# JWT and Refresh Token

**# Generating and Using Refresh Token for Secure Authentication**

## 1. Generating Refresh Token During Login

- **Client Request**: Client server ko login request bhejta hai.
- **Server Response**: Server do tokens generate karta hai:
  - **Access Token**: Short-lived (max 15 minutes validity).
  - **Refresh Token**: Long-lived (valid up to 6 months).
- **Token Storage**:
  - Access Token normal response ke saath return hota hai.
  - Refresh Token HTTP-only secure cookie me store hota hai, jo XSS attacks se bachata hai.

## 2. Access Token Expiry and Refresh Token Usage

- Jab **Access Token expire hota hai**, client automatically Refresh Token ka use karke ek **naya Access Token** generate karta hai.
- Refresh Token long-term stored hota hai aur sirf naye Access Token generate karne ke liye use hota hai.
- Ye system **security aur user experience** dono improve karta hai.

## 3. Security Benefits of Using Two Tokens

- **Access Token short-lived hota hai**, isliye agar compromise ho jaye toh impact limited hota hai.
- **Refresh Token long-lived hota hai**, par sirf naye Access Token generate karne ke liye use hota hai.
- Refresh Token ko **HTTP-only cookies** me store karne se security enhance hoti hai.
- Access Token ke expire hone ke baad bhi **session continuity** maintain hoti hai.

# Implementation via Coding

## Step 1: Create LoginResponseDTO

```
package com.springJourney.Week5Practice.DTOs;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class LoginResponseDTO {
    private Long id;
    private String accessToken;
    private String refreshToken;
}
```

}

## Step 2: Generate Access and Refresh Tokens in JWT Service

**i ) JWT Token →**

```
public String generateAccessToken(UserEntity userEntity){
    log.info("Attempting generateAccessToken method");
    String key = Jwts.builder()
            .subject(userEntity.getUserid().toString())
            .claim("email", userEntity.getEmail())
            .claim("roles", Set.of("Admin", "User"))
            .issuedAt(new Date())
            .expiration(new Date(System.currentTimeMillis() + 1000 * 600))
            .signWith(getSecretKey())
            .compact();
    log.info("Successfully generated access token with id: {} and email: {}",
                userEntity.getUserid(), userEntity.getEmail() );
    return key;
}
```

**ii) Refresh Token →**

```
public String generateRefreshToken(UserEntity userEntity){
    log.info("Attempting generateRefreshToken method");
    String key = Jwts.builder()
            .subject(userEntity.getUserid().toString())
            .issuedAt(new Date())
            .expiration(new Date(System.currentTimeMillis() + 1000 * 600 * 600))
            .signWith(getSecretKey())
            .compact();
    log.info("Successfully generated refresh token with id: {}", userEntity.getUserid());
    return key;
}
```

## Step 3: Use These Tokens in AuthServices

```
public LoginResponseDTO login(LoginDTO loginDTO) {
    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginDTO.getEmail(),
                    loginDTO.getPassword())
    );
    UserEntity userEntity = (UserEntity) authentication.getPrincipal();
    String accessToken = jwtServices.generateAccessToken(userEntity);
    String refreshToken = jwtServices.generateRefreshToken(userEntity);
    return new LoginResponseDTO(userEntity.getUserid(), accessToken, refreshToken);
}
```

### Step 4: Handle Login in AuthController

```
@PostMapping("/login")
public ResponseEntity<LoginResponseDTO> login(@RequestBody LoginDTO loginDTO,
                                              HttpServletRequest req,
                                              HttpServletResponse response)
 {
    LoginResponseDTO loginResponseDTO = authServices.login(loginDTO);
    Cookie cookie = new Cookie("refreshToken", loginResponseDTO.getRefreshToken());
    cookie.setHttpOnly(true);
    response.addCookie(cookie);
    return ResponseEntity.ok(loginResponseDTO);
}
```

### Step 5: Request New Access Token Using Refresh Token

```
@PostMapping("/refresh")
public ResponseEntity<LoginResponseDTO> refresh(HttpServletRequest request) {
    String refreshToken = Arrays.stream(request.getCookies())
            .filter(cookie -> "refreshToken".equals(cookie.getName()))
            .findFirst()
            .map(Cookie::getValue)
            .orElseThrow(() ->
                    new AuthenticationServiceException("Refresh token not found in Cookie"));
    LoginResponseDTO loginResponseDTO = authServices.refreshToken(refreshToken);
    return ResponseEntity.ok(loginResponseDTO);
}
```

### Step 6: Refresh Token Logic in AuthServices

```
public LoginResponseDTO refreshToken(String refreshToken) {
    Long userId = jwtServices.getUserIdFromToken(refreshToken);
    UserEntity userEntity = userService.findById(userId);
    String accessToken = jwtServices.generateAccessToken(userEntity);
    return new LoginResponseDTO(userEntity.getUserid(), accessToken, refreshToken);
}
```

## Final Summary

1. **Login**: Client login request bhejta hai, aur server Access Token aur Refresh Token generate karta hai.
2. **Token Storage**: Access Token response me return hota hai, aur Refresh Token HTTP-only cookie me store hota hai.
3. **Token Expiry Handling**: Agar Access Token expire ho jaye, toh Refresh Token se naya Access Token generate hota hai.
4. **Security**: Access Token short-lived hota hai, aur Refresh Token securely stored hota hai.

5. **Implementation**: Login request, token generation, secure storage, aur token renewal handle karne ke liye proper backend code likhna hota hai.

Ye pura process authentication system ko **secure aur efficient** banata hai, jisse unauthorized access prevent hota hai aur user experience smooth rehta hai.

By Aron