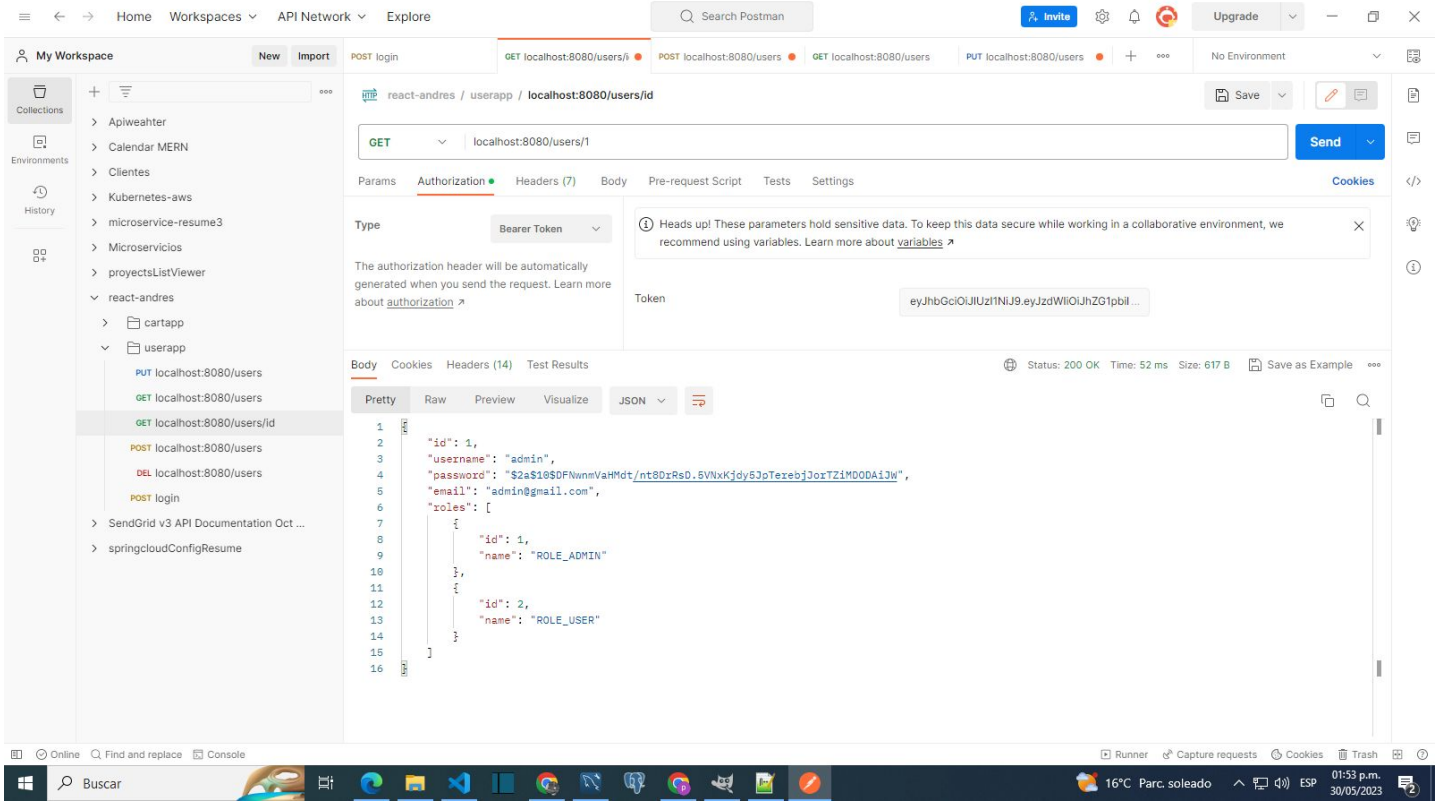
Crear Role.java

con apply se aplican los cambios editados



Crear Role.java

desde postman tiene que traer los usuarios con sus roles



Crear Role.java

agregar role en el método post

m.xml 1 | Role.java 4 | UserServiceImpl.java | RoleRepository.java | UserRepo

main > java > com > theinsideshine > backend > usersapp > usersapp > services > UserServiceImpl.java

```java
@Transactional(readOnly = true)
public Optional<User> findById(Long id) {
    return repository.findById(id);
}

@Override
@Transactional
public User save(User user) {
    user.setPassword(passwordEncoder.encode(user.getPassword()));

    Optional<Role> o = roleRepository.findByName(name:"ROLE_USER");

    List<Role> roles = new ArrayList<>();
    if (o.isPresent()) {
        roles.add(o.orElseThrow());
    }
    user.setRoles(roles);

    return repository.save(user);
}
```

RoleRepository.java - users-app - Visual Studio Code

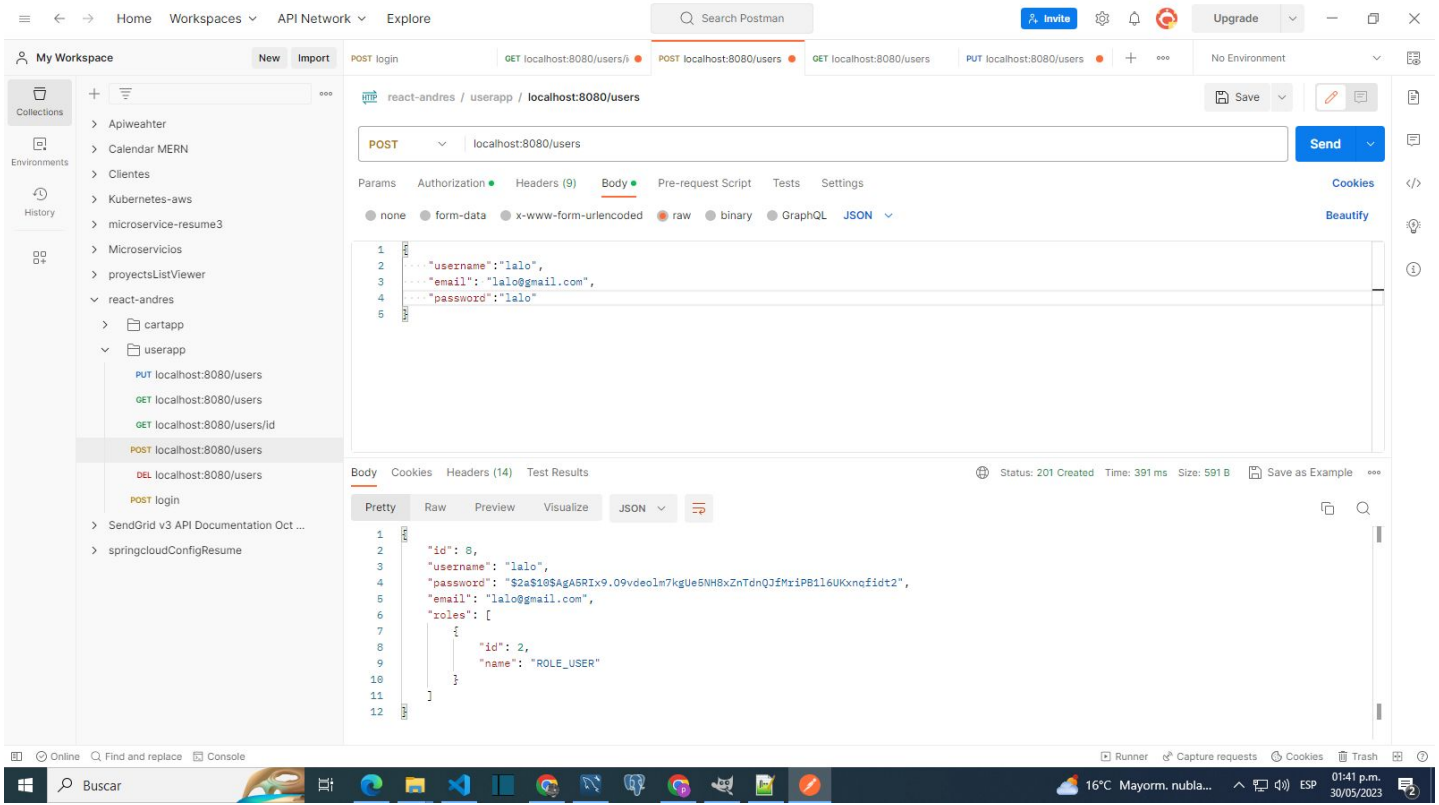
pom.xml 1 | Role.java 4 | UserServiceImpl.java | RoleRepository.java | User

main > java > com > theinsideshine > backend > usersapp > usersapp > repositories > RoleRepositor

```java
package com.theinsideshine.backend.usersapp.usersapp.repositories;

import java.util.Optional;

import org.springframework.data.repository.CrudRepository;

import com.theinsideshine.backend.usersapp.usersapp.models.entities.Role;

public interface RoleRepository
        extends CrudRepository<Role, Long> {
        Optional<Role> findByName(String name);
}
```

