Jwt Request Filter

Path: com.opus.backend.config.JwtRequestFilter.class

This class extends the **OncePerRequestFilter** class from the Spring framework. It's responsible for filtering incoming HTTP requests to validate and authenticate JWT (JSON Web Token) tokens in the "Authorization" header.

1. **@Component**: Marks the class as a Spring component, enabling Spring's component-scanning mechanism to automatically detect and instantiate it as a bean in the application context.
2. **public class JwtRequestFilter extends OncePerRequestFilter**: Declares the class **JwtRequestFilter** which extends **OncePerRequestFilter**. This filter is executed once per request and is used to handle JWT authentication.
3. **private final ObjectMapper mapper = new ObjectMapper();**: Creates an instance of the **ObjectMapper** class from the Jackson library, which is used for converting Java objects to JSON and vice versa.
4. **@Autowired**: The following three fields are annotated with **@Autowired**, indicating that Spring should automatically inject the appropriate beans for these dependencies:
   * **private JwtUserDetailsService jwtUserDetailsService;**: A service for retrieving user details.
   * **private JwtTokenUtil jwtTokenUtil;**: A utility class for handling JWT token operations.
   * **private UserDao userDao;**: A data access object (DAO) for retrieving and manipulating **DAOUser** objects.
5. **protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws ServletException, IOException**: This method is an implementation of the **doFilterInternal()** method from the **OncePerRequestFilter** class. It takes in three arguments:
   * **HttpServletRequest request**: The incoming HTTP request.
   * **HttpServletResponse response**: The HTTP response that will be sent back to the client.
   * **FilterChain chain**: The filter chain, which allows further processing of the request and response after the current filter finishes execution.

Inside the **doFilterInternal()** method, the following actions are performed:

a. Extract the "Authorization" header value from the request.

b. Check if the header value starts with "Bearer " and if so, extract the JWT token.

c. If a token is found, attempt to extract the username from the token and handle any exceptions that may occur.

d. If a username is found and the user is not already authenticated, load the user's details and validate the token.

e. If the token is valid, create a **UsernamePasswordAuthenticationToken** object and set it in the Spring Security context.

f. Call **chain.doFilter(request, response)** to allow further processing of the request and response by other filters in the chain.

Additionally, the code handles various error conditions, such as malformed or expired tokens and expired user subscriptions. In case of an error, an appropriate JSON error response is generated and sent back to the client.