

Protección de rutas

Obtener token en /login

Acceso a recursos protegidos

FilterChain

JwtAuthenticationFilter

JwtValidationFilter

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(management -> management.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .build();
    }
}
```

Con versiones superiores de springboot 3.0.4 el método `authorizeHttpRequests()` esta deprecado

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

```
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 UserDetailsService

```
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

```
@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<>();
    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");
}
}
```

aquí se pueden ver los métodos 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(management -> management.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .build();
    }
}
```

Obtener Token

Para poder realizar la autenticación 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.

```
@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

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** localhost:8080/login
- Body:** JSON (selected)
 - username: "admin"
 - password: "12345"
- Response:** 200 OK (Pretty) JSON output:

```
1   {  
2     "message": "Hola admin, has iniciado sesion con exito!",  
3     "token": "YkxndW5fdG9zZW5fY29uX2FsZ3VuYV9mcmFzZV9zZWNyZXRhLmFkbWlu",  
4     "username": "admin"  
5   }
```

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 encripar 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

```
DefaultSecurityFilterChain | ← 1 bean
Configuration
public class SpringSecurityConfig {

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

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

    ← 1 bean
    @Autowired
    private AuthenticationConfiguration authenticationConfiguration;

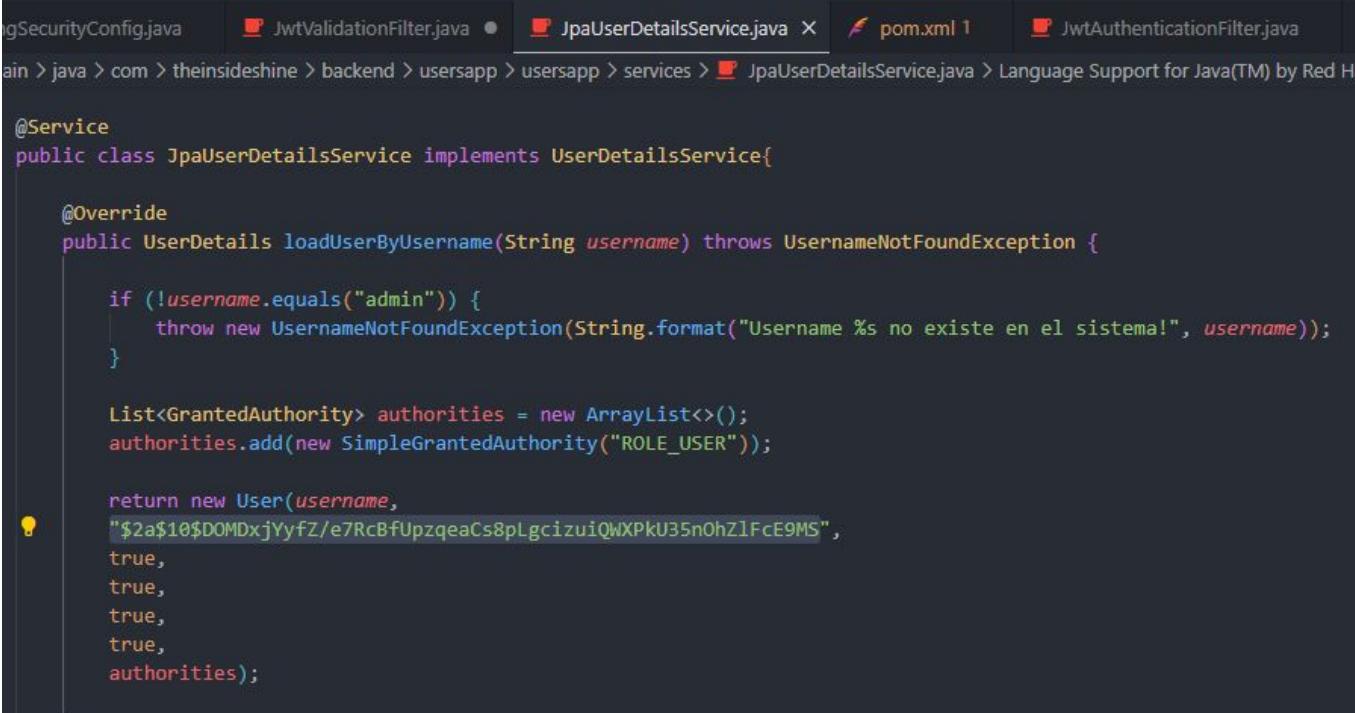
    → 1 bean | ← 1 bean
    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        return http.authorizeHttpRequests()
            .requestMatchers(HttpServletRequest.GET, "/users").permitAll()
            .anyRequest().authenticated()
            .and()
            .csrf(config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

    }
}
```

Obtener Token

Debemos cambiar el password esperado, ya que ahora está encriptado

Obtener Token



```
ngSecurityConfig.java  JwtValidationFilter.java  JpaUserDetailsService.java  pom.xml  JwtAuthenticationFilter.java
ain > java > com > theinsidesshine > backend > usersapp > usersapp > services > JpaUserDetailsService.java > Language Support for Java(TM) by Red Hat

@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,
                "$2a$10$DOMDxjYyfZ/e7RcBfUpzqeaCs8pLgcizuiQWXPKU35nOhZlFcE9MS",
                true,
                true,
                true,
                true,
                authorities);
    }
}
```

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

[Obtener Token](#)

The screenshot shows the Postman interface for a POST request to `localhost:8080/login`. The request body is set to `JSON` and contains the following data:

```
1 ... "username": "admin",
2 ... "password": "12345"
```

The response status is `200 OK` with a response time of `21 ms` and a size of `568 B`. The response body is:

```
1 ...
2   "message": "Hola admin, has iniciado sesion con exito!",
3   "token": "YWxndW5fdG9rZW5fY29uX2FsZ3VuYV9mcmFzZV9zZWNyZXRhLmFkbWlu",
4   "username": "admin"
5 ...
```

Acceso a recursos protegidos



Acceso a recursos protegidos

Vamos a crear otro filtro JwtValidationFilter y el metodo doFilterInternal

Acceso a recursos protegidos

Hay que agregar el filtro a la cadena de rutas

```
validationFilter.java SpringSecurityConfig.java ● TokenJwtConfig.java JwtAuthenticationFilter.java
in > java > com > theinsideshine > backend > usersapp > usersapp > auth > SpringSecurityConfig.java > Language Support for Java(TM) by Red Hat > S
    AuthenticationManager authenticationManager() throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }

    @Autowired
    private AuthenticationConfiguration authenticationConfiguration;

    @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()))
            .addFilter(new JwtValidationFilter(authenticationConfiguration.getAuthenticationManager()))
            .sessionManagement(management -> management.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .build();
    }
}
```

Acceso a recursos protegidos

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

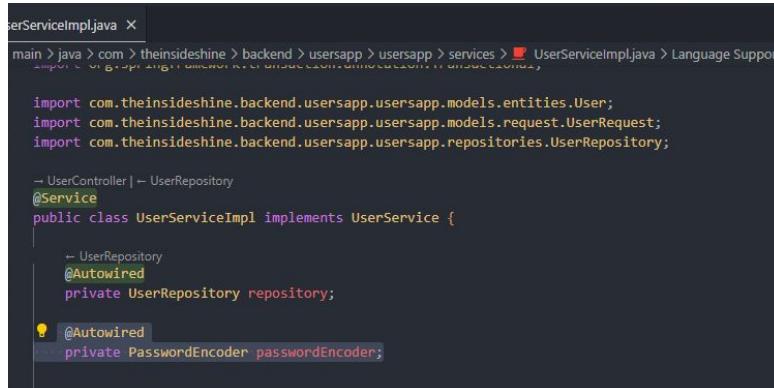
The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. In the center, a POST request to 'localhost:8080/users' is being made. The 'Authorization' tab is selected, showing a dropdown set to 'Bearer Token'. A tooltip explains that the authorization header will be automatically generated when sending the request. Below this, a 'Token' input field contains a long string of characters: 'YWxndW5fdG9rZW5fY29uX2FsZ3VuYV9m...'. At the bottom of the request panel, the status is shown as 'Status: 201 Created'. The 'Body' tab at the bottom displays the JSON response:

```
1  {
2   "id": 9,
3   "username": "josefina",
4   "password": "12345",
5   "email": "josefinas@gmail.com"
6 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray indicating the date and time.