Crear Role.java

debe agregar role user al crear user



Crear Role.java

al autenticar ahora vamos a pasar los roles de la BBDD en `loadUserByUsername`

JpaUserDetailsService.java

src > main > java > com > theinsideshine > backend > usersapp > usersapp > services > JpaUserDetailsService.java > Language Support for Java(TM) by Red Hat >

```java
@Override
@Transactional(readOnly = true)
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    Optional<com.theinsideshine.backend.usersapp.usersapp.models.entities.User> o = repository
            .getUserByUsername(username);

    if (!o.isPresent()) {
        throw new UsernameNotFoundException(String.format("Username %s no existe en el sistema!", username));
    }
    com.theinsideshine.backend.usersapp.usersapp.models.entities.User user = o.orElseThrow();

    List<GrantedAuthority> authorities = user.getRoles()
            .stream()
            .map(r -> new SimpleGrantedAuthority(r.getName()))
            .collect(Collectors.toList());

    return new User(
            user.getUsername(),
            user.getPassword(),
            true,
            true,
            true,
            true,
            authorities);

}
```

Crear Role.java

vamos a agrega en el token los roles en `successfulAuthentication`
en JwtAuthenticationFilter

Crear Role.java

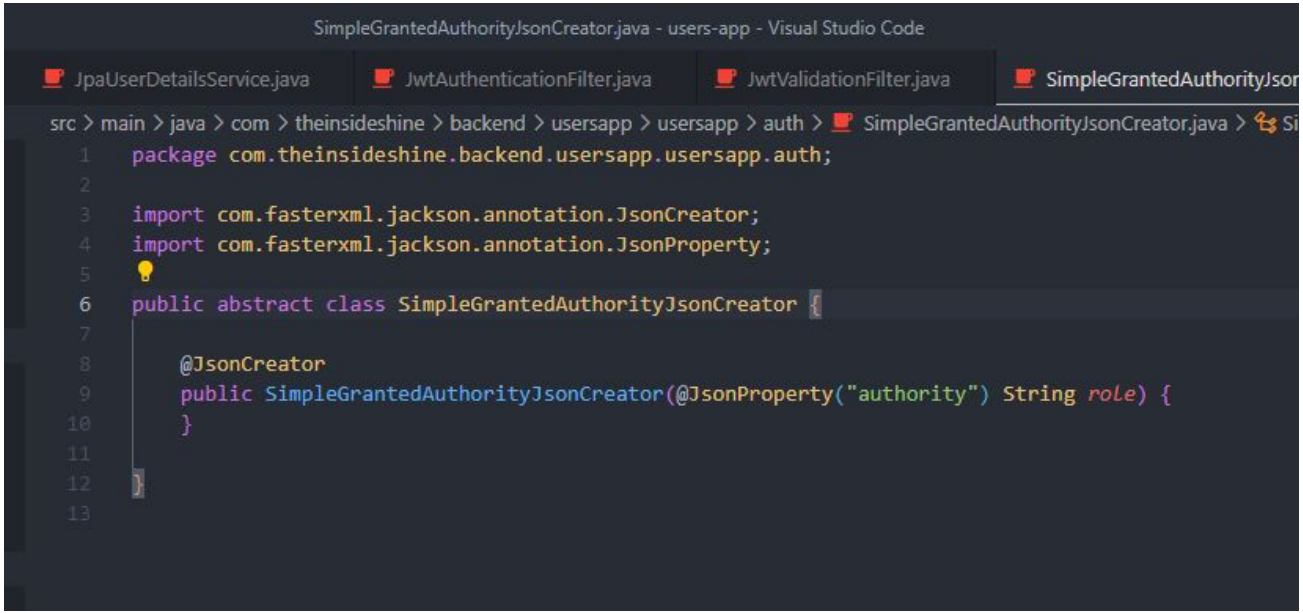
vamos a agrega en el token los roles en `void doFilterInternal`
en JwtValidationFilter

```
61          Object authoritiesClaims = claims.get("authorities");
62          String username = claims.getSubject();
63          Object username2 = claims.get("username");
64          System.out.println(username);
65          System.out.println(username2);
66
67          Collection<? extends GrantedAuthority> authorities = Arrays
68                  .asList(new ObjectMapper()
69                          .addMixIn(target:SimpleGrantedAuthority.class, mixinSource:SimpleGrantedAuthorityJsonCreator.class)
70                          .readValue(authoritiesClaims.toString().getBytes(), valueType:SimpleGrantedAuthority[].class));
71
72          UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(username, null,
73                  authorities);
74
```

