

## Simon Gawar Dak

---

### Security Implementation – CMS Project

#### Overview

The CMS Project integrates security across five microservices using **Spring Security** and **JWT (JSON Web Tokens)**. The goal is to ensure secure access to endpoints, enforce authentication, and support role-based authorization.

The CMS module's structure

CMS-Project/

```
|— pom.xml  
|— common-security/  
|— apigateway/  
|— eurekaserver/  
|— userservice/  
|— courseservice/  
|— assessmentservice/  
|— analyticsservice/  
└— notificationservice/  
  
src/  
└— main/  
    └— java/  
        └— com/  
            └— cms/
```

```
└─ userassessment_service/
    ├─ controller/
    |   └─ UserAssessmentController.java
    ├─ dto/
    |   └─ UserAssessmentDto.java
    ├─ entity/
    |   └─ UserAssessment.java
    ├─ repository/
    |   └─ UserAssessmentRepository.java
    ├─ service/
    |   └─ UserAssessmentService.java
    └─ service
        └─ UserAssessmentServiceApplication.java
```

## Microservices Secured

- User\_service
- Course\_service
- Userassessment\_service
- Notification\_service
- Analytics\_service

## Key Security Components

### 1. JwtTokenProvider.java

This component handles:

- **Token Generation:** Creates JWTs with username and role claims.
- **Token Parsing:** Extracts user information from the token.
- **Token Validation:** Verifies token integrity and expiration.

**Highlights:**

```
public String generateToken(UserDetails userDetails)  
public String getUsernameFromToken(String token)  
public String getRoleFromToken(String token)  
public boolean validateToken(String token)
```

## 2. JwtAuthenticationFilter.java

This filter:

- Intercepts incoming HTTP requests.
- Extracts and validates the JWT from the Authorization header.
- Sets the authenticated user in the SecurityContext.

**Highlights:**

```
@Override  
protected void doFilterInternal(@NotNull HttpServletRequest request,  
                               @NotNull HttpServletResponse response,  
                               @NotNull FilterChain filterChain)
```

## 3. SecurityConfig.java

This configuration:

- Disables CSRF for stateless sessions.
- Permits public access to /api/auth/\*\* and /api/users/public/\*\*.
- Secures all other endpoints.
- Registers JwtAuthenticationFilter before Spring's default filter.
- Uses BCryptPasswordEncoder for password hashing.

### **Highlights:**

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http)
```

@Bean

```
public PasswordEncoder passwordEncoder()
```

### **Security Flow**

1. **User Login:** Authenticated via /api/auth/login, receives JWT.
2. **Token Usage:** JWT is sent in the Authorization header for protected endpoints.
3. **Request Filtering:** JwtAuthenticationFilter validates the token.
4. **Access Control:** Role-based access enforced via Spring Security.

After successfully building jars files for all

```
[INFO] userservice 0.0.1-SNAPSHOT ..... SUCCESS
[ 39.063 s]
[INFO] course-service 0.0.1-SNAPSHOT ..... SUCCESS
[ 25.253 s]
[INFO] user-assessment-service 0.0.1-SNAPSHOT ..... SUCCESS
[ 7.597 s]
[INFO] analyticsservice 0.0.1-SNAPSHOT ..... SUCCESS
[ 24.620 s]
[INFO] notificationservice 0.0.1-SNAPSHOT ..... SUCCESS
[ 24.873 s]
[INFO] learning-platform-parent 1.0.0 ..... SUCCESS
[ 0.007 s]
[INFO] -----
-----
[INFO] BUILD SUCCESS
```

\ 4 ⌂ Connect ⌂ Downloading source jar from known Maven repositories for local repository

## Testing

- Postman workspace includes:
  - Login and registration requests.
  - Protected endpoint access with JWT.
  - Invalid token scenarios.

## Summary

The project phase 2 security implementation covered the five microservices that provide security protection components including **JwtTokenProvider.java**, **JwtAuthenticationFilter.java** and **SecurityConfig.java**. The testing was carried out via postman.