Encriptar password

vamos a encriptar el password al guardar en la base datos



```
UserServiceImpl.java ×
main > java > com.theinsideshine > backend > usersapp > usersapp > services > UserServiceImpl.java > Language Support
import com.theinsideshine.backend.usersapp.usersapp.models.User;
import com.theinsideshine.backend.usersapp.usersapp.models.request.UserRequest;
import com.theinsideshine.backend.usersapp.usersapp.repositories.UserRepository;

→ UserController | ← UserRepository
@Service
public class UserServiceImpl implements UserService {
    ← UserRepository
    @Autowired
    private UserRepository repository;

    @Autowired
    private PasswordEncoder passwordEncoder;
```



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

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

Encriptar password

The screenshot shows the MySQL Workbench interface. In the Navigator pane, under the 'db_users_springboot' schema, the 'users' table is selected. A context menu is open over the table, with the 'Truncate Table...' option highlighted. The 'Output' pane at the bottom shows the following log entries:

#	Action	Message	Duration / Fetch
1	SELECT * FROM db_users_springboot.users LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
2	TRUNCATE `db_users_springboot`.`users`	OK	0.000 sec
3	SELECT * FROM db_users_springboot.users LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

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

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. The main workspace shows a POST request to `localhost:8080/users`. The 'Body' tab is selected, showing the following JSON payload:

```
1 {
2   "username": "admin",
3   "email": "admin@gmail.com",
4   "password": "12345"
```

The response pane shows the result of the POST request:

```
1 {
2   "id": 1,
3   "username": "admin",
4   "password": "$2a$10$DFNwmVaHMdt/nt8DzRsD.SVWxKjdySjzTerebjJorTZiMDODAiJW",
5   "email": "admin@gmail.com"
6 }
```

At the bottom, the status bar indicates the request was successful with a 201 Created status, took 139 ms, and was 555 B in size.

Encryptar password

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- db_cart_springboot
- db_users.springboot
- msvc_users
- sys

Tables

- users

SQL Additions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid

	id	email	password	username
▶	1	admin@gmail.com	\$2a\$10\$DFNkvmVaHmdt/nt8DrRsD.5VnxjdY... admin	admin
2	juan@gmail.com	\$2a\$10\$@mkhfYfR.FtTmQMBCD8gepu6T6Gm4... juan	juan	
3	pepe@gmail.com	\$2a\$10\$-logWYF0/LsCvBnhDtW0m76vb1e... pepe	pepe	
4	maria@gmail.com	\$2a\$10\$zHRala.H05dgJlaA5nAKu9/7OrBbJykh... maria	maria	
5	pedro@gmail.com	\$2a\$10\$/fXEdgs.e4Ujh5j62edBnGM1zODIN... pedro	pedro	
6	daniela@gmail.com	\$2a\$10\$YYWfqnzXnj15tA6XOmIous3U7axfXRL... daniela	daniela	
*	NULL	NULL	NULL	NULL

Administration Schemas

Information

Table: users

Columns:

#	Name	Type
1	id	bigint AI PK
2	email	varchar(255)
3	password	varchar(8)
4	username	varchar(8)

Action Output

#	Time	Action	Message	Duration / Fetch
1	07:46:03	SELECT * FROM db_users.springboot.users LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
2	07:46:17	TRUNCATE `db_users.springboot`.`users`	OK	0.000 sec
3	07:46:20	SELECT * FROM db_users.springboot.users LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
4	06:57:35	SELECT * FROM db_users.springboot.users LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Buscar

Cerca del récord

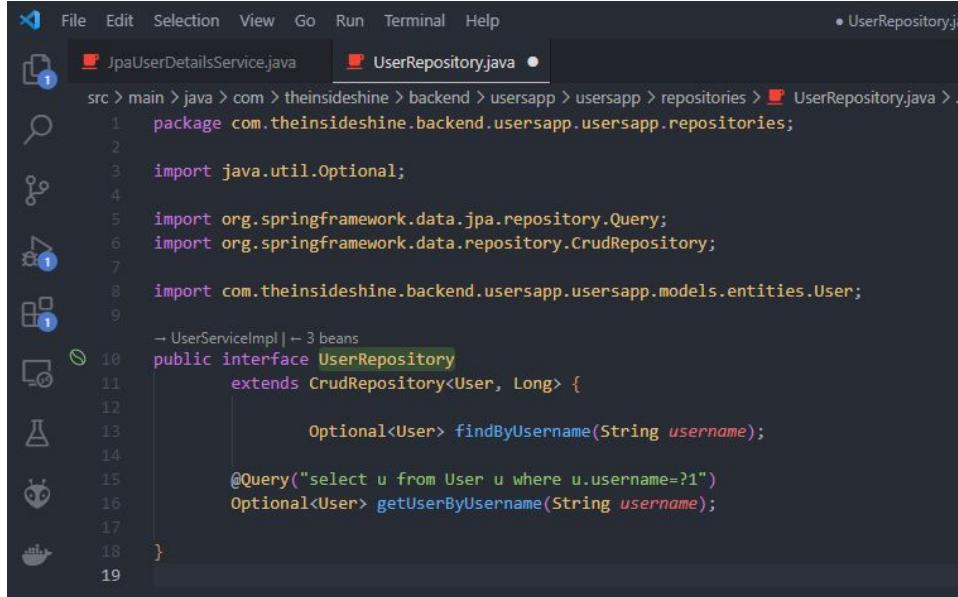
06:57 a.m. 29/05/2023

Login con Jpa

vamos a hacer el login contra la BBDD,

IMPORTANTE: la comparación Con BCryp la hace por debajo SpringSecurity

agregamos al búsqueda por usuario en el repository



The screenshot shows a code editor with two files open:

- UserRepository.java** (highlighted in red)
- JpaUserDetailsService.java** (highlighted in blue)

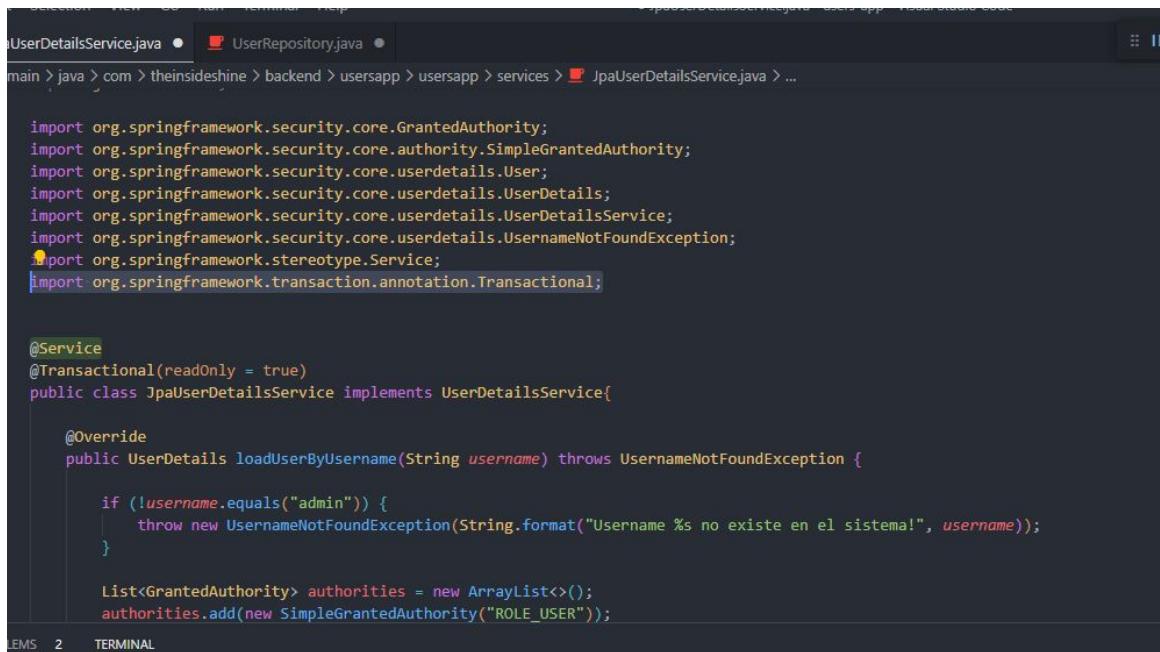
The **UserRepository.java** code is as follows:

```
File Edit Selection View Go Run Terminal Help
src > main > java > com > theinsideshine > backend > usersapp > usersapp > repositories > UserRepository.java > ...
1 package com.theinsideshine.backend.usersapp.usersapp.repositories;
2
3 import java.util.Optional;
4
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.CrudRepository;
7
8 import com.theinsideshine.backend.usersapp.usersapp.models.User;
9
10 public interface UserRepository extends CrudRepository<User, Long> {
11     Optional<User> findByUsername(String username);
12
13     @Query("select u from User u where u.username=?1")
14     Optional<User> getUserByUsername(String username);
15
16 }
17
18 }
```