tenemos que crear la clase `SimpleGrantedAuthorityJsonCreator`

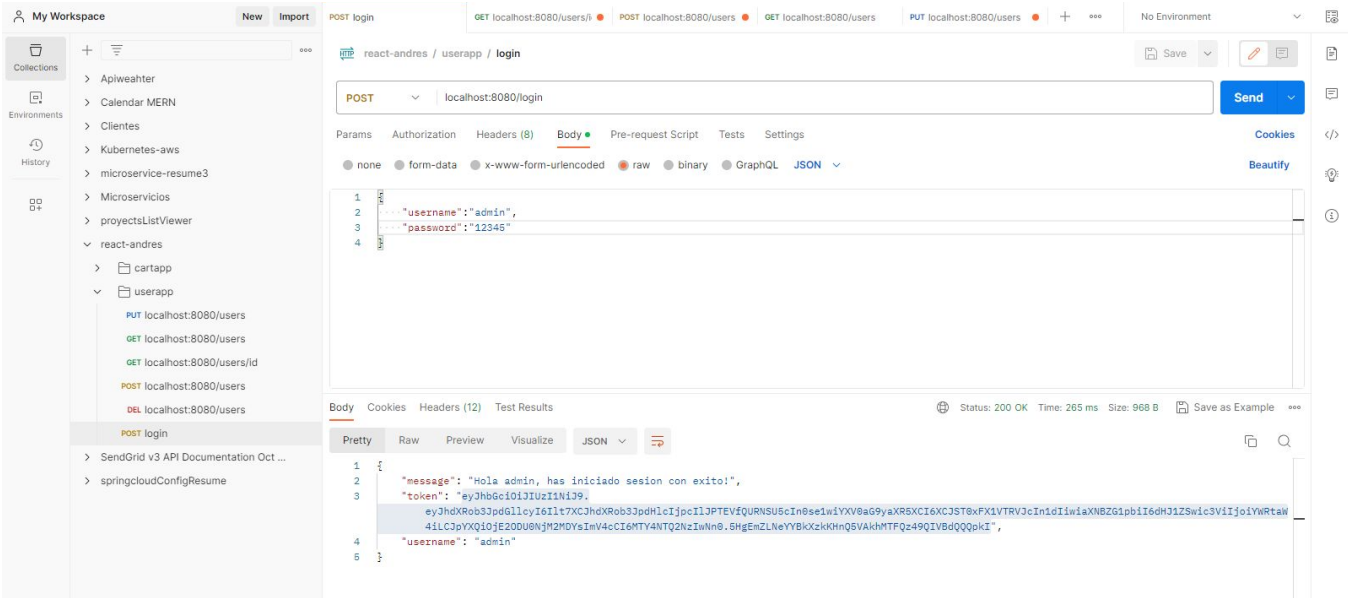
Crear Role.java

tenemos que crear la clase `SimpleGrantedAuthorityJsonCreator`



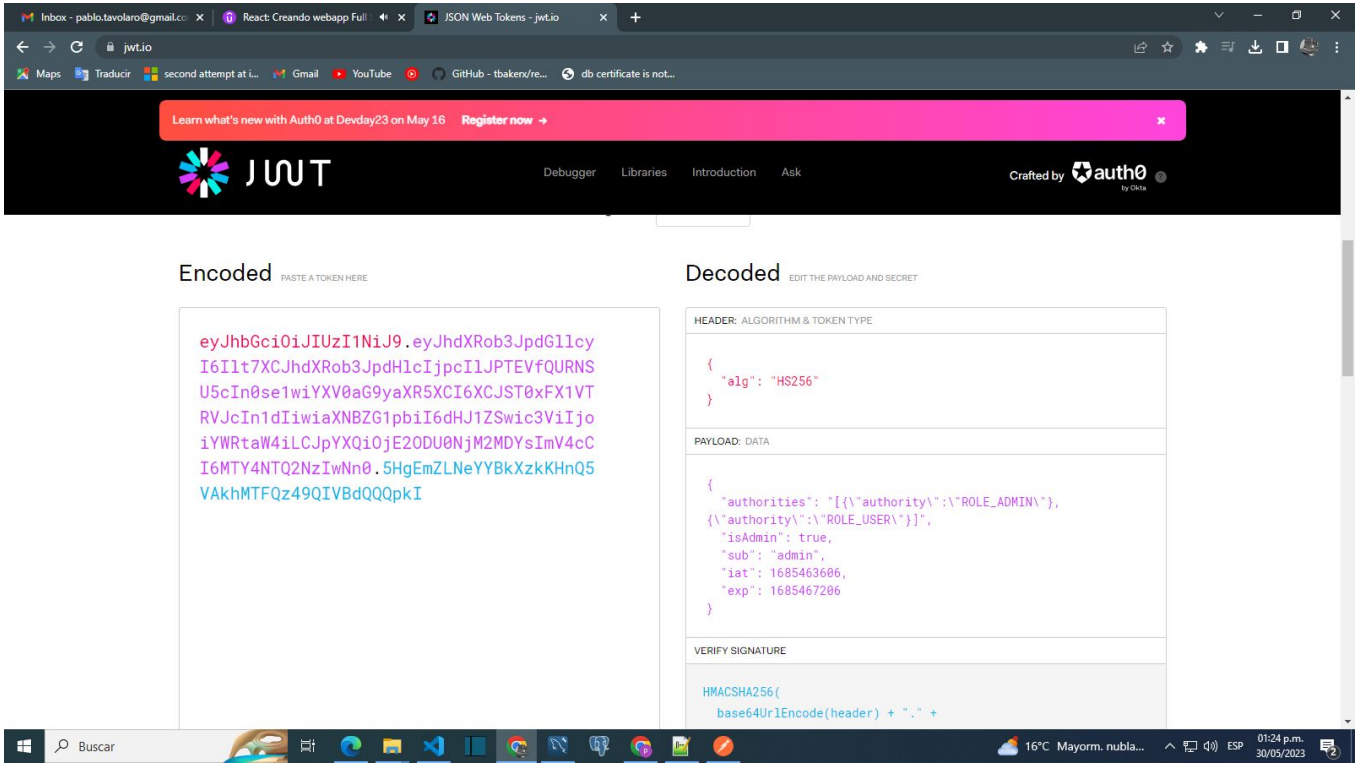
Crear Role.java

ahora debería devolver un token mas largo



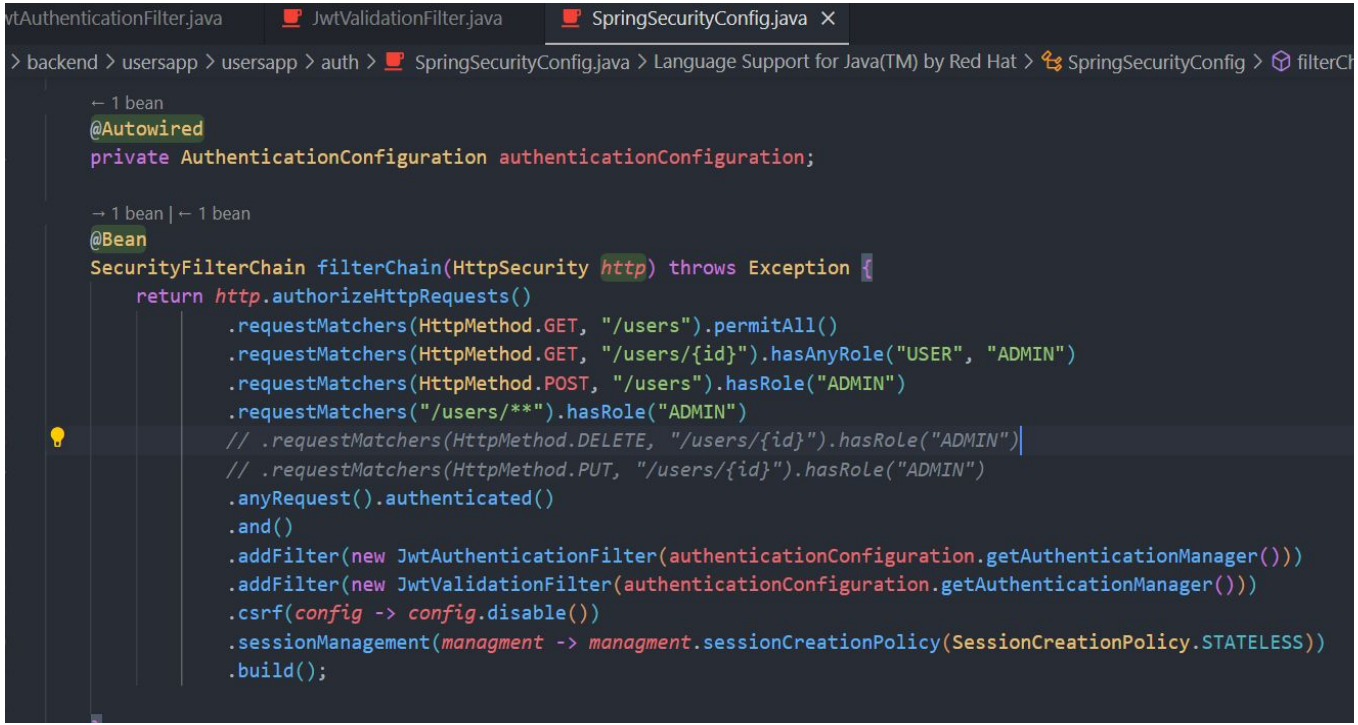
Crear Role.java

si vemos el token en jwt.io , veremos los roles en el



Crear Role.java

vamos agregar a las rutas el manejo de roles en SpringSecurityConfig



levantando el proyecto ya esta habilitado solo el post put y delete para el role admin

Detalles

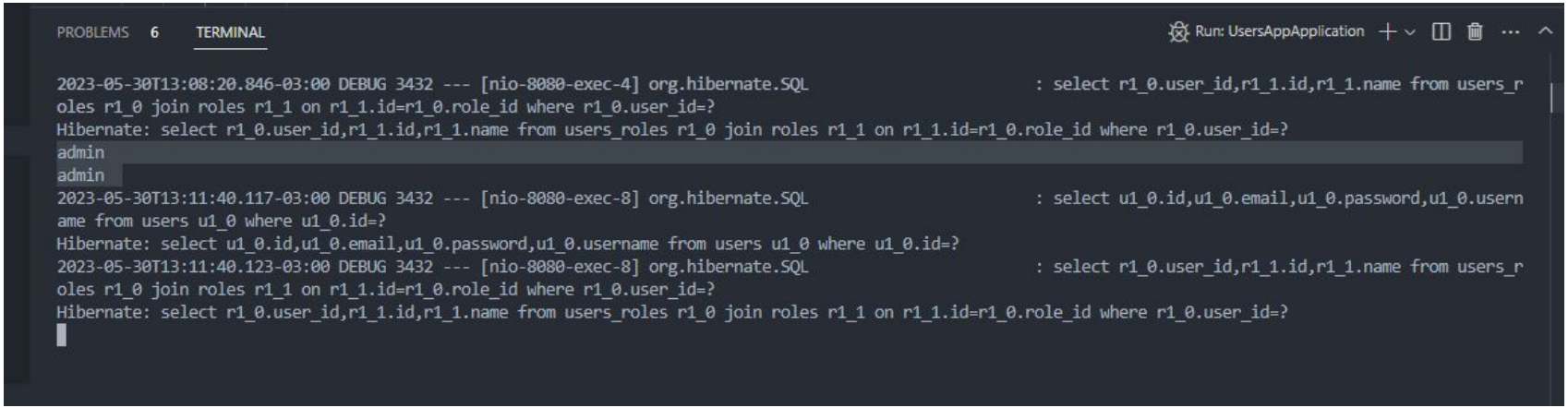
Si queremos agregar información al token lo hacemos JwtAuthenticactionFilter aca agregamos el username

