12. JWT Summary

1.User Authentication (Login):

- **Endpoint**: /login
- **Process**:
  - User submits credentials (username/password) via POST request.
  - Credentials are verified using AuthenticationManager with a UsernamePasswordAuthenticationToken.
  - On successful authentication, a **JWT token** is generated.

2.JWT Token Generation:

- **Class**: JwtService
- **Token Construction**:
  - Claims (e.g., username, roles) are added using setClaims(Map<String, Object>).
  - The token includes metadata such as subject, issuedAt, and expiration.
  - The token is signed using a **HMAC-SHA256** algorithm with a secret key.
  - Output: A compact, self-contained token string.

3.Token Issuance:

- The token is returned to the client in the login response.
- **Client Responsibility**: Store the token securely (e.g., in localStorage or as an HTTP-only cookie).

4.Request with Token:

- For protected endpoints, the client includes the JWT in the Authorization header using the format: Authorization: Bearer <JWT>.

5.JWT Validation on API Requests:

- **Filter**: JwtFilter (extends OncePerRequestFilter)
- **Flow**:
  - Extract the Authorization header.

        - Validate the token signature using the secret key (Key object via Keys.hmacShaKeyFor()).
        - Decode claims using Jwts.parserBuilder().parseClaimsJws(token).getBody().
        - Check token validity:
            - **Signature**: Ensures the token hasn't been tampered with.
            - **Expiration**: Confirms the token is not expired using Claims.getExpiration().

6. **Authentication Context Update**:
    - If the token is valid:
        - Extract the username via extractUserName().
        - Load UserDetails from the user store (via UserDetailsService).
        - Create a UsernamePasswordAuthenticationToken and set it in SecurityContextHolder.
    - If invalid:
        - Deny access or return an unauthorized response.
7. **Security Filter Chain Configuration**:
    - **Session Policy**: SessionCreationPolicy.STATELESS (No server-side sessions).
    - **CSRF**: Disabled for token-based security.
    - **Filters**: Custom JwtFilter added before the UsernamePasswordAuthenticationFilter.
8. **Token Claims and Validation**:
    - Claims extracted (e.g., username, roles) using functional interfaces like Claims::getSubject.
    - Token is validated to ensure:
        - Subject matches authenticated user.
        - Token is not expired (using extractExpiration()).
9. **JWT Libraries and Key Management**:
    - **Library**: io.jsonwebtoken (JJWT).
    - **Key Management**:
        - Secret key dynamically generated or securely configured using environment variables.

        - Base64 encoding for portability, decoded for cryptographic operations.

10. **Post-Validation Request Flow**:
    - Once authenticated, the request proceeds to the controller.
    - Authorization checks are performed based on roles or permissions included in the token.

End-to-End Lifecycle

1. **Login** → User authenticated → JWT issued.
2. **Request** → JWT provided → Validated → User authenticated.
3. **Protected Resource Access** → Authorization ensured via claims.

This flow enables stateless, secure, and scalable API authentication using JWT.