TP Sécurité

Partie 1:

1) Créer un projet « Spring Security JWT »
2) Ajouter les dépendances suivantes :

Web

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

Jpa

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

Mysql Driver

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

Lombok

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

Validation

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

Spring Security

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

Jsonwebtoken

```
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>
```

3) Modifier Application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/BDTest?createDatabaseIfNotExist=true&useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto= update

server.port=8088

# App Properties
ahlem.app.jwtSecret= ahlemSecretKey
ahlem.app.jwtExpirationMs= 3600000
ahlem.app.jwtRefreshExpirationMs= 86400000
```

4) Ajouter deux classe (entités) : User et Role.

User :

```java
package com.example.projettest.models;

import jakarta.persistence.*;


import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Size;
import java.util.HashSet;
import java.util.Set;
7 usages
@Entity
@Table( name = "users",
        uniqueConstraints = {
            @UniqueConstraint(columnNames = "username"),
            @UniqueConstraint(columnNames = "email")
        })
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```



```java
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    3 usages
    @NotBlank
    @Size(max = 20)
    private String username;

    3 usages
    @NotBlank
    @Size(max = 50)
    @Email
    private String email;
```

```
    3 usages
    @NotBlank
    @Size(max = 120)
    private String password;

    2 usages
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable( name = "user_roles",
                joinColumns = @JoinColumn(name = "user_id"),
                inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles = new HashSet<>();
```

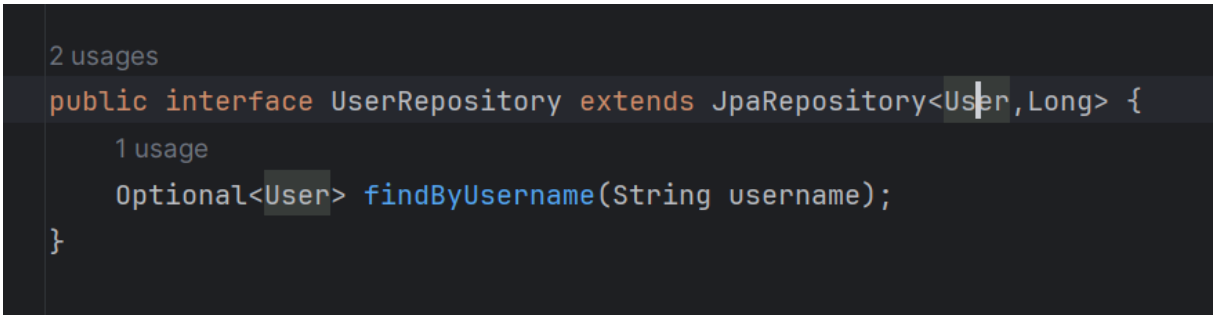
Role :

```
package com.example.projettest.models;

import jakarta.persistence.*;

3 usages
@Entity
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    3 usages
    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private ERole name;

    public Role() {

    }
```

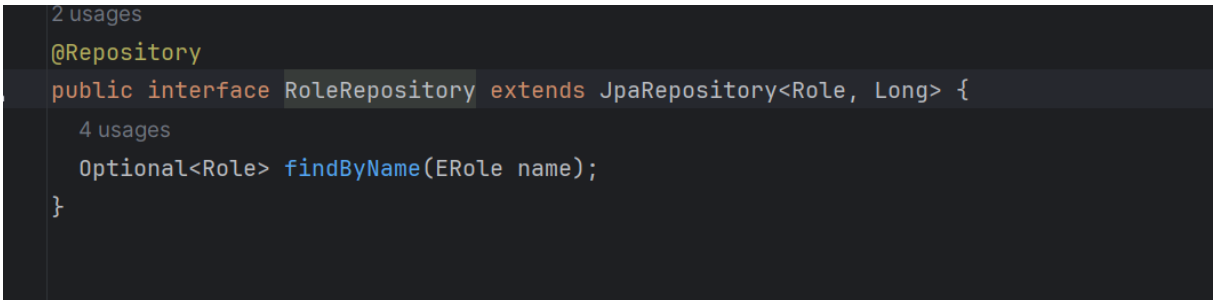
ERole :