Detalles

En la validacion toma el user del subject, ahora tambien le agregamos el campo username e imprimimos los dos ,

Detalles

al hacer un petición por postman que requiera validacion imprimie los dos username



Detalles

Se vemos el token devuelto vemos la info de user name

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJhdXRob3JpdGllcy
I6Ilt7XCJhdXRob3JpdHlcIjpcIlJPTEVfQURNS
U5cIn0se1wiYXV0aG9yaXR5XCI6XCJST0xFX1VT
RVJcIn1dIiwiaXNBZG1pbiI6dHJ1ZSwidXNlcm5
hbWUiOiJhZG1pbiIsInN1YiI6ImFkbWluIiwiaW
F0IjoxNjg1NDYyOTAwLCJleHAiOjE2ODU0NjY1M
DB9.AMPQGeJv9kGumeA1A7GKcjZ1NehQLZl-
eVmELQAhNOY
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

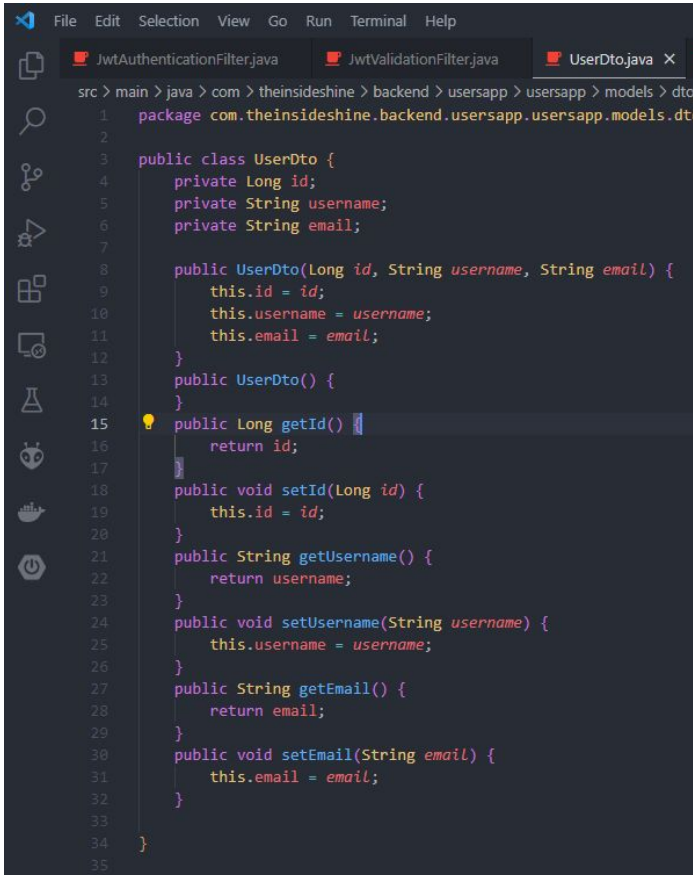
```
{
  "alg": "HS256"
}
```

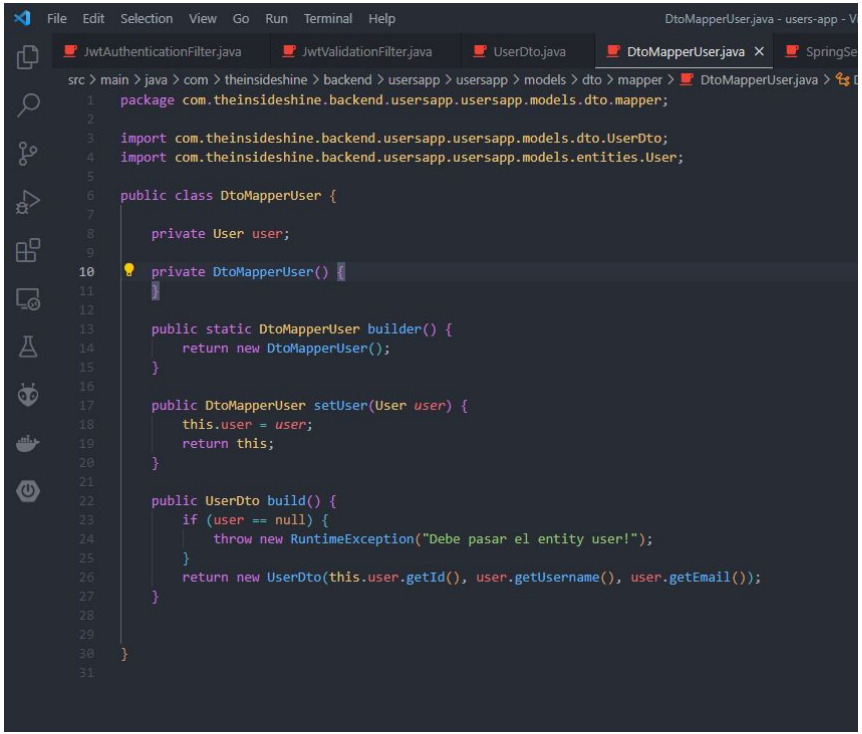
PAYLOAD: DATA

```
{
  "authorities": "[{\"authority\":\"ROLE_ADMIN\"},
{\"authority\":\"ROLE_USER\"}]",
  "isAdmin": true,
  "username": "admin",
  "sub": "admin",
  "iat": 1685462900,
  "exp": 1685466500
}
```

