7.Generating JWT Tokens

JWT (JSON Web Token) is a compact and self-contained way for securely transmitting information between parties as a JSON object. In a Spring Boot application, JWT is commonly used to secure REST APIs. This guide explains how to generate a JWT token after a successful login.

1. Add Dependencies in pom.xml

To work with JWT, include the required dependencies.

Dependencies:

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
    <scope>compile</scope>
</dependency>
```

Explanation:

1. jjwt-api:
   - Provides the main API for working with JWTs (e.g., building and parsing tokens).
2. jjwt-impl:
   - Contains the implementation of the JWT library.
3. jjwt-jackson:
   - Integrates with the Jackson library to serialize/deserialize claims in the JWT.

2. Create JwtService Class

The JwtService class is responsible for generating JWT tokens.

Example:

```java
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

public class JwtService {
  public String generateToken(String username) {
    Map<String, Object> claims = new HashMap<>(); // Claims can include custom data (e.g., roles, permissions)
    claims.put("username", username); // Adding custom claim

    return Jwts.builder()
      .setClaims(claims) // Add claims to the token
      .setSubject(username) // Set the subject (e.g., the username)
      .setIssuedAt(new Date(System.currentTimeMillis())) // Current time as issue time
      .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 3)) // Token expiration time (3 hours)
      .signWith( getKey(),SignatureAlgorithm.HS256) // Sign the token with the secret key
      .compact(); // Generate the token
  }
}
```

Explanation:

1. Map<String, Object> claims:
   - A map to store additional information (claims) such as username, roles, or custom data.
2. setClaims(Map<String, Object>):
   - Adds custom claims (key-value pairs) to the token.
3. setSubject(String username):
   - Sets the username as the subject of the token, representing the authenticated user.
4. setIssuedAt(Date date):
   - Specifies when the token was issued.
5. setExpiration(Date date):
   - Sets the token's expiration time, calculated as the current time plus a duration (e.g., 3 hours in this case).
6. signWith(Key key, SignatureAlgorithm algorithm):
   - Signs the token using the specified algorithm (HS256 in this case) and a secret key.
7. compact():
   - Generates and returns the JWT token as a compact string.

3. Use JwtService in UserController

After a successful login, call the JwtService to generate the token.

Example:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    @Autowired
    private JwtService jwtService;

    @PostMapping("/login")
    public String login(@RequestBody User user) {
        // Authenticate user (logic not shown)
        boolean isAuthenticated = true; // Replace with actual authentication logic

        if (isAuthenticated) {
            return jwtService.generateToken(user.getUsername());
        } else {
            return "Login Failed";
        }
    }
}
```

Explanation:

1. @Autowired JwtService jwtService:
    - Injects the JwtService to use its generateToken() method.
2. Logic in login():
    - Upon successful authentication, calls jwtService.generateToken() with the username and returns the generated JWT.

4. Token Claims and Signing

Claims:

- Claims represent the payload data within the JWT.
- Common claims:
  - sub (subject): Represents the user.
  - iat (issued at): The timestamp of when the token was created.
  - exp (expiration): When the token will expire.

Signing:

- JWT tokens are signed with a secret key to ensure their integrity.
- In this example, the **HS256** algorithm is used, requiring a secure secret key.