Lab6 solutions.

Group Members:

1.	Misghina Niguse	113198
2.	Filimon Isak	113074
3.	Daniel Tekie	111664

Criteria for generating refresh token.

The group members discussed and come up with the following workflow on how to generate a refresh token.

Workflow

Step-1
When the user logs in, the backend server generates two tokens: an access token and a refresh token. The access token has a short lifespan, say 15 minutes, and is used to access the resources on the server. The refresh token has a longer lifespan, say 30 days, and is used to renew the access token.

When the access token is generated, it's tied to the device or IP address that the user is using to log in. This information is stored in the access token.

Step-2

When the user makes a request to access a protected endpoint, the backend server validates the access token and checks if it's tied to the device or IP address that the user is currently using. If the access token is valid and matches the device or IP address, the user is granted access to the requested resource.

Step-3
If the user tries to access the resource from a different device or IP address, the access token is considered invalid, and the user is redirected to the login page.

Step-4
If the access token expires before the user logs out or tries to access a resource from a different device or IP address, the user needs to refresh the token using the refresh token.

Step-5
When the refresh token is used to generate a new access token, the new token is also tied to the device or IP address that the user is currently using.

Step-6

If the user logs out or the refresh token expires, all access tokens tied to that device or IP address are invalidated, and the user needs to log in again to generate new tokens.

This device-based approach adds an additional layer of security to the access token backend server, as it prevents unauthorized access from different devices or IP addresses.

The classes and methods for managing the access and refresh tokens can be developed as follows:

```
@Service
public class JwtTokenService {

 private static final String JWT_SECRET_KEY = "ourSecretKey";

    public String generateAccessToken(String device) {
        long currentTimeMillis = System.currentTimeMillis();
        Date issuedAt = new Date(currentTimeMillis);
        Date expiresAt = new Date(currentTimeMillis + 15 * 60 * 1000); // 15 minutes
        return Jwts.builder()
            .setIssuedAt(issuedAt)
            .setExpiration(expiresAt)
            .claim("device", device)
            .signWith(SignatureAlgorithm.HS256, JWT_SECRET_KEY)
            .compact();
    }
```

```
public String generateRefreshToken(String device) {
        long currentTimeMillis = System.currentTimeMillis();
        Date issuedAt = new Date(currentTimeMillis);
        Date expiresAt = new Date(currentTimeMillis + 30 * 24 * 60 * 60 * 1000); // 30 days
        return Jwts.builder()
            .setIssuedAt(issuedAt)
            .setExpiration(expiresAt)
            .claim("device", device)
            .signWith(SignatureAlgorithm.HS256, JWT_SECRET_KEY)
            .compact();
    }

    public Jws<Claims> parseAccessToken(String token, String device) throws JwtException {
        Jws<Claims> jws = Jwts.parser()
            .setSigningKey(JWT_SECRET_KEY)
            .parseClaimsJws(token);
        if (!jws.getBody().get("device").equals(device)) {
            throw new JwtException("Invalid device for token");
        }
        return jws;
    }
```