Dto

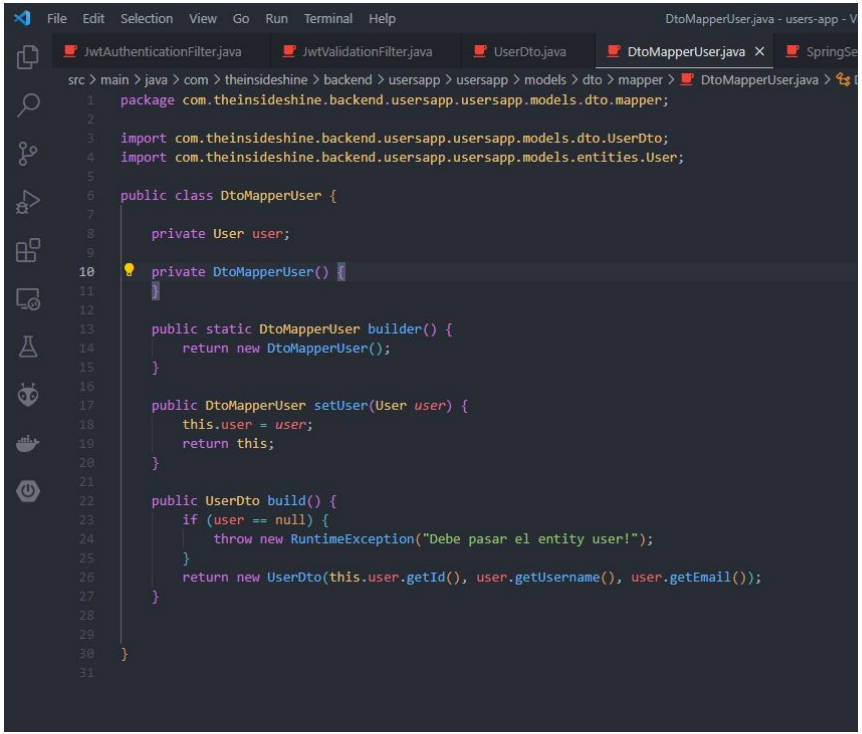
Crear Dto y dtoMapper





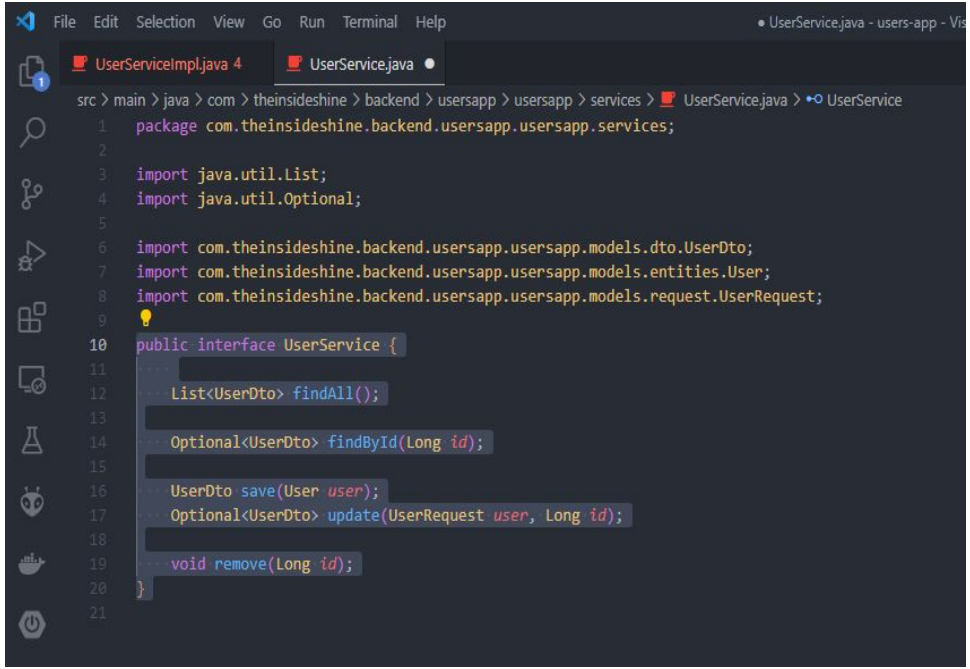
Dto

dtoMapper . la idea es contruir un mapper con el patron de diseño bulider https://medium.com/@Seonggil/how-to-use-builders-efficiently-when-mapping-dto-entity-530bf8d71ed2, principalmente se usa para modificar solo el Dto y que en el mappeo solo agregamos el campo en el return



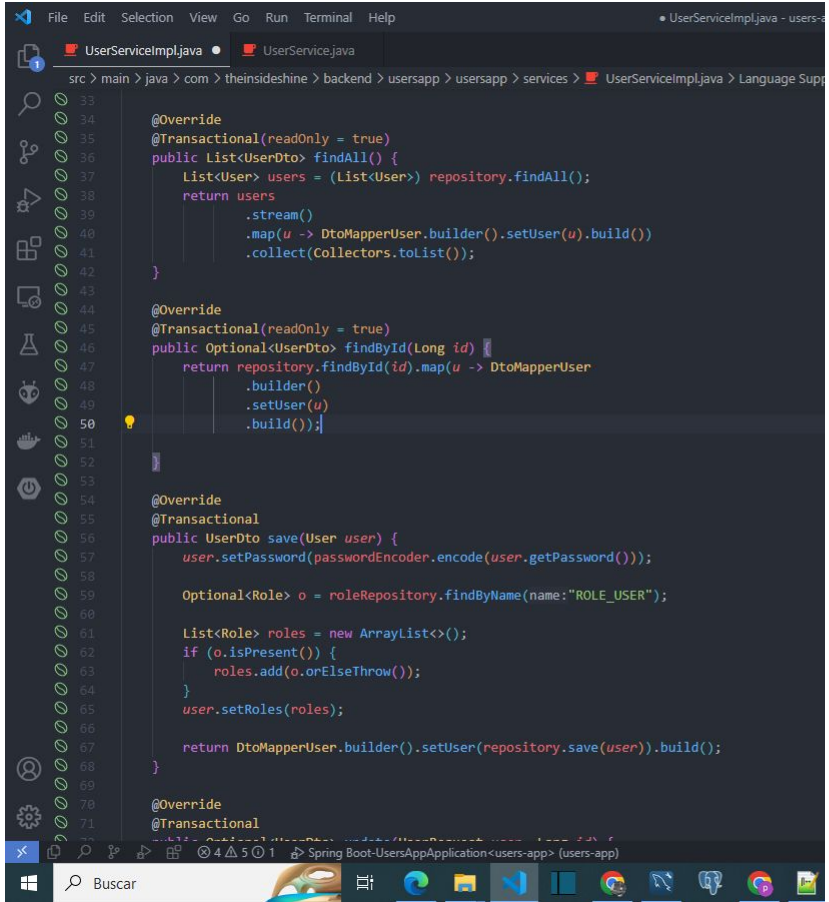
Dto

modificando los services



Dto

modificando los services



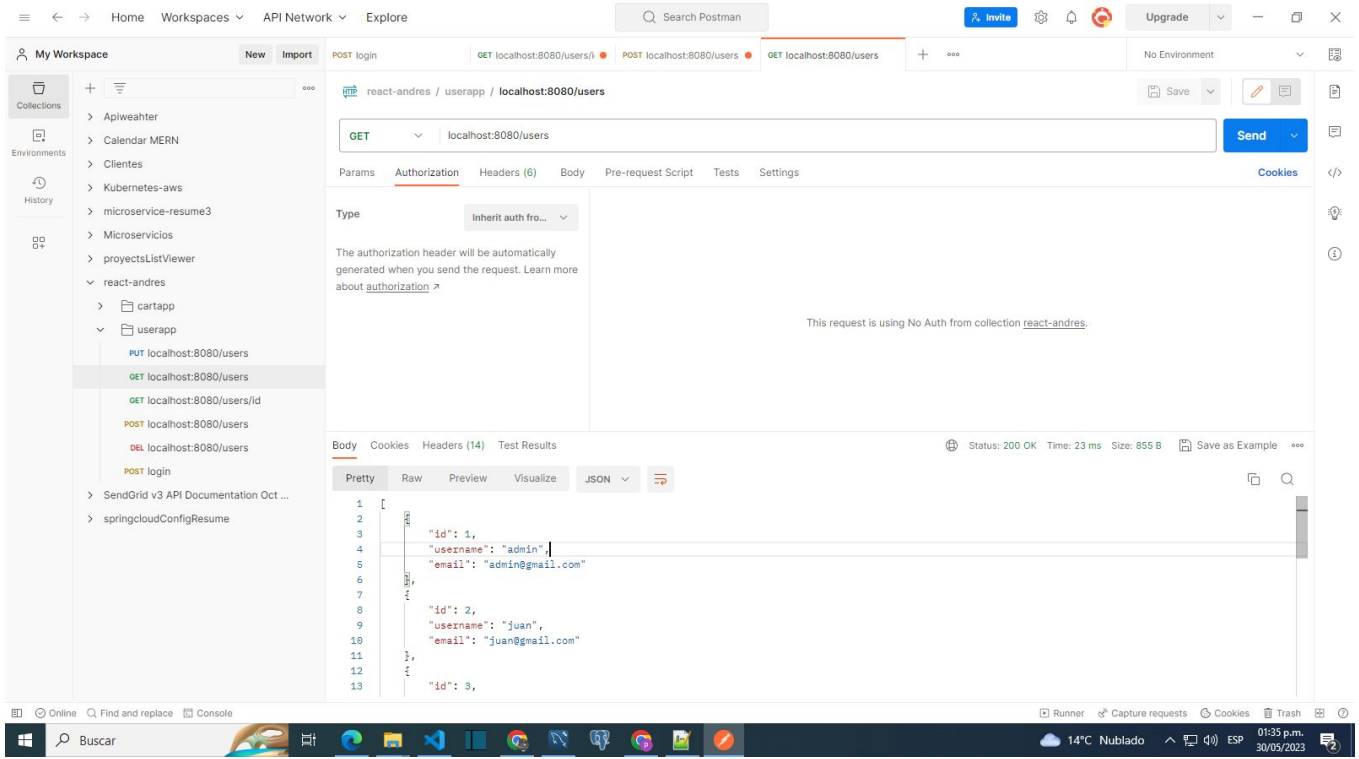
Dto

modificando los services

```java
@Override
@Transactional
public Optional<UserDto> update(UserRequest user, Long id) {
    Optional<User> o = repository.findById(id);
    User userOptional = null;
    if (o.isPresent()) {
        User userDb = o.orElseThrow();
        userDb.setUsername(user.getUsername());
        userDb.setEmail(user.getEmail());
        userOptional = repository.save(userDb);
    }
    return Optional.ofNullable(DtoMapperUser.builder().setUser(userOptional).build());
}
```