Login con Jpa

agregar a UserDetailsService el transaccional importado de springframework

agregamos al busqueda por usuario en el repository



```
UserService.java • UserRepository.java •
main > java > com > theinsidesshine > backend > usersapp > usersapp > services > JpaUserDetailsService.java > ...

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional(readOnly = true)
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"));
    }
}
```

File Selection View Go Run Terminal Help

JpaUserDetailsService.java - users-app - Visual Studio Code

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

```
@Service  
@Transactional(readOnly = true)  
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 = new ArrayList<>();  
        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));  
  
        return new User(  
            user.getUsername(),  
            user.getPassword(),  
            true,  
            true,  
            true,  
            true,  
            authorities);  
    }  
}
```

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

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists several collections: 'Collections', 'Environments', 'History', and 'react-andres' (which contains 'cartapp' and 'userapp'). The main workspace displays a POST request to 'localhost:8080/login'. The 'Body' tab is selected, showing the following JSON payload:

```
1 {"username": "juan",  
2 "password": "juan"}  
3  
4
```

Below the body, the 'Test Results' section shows a successful response with status 200 OK, time 118 ms, and size 566 B. The response body is:

```
1 "message": "Hola juan, has iniciado sesion con exito!",  
2 "token": "YnxndW5fdG9rZw5fY29uX2FsZ3VuYV9mcnFzZV9zZWlyZXRhOmp1YW4=",  
3 "username": "juan"  
4
```

At the bottom of the screen, the Windows taskbar is visible with icons for File Explorer, Edge, Task View, and others. The system tray shows the date and time (07:17 a.m. 29/05/2023), battery level (15°C), and network status.

Token jwt

hasta ahora el token los encryptamos con base 64,

```
    @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



The screenshot shows a Java code editor with several tabs at the top: JpaUserDetailsService.java, pom.xml 1, TokenJwtConfig.java (which is the active tab), SpringSecurityConfig.java, and UserRepository.java. Below the tabs is a breadcrumb navigation bar: c > main > java > com > theinsideshine > backend > usersapp > usersapp > auth > TokenJwtConfig.java > TokenJwtConfig. The main pane displays the following Java code:

```
1 package com.theinsideshine.backend.usersapp.usersapp.auth;
2
3 import java.security.Key;
4
5 import io.jsonwebtoken.SignatureAlgorithm;
6 import io.jsonwebtoken.security.Keys;
7
8 public class TokenJwtConfig {
9
10     // public final static String SECRET_KEY = "algún_token_con_alguna_frase_secreta";
11     public final static Key SECRET_KEY = Keys.secretKeyFor(SignatureAlgorithm.HS256);
12     public final static String PREFIX_TOKEN = "Bearer ";
13     public final static String HEADER_AUTHORIZATION = "Authorization";
14
15 }
```

Token jwt

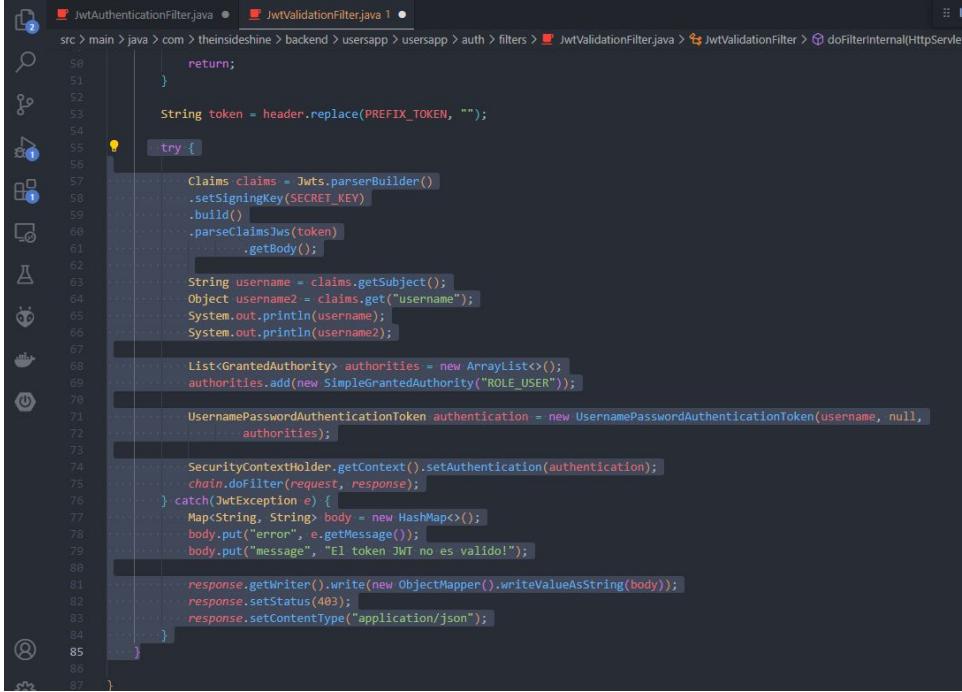
Vamos a cambiar la secret en JwTAuthenticationFilter vamos a cambiar el metodo con la generacion de jwy

```
protected void successfulAuthentication
```

```
59     } catch (IOException e) {
60         e.printStackTrace();
61     }
62     UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(username, password);
63     return authenticationManager.authenticate(authToken);
64 }
65
66
67 @Override
68 protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
69                                     Authentication authResult) throws IOException, ServletException {
70
71     String username = ((org.springframework.security.core.userdetails.User) authResult.getPrincipal())
72             .getUsername();
73
74     String token = Jwts.builder()
75             .setSubject(username)
76             .signWith(SECRET_KEY)
77             .setIssuedAt(new Date())
78             .setExpiration(new Date(System.currentTimeMillis() + 3600000))
79             .compact();
```

Token jwt

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



```
src > main > java > com > theinsideline > backend > usersapp > usersapp > auth > filters > JwtValidationFilter.java > JwtValidationFilter > doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {
    ...
    String token = header.replace(PREFIX_TOKEN, "");
    try {
        Claims claims = Jwts.parserBuilder()
            .setSigningKey(SECRET_KEY)
            .build()
            .parseClaimsJws(token)
            .getBody();
        String username = claims.getSubject();
        Object username2 = claims.get("username");
        System.out.println(username);
        System.out.println(username2);
        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);
    } 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");
    }
}
```

Token jwt

Ahora deberíamos tener en el token el Jwt

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. In the center, a 'POST login' request is being made to 'localhost:8080/login'. The 'Body' tab is selected, showing the following JSON payload:

```
1 {  
2   "username": "admin",  
3   "password": "12345"  
4 }
```

The 'Test Results' section shows the response status as 200 OK, with a response body containing:

```
1 {  
2   "message": "Hola admin, has iniciado session con exito!",  
3   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3M2gibpbIlzImNrcC16MTY4NTQ2MjM4N1ie12XhWijoxNjg1NDY1OTg2fQ,z11mm1lCK3HmFyLkdQZcfi60zUoas7GmstTecb17vI",  
4   "username": "admin"  
5 }
```

At the bottom, the Windows taskbar shows various open applications, and the system tray indicates it's 01:00 p.m. on 30/05/2023.

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

```
@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

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database schema with the `roles` table selected.
- SQL Editor:** Displays the query `SELECT * FROM db_users.springboot.roles;`.
- Result Grid:** Shows the data for the `roles` table:

id	name
ROLE_ADMIN	ROLE_ADMIN
ROLE_USER	ROLE_USER
NULL	NULL
- Output:** Shows the action history for the current session:

#	Time	Action	Message	Duration / Fetch
2	07:46:17	TRUNCATE 'db_users.springboot'.`users`	OK	0.000 sec
3	07:46:20	SELECT * FROM db_users.springboot.users LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
4	06:57:35	SELECT * FROM db_users.springboot.users LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
5	13:31:18	SELECT * FROM db_users.springboot.users LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
6	13:32:30	SELECT * FROM db_users.springboot.roles LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
- System Tray:** Shows the Windows taskbar with various pinned icons and system status information (16°C, 10:35 p.m., 30/05/2023).

