10.Setting AuthToken In SecurityContext

1. Getting the Authorization Header

Example:

```
String authHeader = request.getHeader( "Authorization" );
```

Explanation:

- The Authorization header in HTTP requests is used to send the JWT.
- The JWT typically begins with the Bearer prefix, followed by the actual token.

2. Autowiring JwtService in JwtFilter

Example:

```
@Autowired
private JwtService jwtService;
```

Explanation:

- The JwtService is autowired into the JwtFilter to handle token-related operations such as extracting the username and validating the token.

3. Adding Logic in doFilterInternal

Example:

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain)
        throws ServletException, IOException {

    String authHeader = request.getHeader("Authorization");
    String token = null;
    String username = null;

    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        token = authHeader.substring(7);
        username = jwtService.extractUserName(token);
    }

    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        UserDetails userDetails =
            context.getBean(UserDetailsService.class).loadUserByUsername(username);

        if (jwtService.validateToken(token, userDetails)) {
            UsernamePasswordAuthenticationToken authentication = new
                    UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    }

    filterChain.doFilter(request, response);
}
```

Explanation:

1. **Extracting the Token**:
   - Checks if the Authorization header exists and starts with Bearer .
   - Removes the Bearer prefix to isolate the token.
2. **Extracting the Username**:
   - Calls the extractUserName method in JwtService to decode and retrieve the username from the token.
3. **Validating the Token**:
   - Ensures that:
     - The username from the token matches the UserDetails object.
     - The token is valid (e.g., it hasn't expired).
4. **Setting the Authentication**:
   - If the token is valid, creates a UsernamePasswordAuthenticationToken and sets it in the SecurityContext.

5. **Continuing the Filter Chain**:
    - Calls filterChain.doFilter() to allow further processing of the request.

4. extractUserName Method in JwtService

Example:

```
public String extractUserName(String token) {
    // Extract the username from the JWT token
}
```

Explanation:

- This method parses the JWT token to extract the username claim.
- Typically implemented using libraries like **JJWT** or **Nimbus JOSE**.

5. validateToken Method in JwtService

Example:

```
public boolean validateToken(String token, UserDetails userDetails) {
    // Logic to validate the token (e.g., checking claims, expiration, and signature)
    return true;
}
```

Explanation:

- Validates the token by:
    1. Ensuring the token's signature matches.
    2. Checking the expiration time.
    3. Verifying that the token's username matches the UserDetails object's username.

- Returns true if the token is valid, false otherwise.

Summary of the Workflow:

1. The client sends a request with a JWT in the Authorization header.
2. The JwtFilter extracts and validates the token.
3. If the token is valid, the SecurityContext is updated with authentication.
4. The request proceeds to the next stage in the filter chain.