Dto

Ahora devuelvo solo el DTO



resumen

En el SpringSecurityConfig se ponen la rutas y sus permisos , se le deben pasar los filtros Jwt de autorización y validación , en el TokenJwtConfig se guardan las constantes

SprinSecurityConfig

JwtAuhtorizationFilter

JwtValidationFilter

TokenJwtConfig

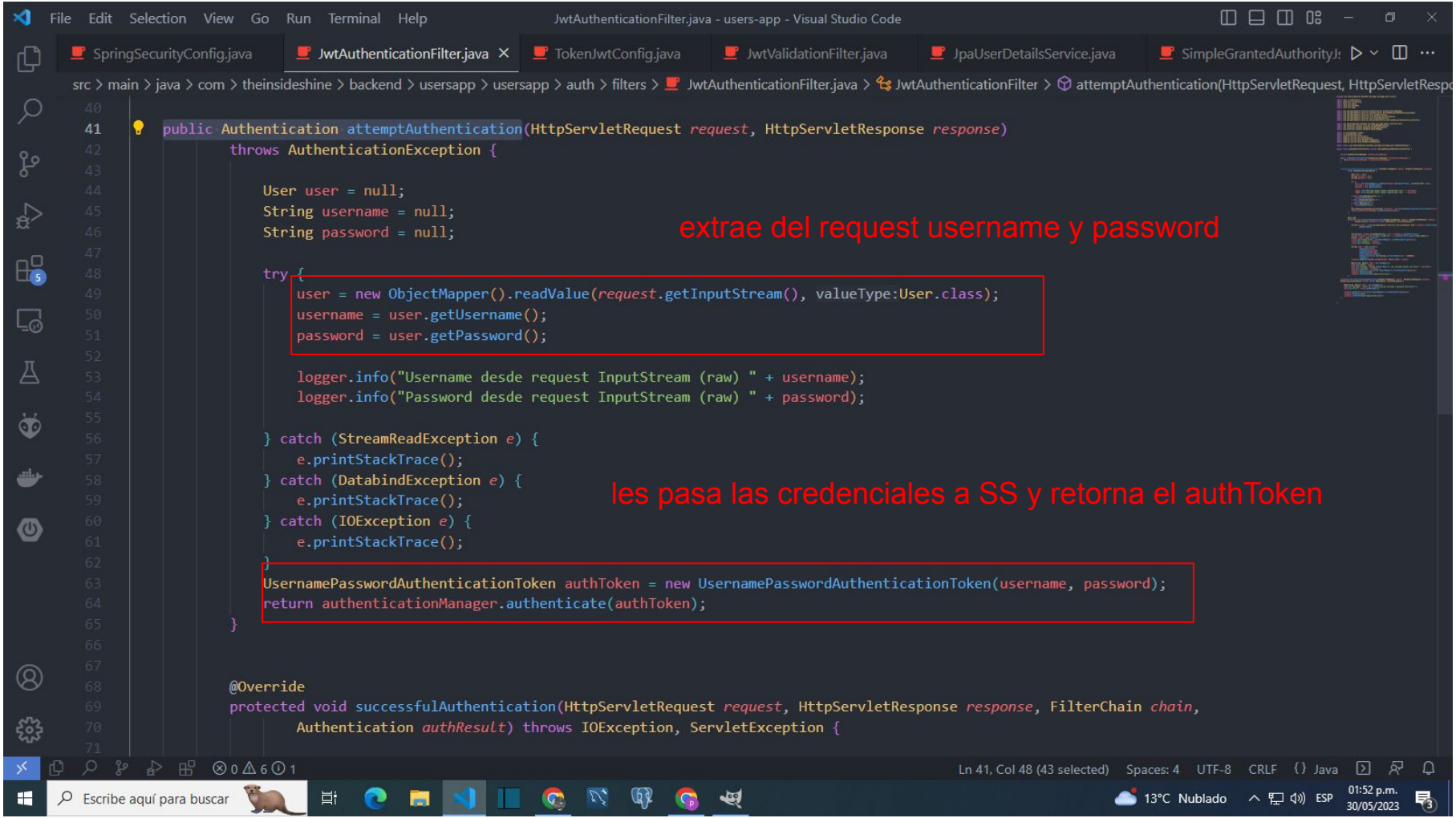
resumen

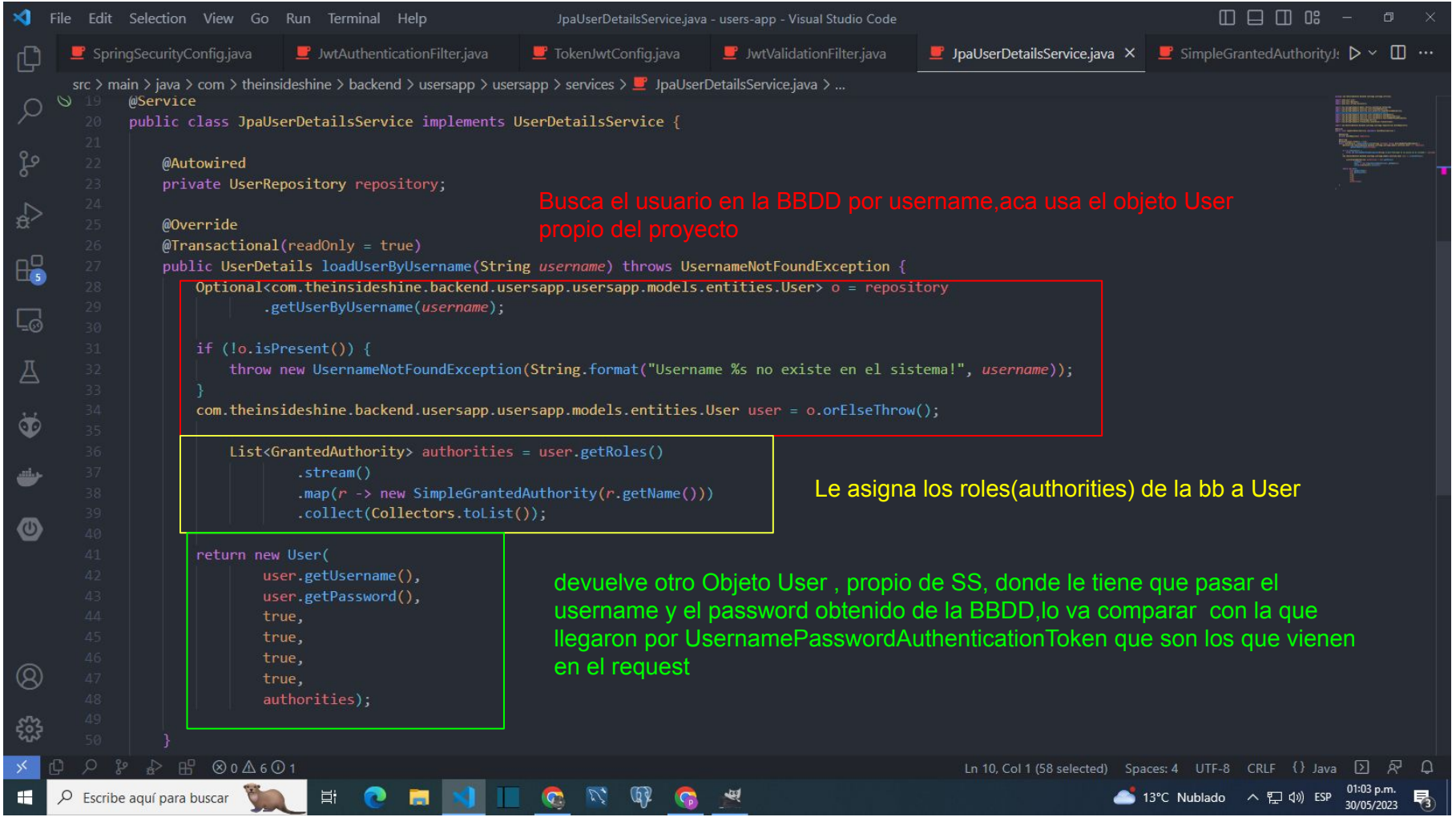
La clase `SimpleGrantedAuthorityJsonCreator` se crea para pasar los roles en la validación y la JpaUserDetailService realiza la contra la base de datos, implementado la interface UserDetailService