5) Ajouter les Repository pour Chaque Classe :

Pour User



Pour Role



6) Ajouter un Package Service Contient deux Classe :

UserDetailsImp :

```java
import ...
7 usages
public class UserDetailsImpl implements UserDetails {
    no usages
    private static final long serialVersionUID = 1L;
    4 usages
    private Long id;
    2 usages
    private String username;
    2 usages
    private String email;
    2 usages
    @JsonIgnore
    private String password;
    2 usages
    private Collection<? extends GrantedAuthority> authorities;
```

```java
private Collection<? extends GrantedAuthority> authorities;
1 usage
public UserDetailsImpl(Long id, String username, String email, String password,
                       Collection<? extends GrantedAuthority> authorities) {
    this.id = id;
    this.username = username;
    this.email = email;
    this.password = password;
    this.authorities = authorities;
}
1 usage
```

```java
1 usage
public static UserDetailsImpl build(User user) {
    List<GrantedAuthority> authorities = user.getRoles().stream() Stream<Role>
            .map(role -> new SimpleGrantedAuthority(role.getName().name())) Stream<SimpleGra
            .collect(Collectors.toList());

    return new UserDetailsImpl(
            user.getId(),
            user.getUsername(),
            user.getEmail(),
            user.getPassword(),
            authorities);
}
```

UserDetailsServiceImp

```java
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    UserRepository userRepository;
    1 usage
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User Not Found with username: "
                        + username));

        return UserDetailsImpl.build(user);
    }
}
```

7) Créer un package JWT : qui Contient 3 classes :

JwtUtils :

```java
2 usages
@Component
public class JwtUtils {
    5 usages
    private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

    @Value("${ahlem.app.jwtSecret}")
    private String jwtSecret;

    @Value("${ahlem.app.jwtExpirationMs}")
    private int jwtExpirationMs;


    no usages
    public String generateJwtToken(UserDetailsImpl userPrincipal) {
        return generateTokenFromUsername(userPrincipal.getUsername());
    }
```

```java
1 usage
public String generateTokenFromUsername(String username) {
    return Jwts.builder().setSubject(username).setIssuedAt(new Date())
            .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs)).signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();
}
1 usage
public String getUserNameFromJwtToken(String token) {
    return Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
}
```

```
    1 usage
    public boolean validateJwtToken(String authToken) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
            return true;
        } catch (SignatureException e) {
            logger.error("Invalid JWT signature: {}", e.getMessage());
        } catch (MalformedJwtException e) {
            logger.error("Invalid JWT token: {}", e.getMessage());
        } catch (ExpiredJwtException e) {
            logger.error("JWT token is expired: {}", e.getMessage());
        } catch (UnsupportedJwtException e) {
            logger.error("JWT token is unsupported: {}", e.getMessage());
        } catch (IllegalArgumentException e) {
            logger.error("JWT claims string is empty: {}", e.getMessage());
        }

        return false;
    }
}
```

AuthTokenFilter

```
@Component
public class AuthTokenFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtils jwtUtils;

    @Autowired
    private UserDetailsServiceImpl userDetailsService;
    1 usage
    private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);
    no usages
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        try {
            String jwt = parseJwt(request);
            if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
                String username = jwtUtils.getUserNameFromJwtToken(jwt);
                UserDetails userDetails = userDetailsService.loadUserByUsername(username);
                UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails,
                        credentials: null, userDetails.getAuthorities());
                authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authentication);}
        } catch (Exception e) {
            logger.error("Cannot set user authentication: {}", e.getMessage());
        }
        filterChain.doFilter(request, response);
    }
```

```
1 usage
private String parseJwt(HttpServletRequest request) {
    String headerAuth = request.getHeader( s: "Authorization");

    if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
        return headerAuth.substring(7, headerAuth.length());
    }

    return null;
}
```