Crear Role.java

con apply se aplican los cambios editados

The screenshot shows the MySQL Workbench interface. The main window displays the SQL Editor with the title "Review SQL Script to be Applied on the Database". The script content is as follows:

```
1  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (1, '1');
2  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (1, '2');
3  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (2, '2');
4  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (3, '2');
5  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (4, '2');
6  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (5, '2');
7  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (6, '2');
8  INSERT INTO `db_users_springboot`.`users_roles`(`user_id`, `role_id`) VALUES (7, '2');
```

The Navigator pane on the left shows the database schema, including the "users_roles" table under the "db_users_springboot" schema. The Result Grid pane shows a single row of data from the "users_roles" table.

The bottom section of the interface includes the History tab, which lists recent database actions:

#	Time	Action	Message	Duration / Fetch
3	07:46:20	SELECT * FROM db_users_springboot.users LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
4	06:57:35	SELECT * FROM db_users_springboot.users LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
5	13:31:18	SELECT * FROM db_users_springboot.roles LIMIT 0, 1000	7 row(s) returned	0.000 sec / 0.000 sec
6	13:32:30	SELECT * FROM db_users_springboot.users_roles LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
7	13:36:31	SELECT * FROM db_users_springboot.users_roles LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

The status bar at the bottom right shows the date and time: 01:37 p.m. 30/05/2023.

Crear Role.java

desde postman tiene que traer los usuarios con sus roles

The screenshot shows the Postman interface with the following details:

- Request URL:** GET localhost:8080/users/1
- Authorization:** Bearer Token (with a placeholder token value)
- Response Body (Pretty JSON):**

```
1  {
2     "id": 1,
3     "username": "admin",
4     "password": "$2a$10$FNwmVaHMdt/nt8DzRsD.sVNxKjdy5JpTezebjJorTziMD0DAiJW",
5     "email": "admin@gmail.com",
6     "roles": [
7         {
8             "id": 1,
9             "name": "ROLE_ADMIN"
10        },
11        {
12            "id": 2,
13            "name": "ROLE_USER"
14        }
15    ]
16}
```
- Status:** 200 OK
- Time:** 52 ms
- Size:** 617 B

Crear Role.java

agregar role en el método post

```
pom.xml 1 Role.java 4 UserServiceImpl.java X RoleRepository.java X UserRepository.java X
main > java > com > theinsideshine > backend > usersapp > usersapp > services > UserServiceImpl.java >
1 @Transactional(readOnly = true)
2     public Optional<User> findById(Long id) {
3         return repository.findById(id);
4     }
5
6     @Override
7     @Transactional
8     public User save(User user) {
9         user.setPassword(passwordEncoder.encode(user.getPassword()));
10
11         Optional<Role> o = roleRepository.findByName(name:"ROLE_USER");
12
13         List<Role> roles = new ArrayList<>();
14         if (o.isPresent()) {
15             roles.add(o.orElseThrow());
16         }
17         user.setRoles(roles);
18
19         return repository.save(user);
20     }
21 }
```

```
pom.xml 1 Role.java 4 UserServiceImpl.java X RoleRepository.java X UserRepository.java X
main > java > com > theinsideshine > backend > usersapp > usersapp > repositories > RoleRepository.java >
1 package com.theinsideshine.backend.usersapp.usersapp.repositories;
2
3 import java.util.Optional;
4
5 import org.springframework.data.repository.CrudRepository;
6
7 import com.theinsideshine.backend.usersapp.usersapp.models.entities.Role;
8
9 public interface RoleRepository
10     extends CrudRepository<Role, Long> {
11     Optional<Role> findByName(String name);
12 }
13 }
```

Crear Role.java

debe agregar role user al crear user

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections, environments, and history items. In the center, a POST request is being made to `localhost:8080/users`. The 'Body' tab is selected, showing the following JSON payload:

```
1  {
2     "username": "lalo",
3     "email": "lalo@gmail.com",
4     "password": "lalo"
5 }
```

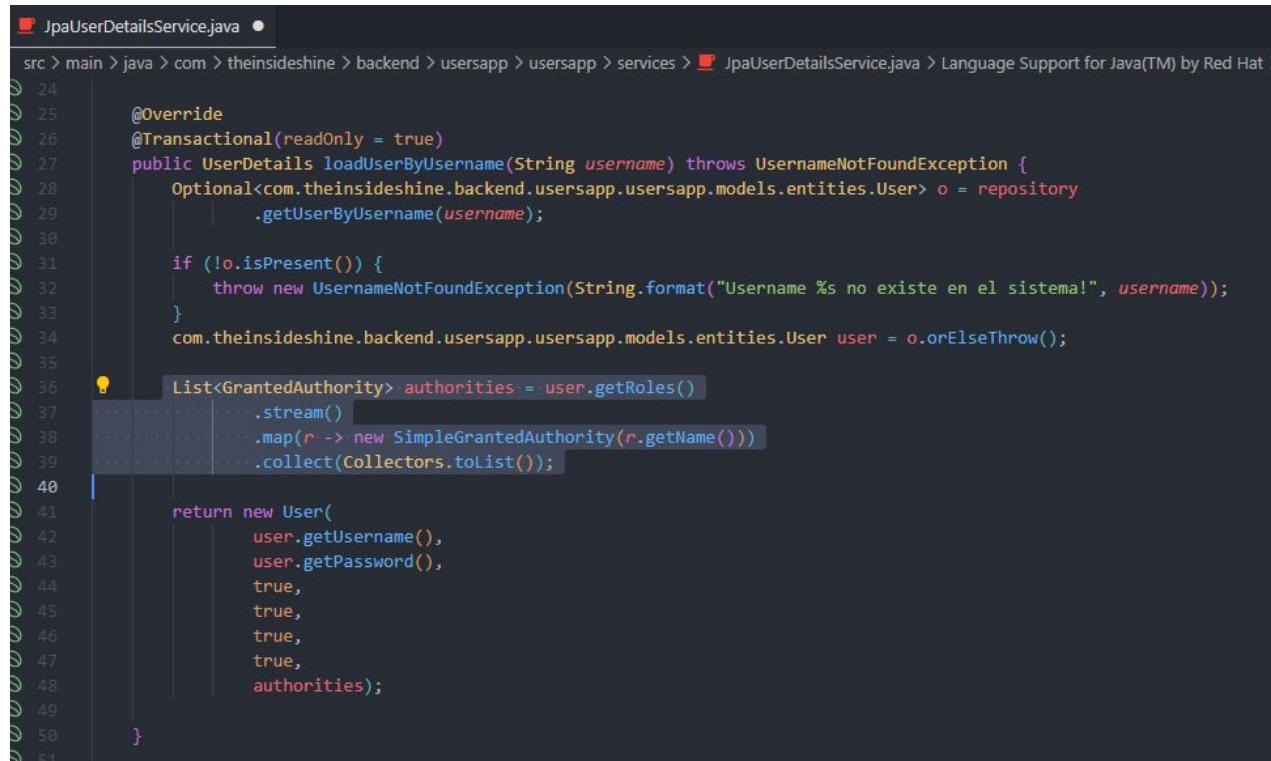
Below the request, the response pane displays the created user's details, including their assigned roles:

```
1  {
2     "id": 8,
3     "username": "lalo",
4     "password": "$2a$10$AgASRlx9.09vdeolm7kgUe5NH8xZnTdnQJfMriPB1l6UKXnaFidt2",
5     "email": "lalo@gmail.com",
6     "roles": [
7         {
8             "id": 2,
9             "name": "ROLE_USER"
10        }
11    ]
12 }
```

The bottom status bar indicates the response was successful with a status of 201 Created, took 391 ms, and had a size of 591 B.

