# Security Implementation: Access and Refresh Tokens

Welcome to the security documentation for the you_are_the_hero project. This document will guide contributors on how access tokens and refresh tokens are used within our application, explain their roles, and provide examples to help you contribute effectively.

## Overview

In our application, we use **JWT (JSON Web Tokens)** to secure communication between the client and the server. JWTs are used as access tokens, and we plan to introduce refresh tokens to provide extended sessions securely.

The following sections outline how to **generate**, **validate**, and **refresh tokens** in the context of our Spring Boot application.

---

## Table of Contents

---

## 1. Access Tokens vs Refresh Tokens

### Access Tokens

- **Purpose**: Used to authenticate users during every request.
- **Lifetime**: Short-lived, usually around 10 minutes to a few hours.
- **Usage**: Sent by the client in the Authorization header (Bearer <token>) during API requests.
- **Storage**: Typically stored in-memory or local storage (for web clients).

### Refresh Tokens

- **Purpose**: Used to obtain a new access token without re-authenticating the user.
- **Lifetime**: Long-lived, lasting days or weeks.
- **Usage**: Sent to a dedicated endpoint to refresh the access token once it expires.

- **Storage**: Kept securely in Http Only cookies or secure storage to prevent misuse.

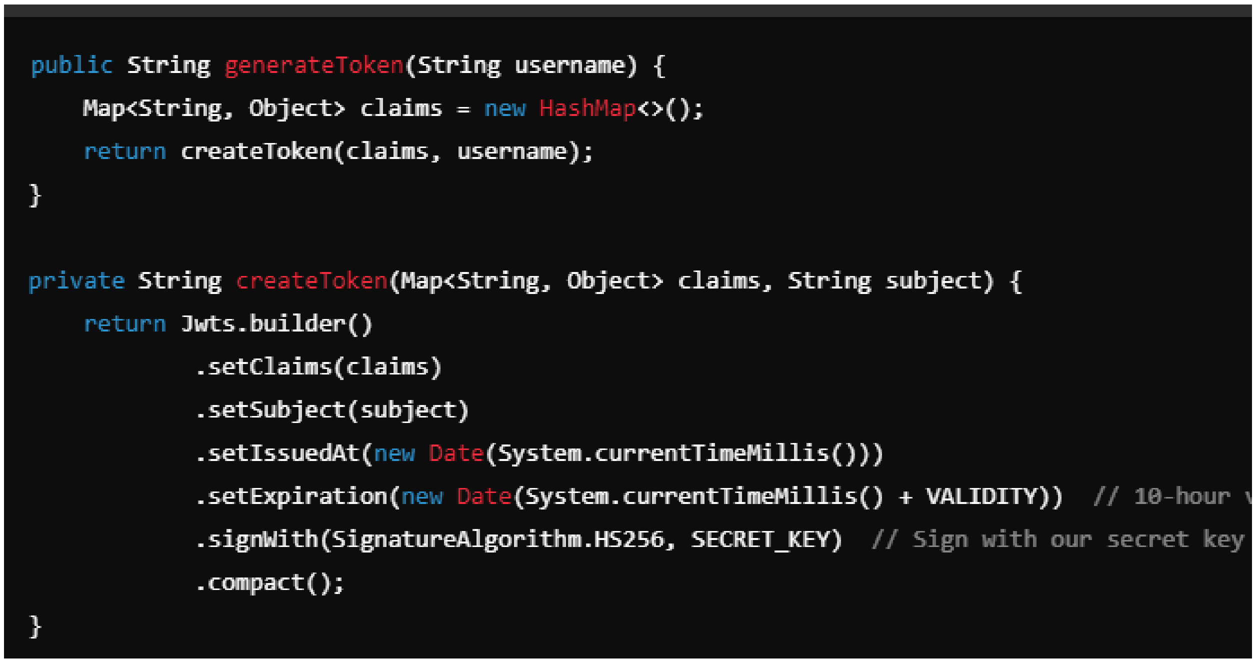
**2. JWT Generation**

In the application, JWTs are generated by the **JwtUtil** class. This utility signs tokens with a secret key and includes claims such as the username and expiration date.

**Token Generation Process:**

1. A client sends their credentials (username/password) to the /auth/login endpoint.
2. If valid, the authentication manager processes the login and generates a JWT with:
   - **Subject**: The username of the user.
   - **Claims**: Additional data such as roles.
   - **Expiration**: 10 hours by default.

**Code Snippet** (from JwtUtil):



```java
public String generateToken(String username) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, username);
}

private String createToken(Map<String, Object> claims, String subject) {
    return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + VALIDITY))  // 10-hour
            .signWith(SignatureAlgorithm.HS256, SECRET_KEY)  // Sign with our secret key
            .compact();
}
```

---

**3. JWT Validation**

The **AuthenticationFilter** class intercepts requests to validate the JWT. If the token is valid and not expired, the request is allowed to access secured endpoints.

**Token Validation Process:**

- The token is extracted from the request header (Authorization: Bearer <token>).
- The **JwtUtil** class checks:

1. If the token is valid (correct signature).
2. If the token has expired.
3. If the token's user matches the current user.

```
public Boolean validateToken(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}
```

**4. Token Refresh Process**

A refresh token system allows the user to stay authenticated without needing to log in repeatedly. Although not implemented in the current system, here's how you can introduce refresh tokens:

1. **Generate Refresh Token**: Alongside the access token, generate a refresh token with a longer expiration (e.g., 7 days).
2. **Store Refresh Token Securely**: Store the refresh token in an HttpOnly cookie to enhance security.
3. **Refresh Endpoint**: Provide an endpoint /auth/refresh where the client can send the refresh token to obtain a new access token once the old one expires.
4. **Validate and Issue New Token**: Upon receiving the refresh token, validate it and issue a new access token if valid.

```
public String refreshAccessToken(String refreshToken) {
    if (validateRefreshToken(refreshToken)) {
        String username = extractUsername(refreshToken);
        return generateToken(username);  // Generate new access token
    }
    throw new Exception("Invalid refresh token");
}
```

**. Code Examples**

**Login Request (Access Token Generation)**

```
POST /auth/login
{
  "username": "john_doe",
  "password": "password123"
}
```

Response

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

**Using Access Token**

Include the token in the Authorization header:

```
GET /api/secure-endpoint
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Refreshing the Access Token

```
POST /auth/refresh
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Response

```
{
  "accessToken": "newlyGeneratedAccessToken"
}
```

**6. Best Practices**

- **Keep Access Tokens Short-lived**: Access tokens should have short expiration times for better security.

- **Store Refresh Tokens Securely**: Ensure that refresh tokens are stored in secure locations (e.g., HttpOnly cookies) to minimize risk.
- **Rotate Tokens**: Implement token rotation to ensure that refresh tokens cannot be used indefinitely.
- **Secure Your JWT Secret**: Protect your signing secret (SECRET_KEY) using environment variables and ensure it's rotated periodically.

**7. Future Improvements**

- **Implement Token Revocation**: Allow users to revoke refresh tokens, particularly when logging out.
- **Use Refresh Token Rotation**: Issue a new refresh token each time an access token is refreshed.
- **Support Multi-factor Authentication**: Add support for multi-factor authentication (MFA) for improved security during login.

.

---