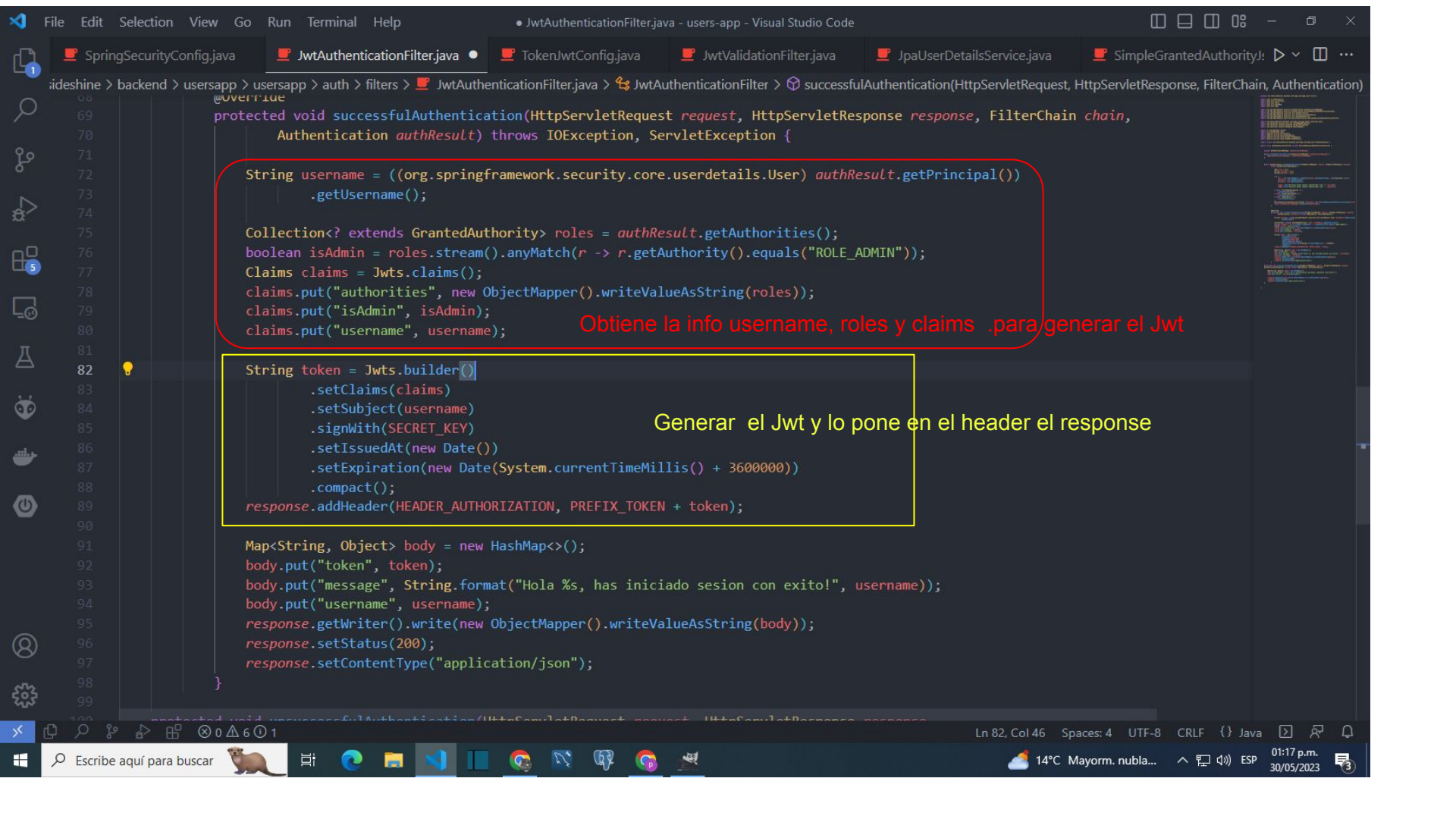
JwtValidationFilter

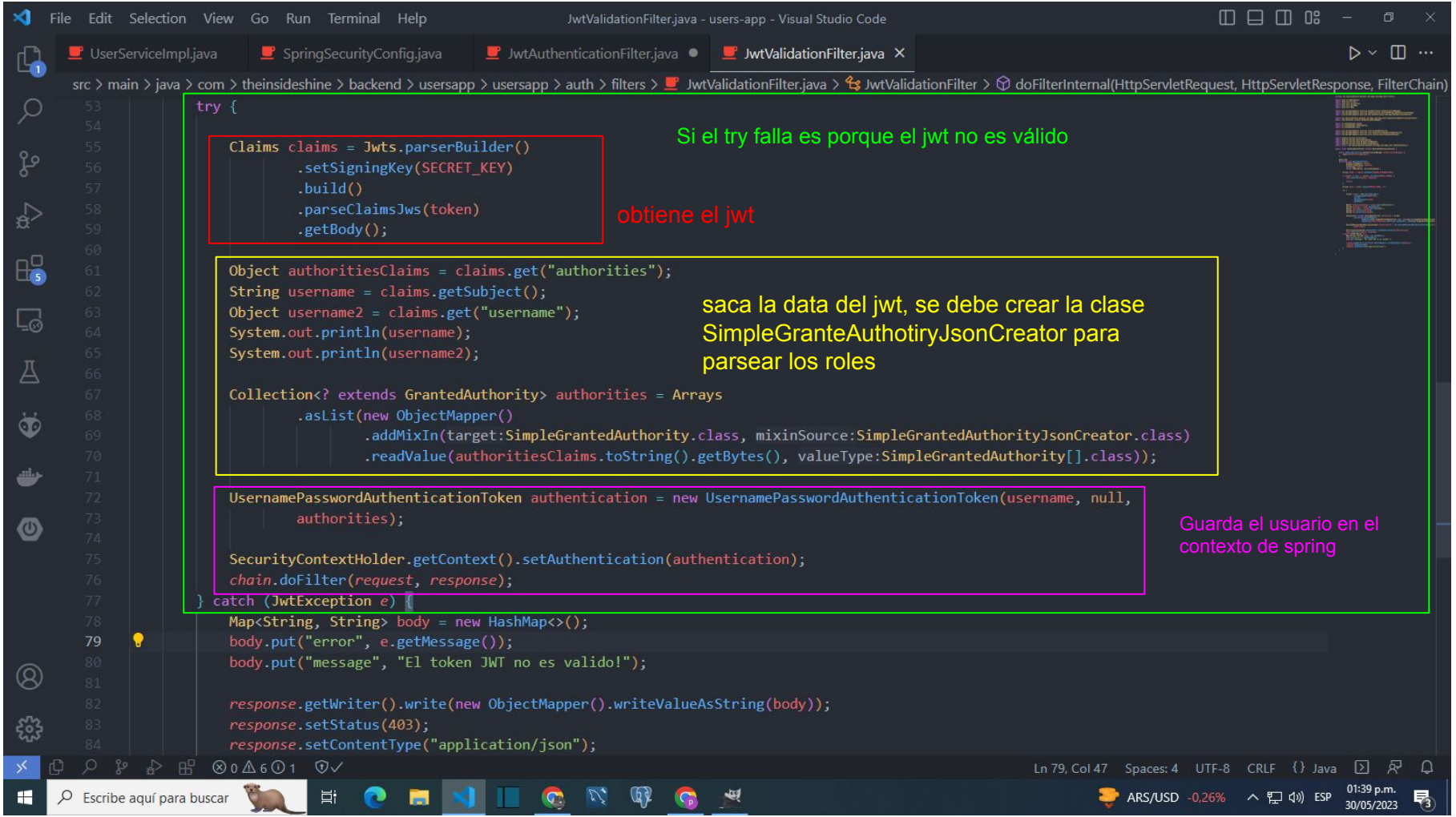
`SimpleGrantedAuthorityJsonCreator`

JpaUserDetailService


File Edit Selection View Go Run Terminal Help
JwtAuthenticationFilter.java - users-app - Visual Studio Code
SpringSecurityConfig.java
JwtAuthenticationFilter.java
TokenJwtConfig.java
JwtValidationFilter.java
JpaUserDetailsService.java
SimpleGrantedAuthorityJ
src > main > java > com > theinsideshine > backend > usersapp > usersapp > auth > filters > JwtAuthenticationFilter.java > JwtAuthenticationFilter > attemptAuthentication(HttpServletRequest, HttpServletResp
41 public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
42 throws AuthenticationException {
44 User user = null;
45 String username = null;
46 String password = null;
48 try {
49 user = new ObjectMapper().readValue(request.getInputStream(), valueType:User.class);
50 username = user.getUsername();
51 password = user.getPassword();
53 logger.info("Username desde request InputStream (raw) " + username);
54 logger.info("Password desde request InputStream (raw) " + password);
56 } catch (StreamReadException e) {
57 e.printStackTrace();
58 } catch (DatabindException e) {
59 e.printStackTrace();
60 } catch (IOException e) {
61 e.printStackTrace();
63 UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(username, password);
64 return authenticationManager.authenticate(authToken);
65 }
68 @Override
69 protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
70 Authentication authResult) throws IOException, ServletException {
extrae del request username y password
les pasa las credenciales a SS y retorna el authToken
Ln 41, Col 48 (43 selected) Spaces: 4 UTF-8 CRLF Java
Escribe aquí para buscar
13°C Nublado
ESP
01:52 p.m.
30/05/2023


File Edit Selection View Go Run Terminal Help
JpaUserDetailsService.java - users-app - Visual Studio Code
SpringSecurityConfig.java
JwtAuthenticationFilter.java
TokenJwtConfig.java
JwtValidationFilter.java
JpaUserDetailsService.java
SimpleGrantedAuthorityJ
src > main > java > com > theinsideshine > backend > usersapp > usersapp > services > JpaUserDetailsService.java > ...
@Service
public class JpaUserDetailsService implements UserDetailsService {
@Autowired
private UserRepository repository;
@Override
@Transactional(readOnly = true)
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
Optional<com.theinsideshine.backend.usersapp.usersapp.models.entities.User> o = repository
.getUserByUsername(username);
if (!o.isPresent()) {
throw new UsernameNotFoundException(String.format("Username %s no existe en el sistema!", username));
}
com.theinsideshine.backend.usersapp.usersapp.models.entities.User user = o.orElseThrow();
List<GrantedAuthority> authorities = user.getRoles()
.stream()
.map(r -> new SimpleGrantedAuthority(r.getName()))
.collect(Collectors.toList());
return new User(
user.getUsername(),
user.getPassword(),
true,
true,
true,
true,
authorities);
}
Busca el usuario en la BBDD por username,aca usa el objeto User propio del proyecto
Le asigna los roles(authorities) de la bb a User
devuelve otro Objeto User , propio de SS, donde le tiene que pasar el username y el password obtenido de la BBDD,lo va comparar con la que llegaron por UsernamePasswordAuthenticationToken que son los que vienen en el request
Ln 10, Col 1 (58 selected) Spaces: 4 UTF-8 CRLF Java
Escribe aquí para buscar
13°C Nublado
ESP
01:03 p.m.
30/05/2023


File Edit Selection View Go Run Terminal Help
• JwtAuthenticationFilter.java - users-app - Visual Studio Code