Crear Role.java

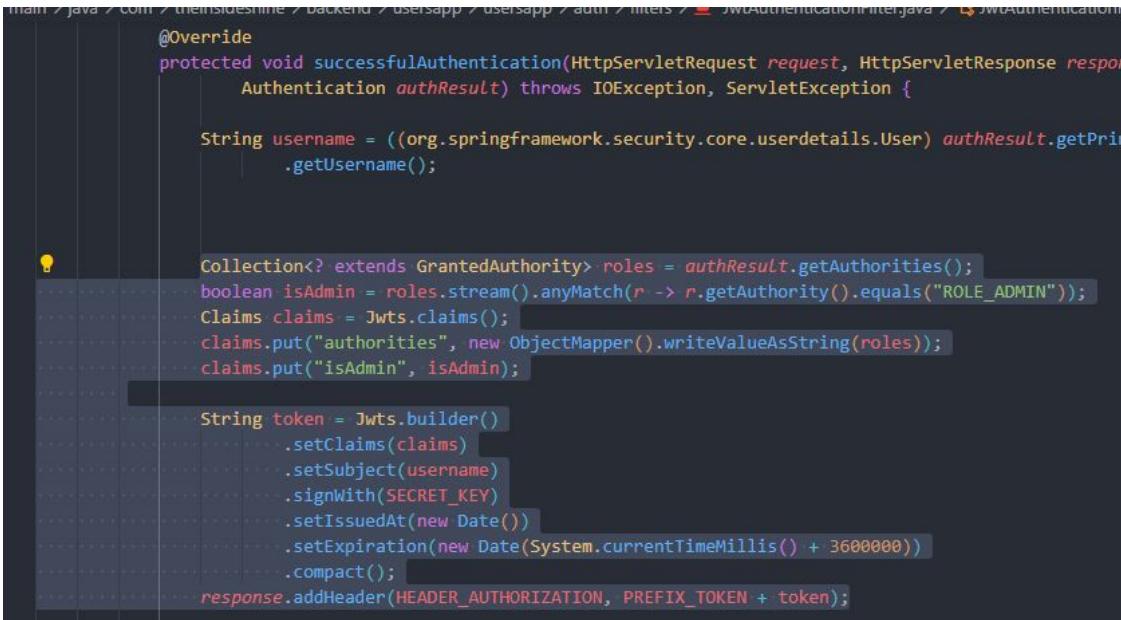
al autenticar ahora vamos a pasar los roles de la BBDD en `loadUserByUsername`



```
src > main > java > com > theinsideshine > backend > usersapp > usersapp > services > JpaUserDetailsService.java > Language Support for Java(TM) by Red Hat >
24
25     @Override
26     @Transactional(readOnly = true)
27     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
28         Optional<com.theinsideshine.backend.usersapp.usersapp.models.entities.User> o = repository
29             .getUserByUsername(username);
30
31         if (!o.isPresent()) {
32             throw new UsernameNotFoundException(String.format("Username %s no existe en el sistema!", username));
33         }
34         com.theinsideshine.backend.usersapp.usersapp.models.entities.User user = o.orElseThrow();
35
36         List<GrantedAuthority> authorities = user.getRoles()
37             .stream()
38             .map(r -> new SimpleGrantedAuthority(r.getName()))
39             .collect(Collectors.toList());
40
41         return new User(
42             user.getUsername(),
43             user.getPassword(),
44             true,
45             true,
46             true,
47             true,
48             authorities);
49
50     }
51
```

Crear Role.java

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



```
main / java / com / themidesigns / backend / usersapp / usersapp / auth / filters / JwtAuthenticationFilter.java / JwtAuthenticationFilter.java

@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
    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);

    String token = Jwts.builder()
        .setClaims(claims)
        .setSubject(username)
        .signWith(SECRET_KEY)
        .setIssuedAt(new Date())
        .setExpiration(new Date(System.currentTimeMillis() + 3600000))
        .compact();
    response.addHeader(H HEADER_AUTHORIZATION, PREFIX_TOKEN + token);
}
```

Crear Role.java

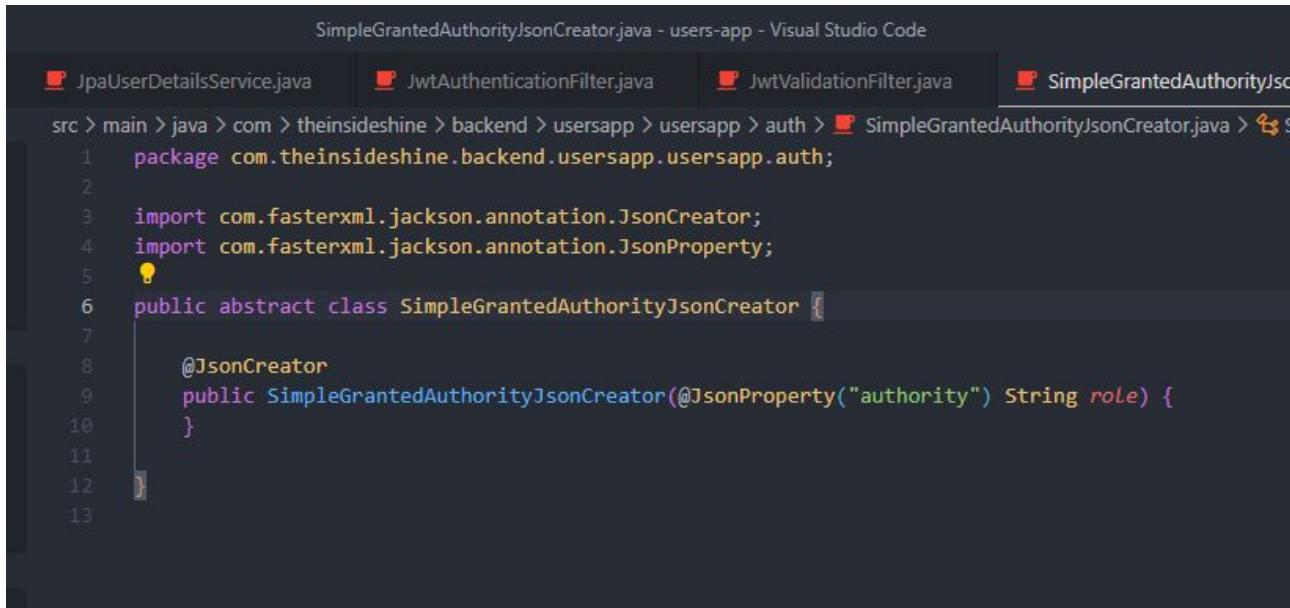
vamos a agrega en el token los roles en `void doFilterInternal`
en `JwtValidationFilter`

```
59
60
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`



The screenshot shows a Visual Studio Code window with the title "SimpleGrantedAuthorityJsonCreator.java - users-app - Visual Studio Code". The tab bar includes "JpaUserDetailsService.java", "JwtAuthenticationFilter.java", "JwtValidationFilter.java", and "SimpleGrantedAuthorityJsonCreator.java". The code editor displays the following Java code:

```
src > main > java > com > theinsideshine > backend > usersapp > usersapp > auth > SimpleGrantedAuthorityJsonCreator.java > SimpleGrantedAuthorityJsonCreator.java
1 package com.theinsideshine.backend.usersapp.usersapp.auth;
2
3 import com.fasterxml.jackson.annotation.JsonCreator;
4 import com.fasterxml.jackson.annotation.JsonProperty;
5
6 public abstract class SimpleGrantedAuthorityJsonCreator {
7
8     @JsonCreator
9     public SimpleGrantedAuthorityJsonCreator(@JsonProperty("authority") String role) {
10
11
12 }
13 }
```

Crear Role.java

ahora debería devolver un token mas largo

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various environments and projects, including 'react-andres' which is currently selected. In the main workspace, a 'POST login' request is being made to 'localhost:8080/login'. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   ... "username": "admin",  
3   ... "password": "12345"  
4 }
```

The response status is 200 OK, with a response body containing a JSON object:

```
1 {  
2   "message": "Hola admin, has iniciado sesion con exito!",  
3   "token": "eyJhbGciOiJIUzI1NiJ9.  
4   eyJhdXRob3JpdGllcyI6I1t7XCCjdXRob3JpdHlciJpcI1JPTEVfQURNSU5cIn0se1wiYXV0aG9yaXR5XCi6XCJSt0xFX1VTRVJcIn1dIiwiXNBZGipbiI6dHj1ZSwic3ViIjoiYwRtaW  
41LC3pXQ10jE20D0UNjM2MDYsImV4cCI6MTY4NTQ2NzIwNn0.5HgEmZLNeYYBkKzkKhQ5VAkhMTFQz49Q1VBdQQQpkI",  
5   "username": "admin"  
6 }
```

Crear Role.java

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

The screenshot shows a browser window with the jwt.io website open. The URL bar shows "jwt.io". The main content area has two sections: "Encoded" on the left and "Decoded" on the right.

Encoded: Displays the raw JWT token:
eyJhbGciOiJIUzI1NiJ9.eyJhdXRob3JpdGllcyI6Ilt7XCJhdXRob3JpdHlcIjpcIlJPTEVfQURNSU5cIn0se1wiYXV0aG9yaXR5XCI6XCJST0xFX1VT RVJcIn1dIiwiiaXNBZG1pbii6dHJ1ZSwic3ViIjo iYWRTaW4iLCJpYXQiOjE2ODU0NjM2MDYsImV4cC I6MTY4NTQ2NzIwNn0.5HgEmZLNeYYBkXzkKHnQS VAkhMTFQz49QIVBdQQ0pkI

Decoded: Displays the decoded JWT structure with roles.
HEADER: ALGORITHM & TOKEN TYPE
{
 "alg": "HS256"
}
PAYOUT: DATA
{
 "authorities": "[{"authority": "ROLE_ADMIN"}, {"authority": "ROLE_USER"}]",
 "isAdmin": true,
 "sub": "admin",
 "iat": 1685463606,
 "exp": 1685467206
}
VERIFY SIGNATURE
HMACSHA256(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload)
)

The browser taskbar at the bottom shows various pinned icons and the system tray with weather information (16°C), date (30/05/2023), and battery status.