SpringSecurityConfig.java
JwtAuthenticationFilter.java
TokenJwtConfig.java
JwtValidationFilter.java
JpaUserDetailsService.java
SimpleGrantedAuthorityJ
sideshine > backend > usersapp > usersapp > auth > filters > JwtAuthenticationFilter.java > JwtAuthenticationFilter > successfulAuthentication(HttpServletRequest, HttpServletResponse, FilterChain, Authentication)
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
Authentication authResult) throws IOException, ServletException {
String username = ((org.springframework.security.core.userdetails.User) authResult.getPrincipal())
.getUsername();
Collection<? extends GrantedAuthority> roles = authResult.getAuthorities();
boolean isAdmin = roles.stream().anyMatch(r -> r.getAuthority().equals("ROLE_ADMIN"));
Claims claims = Jwts.claims();
claims.put("authorities", new ObjectMapper().writeValueAsString(roles));
claims.put("isAdmin", isAdmin);
claims.put("username", username);
Obtiene la info username, roles y claims .para generar el Jwt
String token = Jwts.builder()
.setClaims(claims)
.setSubject(username)
.signWith(SECRET_KEY)
.setIssuedAt(new Date())
.setExpiration(new Date(System.currentTimeMillis() + 3600000))
.compact();
response.addHeader(HEADER_AUTHORIZATION, PREFIX_TOKEN + token);
Generar el Jwt y lo pone en el header el response
Map<String, Object> body = new HashMap<>();
body.put("token", token);
body.put("message", String.format("Hola %s, has iniciado sesion con exito!", username));
body.put("username", username);
response.getWriter().write(new ObjectMapper().writeValueAsString(body));
response.setStatus(200);
response.setContentType("application/json");
}
Ln 82, Col 46 Spaces: 4 UTF-8 CRLF Java
Escribe aquí para buscar
14°C Mayorm. nubla...
ESP
01:17 p.m.
30/05/2023


File Edit Selection View Go Run Terminal Help
JwtValidationFilter.java - users-app - Visual Studio Code
UserServiceImpl.java
SpringSecurityConfig.java
JwtAuthenticationFilter.java
JwtValidationFilter.java
src > main > java > com > theinsideshine > backend > usersapp > usersapp > auth > filters > JwtValidationFilter.java > JwtValidationFilter > doFilterInternal(HttpServletRequest, HttpServletResponse, FilterChain)
try {
Claims claims = Jwts.parserBuilder()
.setSigningKey(SECRET_KEY)
.build()
.parseClaimsJws(token)
.getBody();
Si el try falla es porque el jwt no es válido
obtiene el jwt
Object authoritiesClaims = claims.get("authorities");
String username = claims.getSubject();
Object username2 = claims.get("username");
System.out.println(username);
System.out.println(username2);
saca la data del jwt, se debe crear la clase SimpleGranteAuthotiryJsonCreator para parsear los roles
Collection<? extends GrantedAuthority> authorities = Arrays
.asList(new ObjectMapper()
.addMixIn(target:SimpleGrantedAuthority.class, mixinSource:SimpleGrantedAuthorityJsonCreator.class)
.readValue(authoritiesClaims.toString().getBytes(), valueType:SimpleGrantedAuthority[].class));
UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(username, null,
authorities);
SecurityContextHolder.getContext().setAuthentication(authentication);
chain.doFilter(request, response);
Guarda el usuario en el contexto de spring
} catch (JwtException e) {
Map<String, String> body = new HashMap<>();
body.put("error", e.getMessage());
body.put("message", "El token JWT no es valido!");
response.getWriter().write(new ObjectMapper().writeValueAsString(body));
response.setStatus(403);
response.setContentType("application/json");
Ln 79, Col 47 Spaces: 4 UTF-8 CRLF Java
Escribe aquí para buscar
ARS/USD -0,26%
ESP
01:39 p.m.
30/05/2023