Crear Role.java

vamos agregar a las rutas el manejo de roles en SpringSecurityConfig



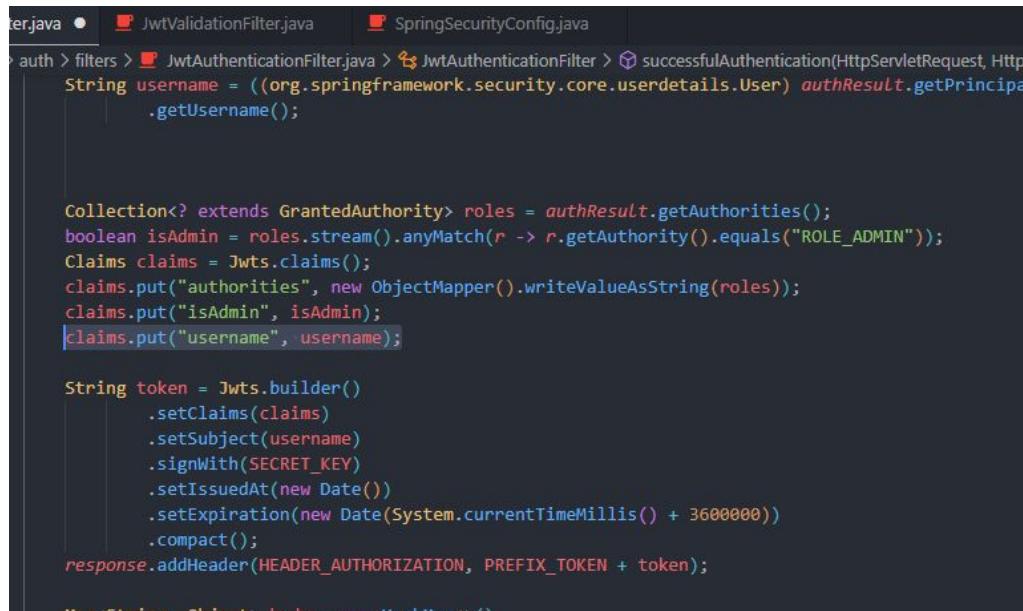
```
ytAuthenticationFilter.java | JwtValidationFilter.java | SpringSecurityConfig.java X
> backend > usersapp > usersapp > auth > SpringSecurityConfig.java > Language Support for Java(TM) by Red Hat > SpringSecurityConfig > filterChain
← 1 bean
@.Autowired
private AuthenticationConfiguration authenticationConfiguration;

→ 1 bean | ← 1 bean
@Bean
SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http.authorizeHttpRequests()
        .requestMatchers(HttpMethod.GET, "/users").permitAll()
        .requestMatchers(HttpMethod.GET, "/users/{id}").hasAnyRole("USER", "ADMIN")
        .requestMatchers(HttpMethod.POST, "/users").hasRole("ADMIN")
        .requestMatchers("/users/**").hasRole("ADMIN")
        // .requestMatchers(HttpMethod.DELETE, "/users/{id}").hasRole("ADMIN")
        // .requestMatchers(HttpMethod.PUT, "/users/{id}").hasRole("ADMIN")
        .anyRequest().authenticated()
        .and()
        .addFilter(new JwtAuthenticationFilter(authenticationConfiguration.getAuthenticationManager()))
        .addFilter(new JwtValidationFilter(authenticationConfiguration.getAuthenticationManager()))
        .csrf(config -> config.disable())
        .sessionManagement(management -> management.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .build();
}
```

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



The screenshot shows a Java code editor with the file `JwtAuthenticationFilter.java` open. The code is part of a Spring Security filter chain, specifically the `JwtAuthenticationFilter`. It retrieves the authenticated user from the `authResult` and extracts their `username`, `roles`, and `isAdmin` status. It then creates a `Claims` object with these details and signs it using a secret key to generate a JWT token. Finally, it adds the token to the response header.

```
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);

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);
```

Detalles

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



```
    return;
}

String token = header.replace(PREFIX_TOKEN, "");

try {
    Claims claims = Jwts.parserBuilder()
        .setSigningKey(SECRET_KEY)
        .build()
        .parseClaimsJws(token)
        .getBody();

    Object authoritiesClaims = claims.get("authorities");
    String username = claims.getSubject();
    Object username2 = claims.get("username");
    System.out.println(username);
    System.out.println(username2);

    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);
}

```

The screenshot shows a Java code editor with syntax highlighting. The code is part of a class that handles token validation. It uses the Jwts parser to extract claims from a token. It then prints the subject (username) and an additional 'username' claim to the console. Finally, it creates a UsernamePasswordAuthenticationToken object with the extracted username and authorities.

Detalles

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

```
PROBLEMS 6 TERMINAL  Run: UsersAppApplication +   ...   
2023-05-30T13:08:20.846-03:00 DEBUG 3432 --- [nio-8080-exec-4] org.hibernate.SQL : select r1_0.user_id,r1_1.id,r1_1.name from users_r  
oles r1_0 join roles r1_1 on r1_1.id=r1_0.role_id where r1_0.user_id=?  
Hibernate: select r1_0.user_id,r1_1.id,r1_1.name from users_roles r1_0 join roles r1_1 on r1_1.id=r1_0.role_id where r1_0.user_id=?  
admin  
admin  
2023-05-30T13:11:40.117-03:00 DEBUG 3432 --- [nio-8080-exec-8] org.hibernate.SQL : select u1_0.id,u1_0.email,u1_0.password,u1_0 usern  
ame from users u1_0 where u1_0.id=?  
Hibernate: select u1_0.id,u1_0.email,u1_0.password,u1_0.username from users u1_0 where u1_0.id=?  
2023-05-30T13:11:40.123-03:00 DEBUG 3432 --- [nio-8080-exec-8] org.hibernate.SQL : select r1_0.user_id,r1_1.id,r1_1.name from users_r  
oles r1_0 join roles r1_1 on r1_1.id=r1_0.role_id where r1_0.user_id=?  
Hibernate: select r1_0.user_id,r1_1.id,r1_1.name from users_roles r1_0 join roles r1_1 on r1_1.id=r1_0.role_id where r1_0.user_id=?
```

Detalles

Se vemos el token devuelto vemos la info de user name

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJhdXRob3JpdGllcyI6Il...  
eyJhbGciOiJIUzI1NiJ9.eyJhdXRob3JpdGllcyI6Il...  
U5cIn0se1wiYXV0aG9yaXR5XCI6XCJST0xFX1VT  
RVJcIn1dIiwiaXNBZG1pbiiI6dHJ1ZSwidXNlcm5  
hbWUiOjJhZG1pbiiIsInN1YiI6ImFkbWluIiwiaW  
F0IjoxNjg1NDYyOTAwLCJleHAiOjE2ODU0NjY1M  
DB9.AMPQGeJv9kGumeA1A7GKcjZ1NehQLZ1-  
eVmELQAhNOY
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

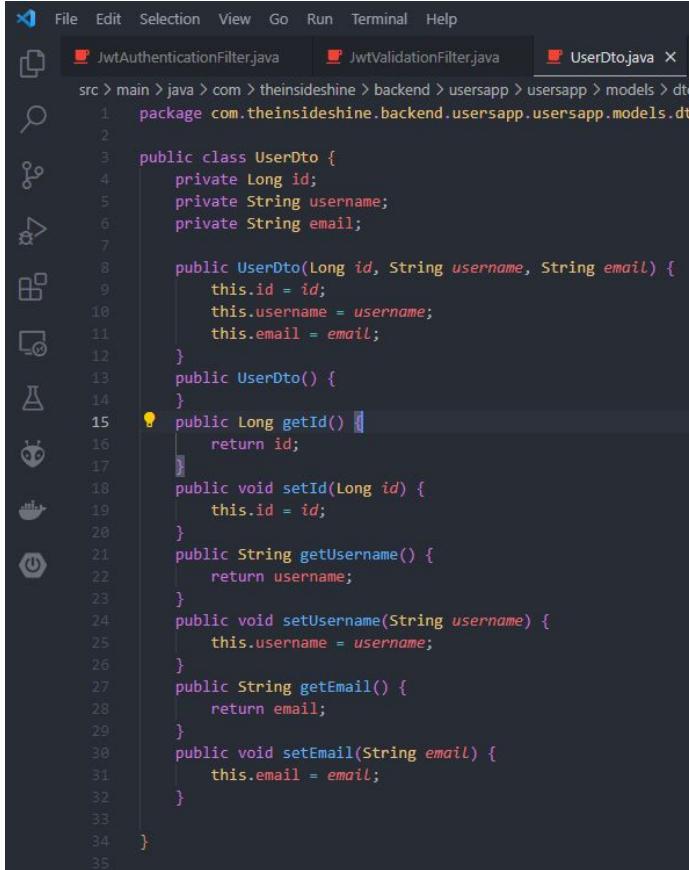
```
{  
  "alg": "HS256"  
}
```

PAYLOAD: DATA

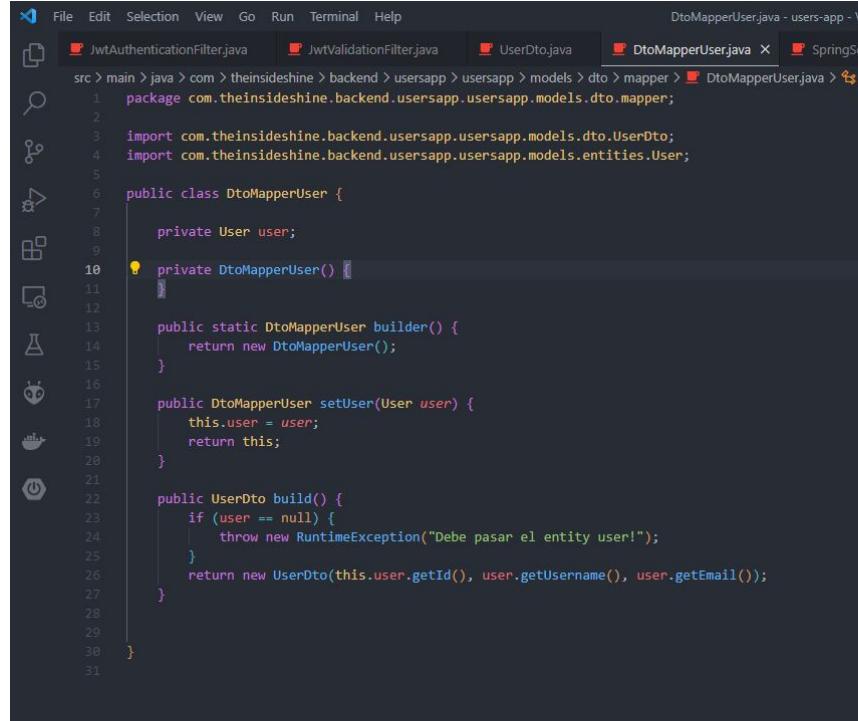
```
{  
  "authorities": "[{\\"authority\\":\\"ROLE_ADMIN\\"},  
 {\\"authority\\":\\"ROLE_USER\\"}]",  
  "isAdmin": true,  
  "username": "admin",  
  "sub": "admin",  
  "iat": 1685462900,  
  "exp": 1685466500  
}
```

Dto

Crear Dto y dtoMapper



```
src > main > java > com > theinsideshine > backend > usersapp > usersapp > models > dto
1 package com.theinsideshine.backend.usersapp.usersapp.models.dto;
2
3 public class UserDto {
4     private Long id;
5     private String username;
6     private String email;
7
8     public UserDto(Long id, String username, String email) {
9         this.id = id;
10        this.username = username;
11        this.email = email;
12    }
13    public UserDto() {
14    }
15    public Long getId() {
16        return id;
17    }
18    public void setId(Long id) {
19        this.id = id;
20    }
21    public String getUsername() {
22        return username;
23    }
24    public void setUsername(String username) {
25        this.username = username;
26    }
27    public String getEmail() {
28        return email;
29    }
30    public void setEmail(String email) {
31        this.email = email;
32    }
33}
34}
```

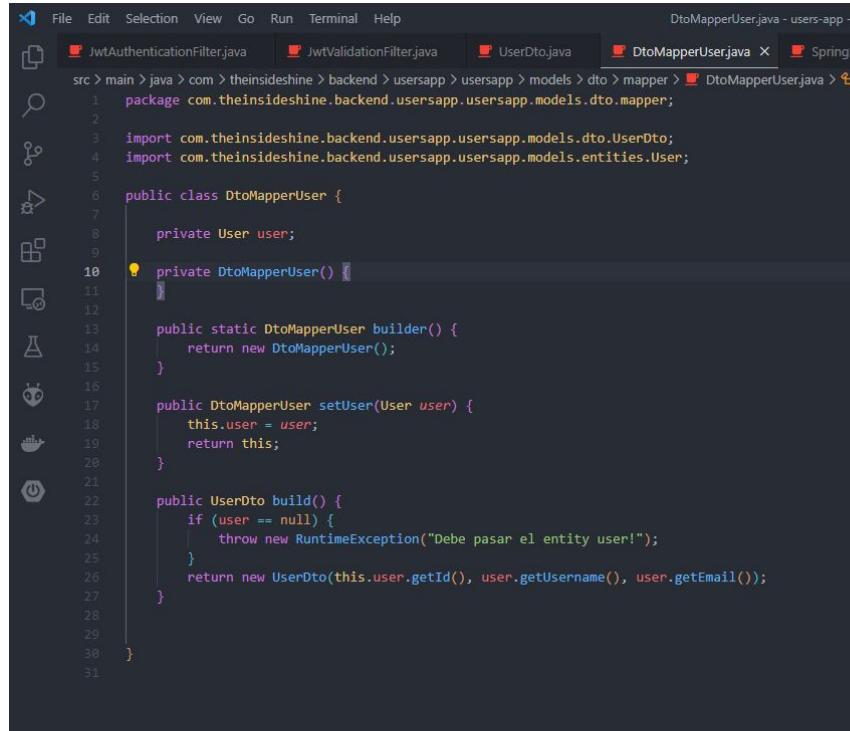


```
src > main > java > com > theinsideshine > backend > usersapp > usersapp > models > dto > mapper > DtoMapperUser.java
1 package com.theinsideshine.backend.usersapp.usersapp.models.dto.mapper;
2
3 import com.theinsideshine.backend.usersapp.usersapp.models.UserDto;
4 import com.theinsideshine.backend.usersapp.usersapp.models.entities.User;
5
6 public class DtoMapperUser {
7
8     private User user;
9
10    private DtoMapperUser() {
11    }
12
13    public static DtoMapperUser builder() {
14        return new DtoMapperUser();
15    }
16
17    public DtoMapperUser setUser(User user) {
18        this.user = user;
19        return this;
20    }
21
22    public UserDto build() {
23        if (user == null) {
24            throw new RuntimeException("Debe pasar el entity user!");
25        }
26        return new UserDto(this.user.getId(), user.getUsername(), user.getEmail());
27    }
28
29
30}
31
```

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



```
File Edit Selection View Go Run Terminal Help
src > main > java > com > theinsideshine > backend > usersapp > usersapp > models > dto > mapper > DtoMapperUser.java > DtoMapperUser.java
1 package com.theinsideshine.backend.usersapp.usersapp.models.dto.mapper;
2
3 import com.theinsideshine.backend.usersapp.usersapp.models.UserDto;
4 import com.theinsideshine.backend.usersapp.usersapp.models.entities.User;
5
6 public class DtoMapperUser {
7
8     private User user;
9
10    private DtoMapperUser() {
11    }
12
13    public static DtoMapperUser builder() {
14        return new DtoMapperUser();
15    }
16
17    public DtoMapperUser setUser(User user) {
18        this.user = user;
19        return this;
20    }
21
22    public UserDto build() {
23        if (user == null) {
24            throw new RuntimeException("Debe pasar el entity user!");
25        }
26        return new UserDto(this.user.getId(), user.getUsername(), user.getEmail());
27    }
28
29
30
31 }
```

Dto

modificando los services

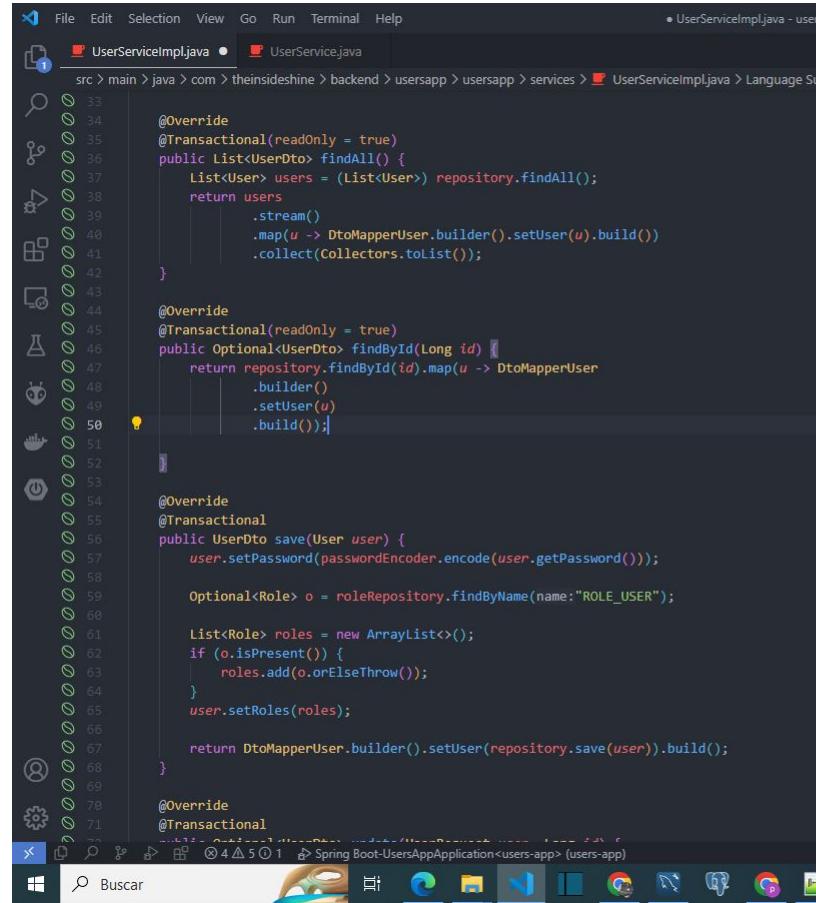
The screenshot shows a Java code editor with two files open: `UserServiceImpl.java` and `UserService.java`. The `UserService.java` file is the one being edited, showing the following code:

```
File Edit Selection View Go Run Terminal Help
src > main > java > com > theinsidesshine > backend > usersapp > usersapp > services > UserService.java - users-app - Vis
UserService.java 1 UserService.java

1 package com.theinsidesshine.backend.usersapp.usersapp.services;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import com.theinsidesshine.backend.usersapp.usersapp.models.dto.UserDto;
7 import com.theinsidesshine.backend.usersapp.usersapp.models.entities.User;
8 import com.theinsidesshine.backend.usersapp.usersapp.models.request.UserRequest;
9
10 public interface UserService {
11     ...
12     List<UserDto> findAll();
13
14     Optional<UserDto> findById(Long id);
15
16     UserDto save(User user);
17     Optional<UserDto> update(UserRequest user, Long id);
18
19     void remove(Long id);
20 }
21
```


Dto

modificando los services



The screenshot shows a Java code editor with the file `UserServiceImpl.java` open. The code implements a service interface `UserService` for managing users. It includes methods for finding all users, finding a user by ID, saving a user, and updating a user's password.

```
File Edit Selection View Go Run Terminal Help
UserServiceImpl.java • UserService.java
src > main > java > com > theinsideshire > backend > usersapp > usersapp > services > UserServiceImpl.java > Language Support

@Override
@Transactional(readOnly = true)
public List<UserDto> findAll() {
    List<User> users = (List<User>) repository.findAll();
    return users
        .stream()
        .map(u -> DtoMapperUser.builder().setUser(u).build())
        .collect(Collectors.toList());
}

@Override
@Transactional(readOnly = true)
public Optional<UserDto> findById(Long id) {
    return repository.findById(id).map(u -> DtoMapperUser
        .builder()
        .setUser(u)
        .build());
}

@Override
@Transactional
public UserDto 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 DtoMapperUser.builder().setUser(repository.save(user)).build();
}

@Override
@Transactional
public void update(User user) {
    User existingUser = repository.findById(user.getId()).orElseThrow();
    existingUser.setName(user.getName());
    existingUser.setPassword(passwordEncoder.encode(user.getPassword()));
    existingUser.setRoles(user.getRoles());
    repository.save(existingUser);
}
```

Dto

modificando los services

```
 69
 70     @Override
 71     @Transactional
 72     public Optional<UserDto> update(UserRequest user, Long id) {
 73         Optional<User> o = repository.findById(id);
 74         User userOptional = null;
 75         if (o.isPresent()) {
 76             User userDb = o.orElseThrow();
 77             userDb.setUsername(user.getUsername());
 78             userDb.setEmail(user.getEmail());
 79             userOptional = repository.save(userDb);
 80         }
 81         return Optional.ofNullable(DtoMapperUser.builder().setUser(userOptional).build());
 82     }
 83 }
```

Dto

Ahora devuelvo solo el DTO

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections and environments. The main workspace shows a collection named 'react-andres' with a sub-collection 'userapp'. Under 'userapp', there are several requests: 'PUT localhost:8080/users', 'GET localhost:8080/users', 'GET localhost:8080/users/id', 'POST localhost:8080/users', 'DEL localhost:8080/users', and 'POST login'. The 'GET localhost:8080/users' request is selected and expanded. The 'Authorization' tab is active, stating that an authorization header will be automatically generated. The 'Body' tab displays the response in JSON format:

```
1 [  
2   {  
3     "id": 1,  
4     "username": "admin",  
5     "email": "admin@gmail.com"  
6   },  
7   {  
8     "id": 2,  
9     "username": "juan",  
10    "email": "juan@gmail.com"  
11  },  
12  {  
13    "id": 3,  
14  }]
```

The status bar at the bottom indicates the request was successful (Status: 200 OK), took 23 ms, and was 855 B in size.

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

SpringSecurityConfig

JwtAuthorizationFilter

JwtValidationFilter

TokenJwtConfig

resumen

La clase `SimpleGrantedAuthorityJsonCreator` se crea para pasar los roles en la validación y la JpaUserDetailsService realiza la consulta contra la base de datos, implementando la interface UserDetailsService

JwtValidationFilter

`SimpleGrantedAuthorityJsonCreator`

JpaUserDetailsService

File Edit Selection View Go Run Terminal Help JpaUserDetailsService.java - users-app - Visual Studio Code

```
src > main > java > com > theinsideshine > backend > usersapp > usersapp > services > JpaUserDetailsService.java > ...
```

19 @Service
20 public class JpaUserDetailsService implements UserDetailsService {
21
22 @Autowired
23 private UserRepository repository;
24
25 @Override
26 @Transactional(readOnly = true)
27 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
28 Optional<com.theinsideshine.backend.usersapp.usersapp.models.entities.User> o = repository
29 .getUserByUsername(username);
30
31 if (!o.isPresent()) {
32 throw new UsernameNotFoundException(String.format("Username %s no existe en el sistema!", username));
33 }
34 com.theinsideshine.backend.usersapp.usersapp.models.entities.User user = o.orElseThrow();
35
36 List<GrantedAuthority> authorities = user.getRoles()
37 .stream()
38 .map(r -> new SimpleGrantedAuthority(r.getName()))
39 .collect(Collectors.toList());
40
41 return new User(
42 user.getUsername(),
43 user.getPassword(),
44 true,
45 true,
46 true,
47 true,
48 authorities);
49 }
50 }

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 13°C Nublado 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 SimpleGrantedAuthority.java

.ideshine > backend > usersapp > usersapp > auth > filters > JwtAuthenticationFilter.java > JwtAuthenticationFilter > successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain, Authentication authResult) throws IOException, ServletException {

```
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(H 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 UFT-8 CRLF {} Java

Escribe aquí para buscar       

14°C Mayorm. nubla.. 01:17 p.m. 30/05/2023

UserServiceImpl.java

SpringSecurityConfig.java

JwtAuthenticationFilter.java

JwtValidationFilter.java

src \ main \ java \ com \ theinsideshine \ backend \ usersapp \ usersapp \ auth \ filters \ JwtValidationFilter.java > 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
SimpleGrantedAuthorityJsonCreator 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);
```

} 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");
```

Guarda el usuario en el
contexto de spring

Escribe aquí para buscar

