**JWT Filter:**

In the provided code, \*\*servlets\*\* are part of the Java EE (Enterprise Edition) framework and are used to handle HTTP requests and responses in a web application. Specifically, in this Spring Boot application, the `JwtFilter` class extends `OncePerRequestFilter`, which is a Spring-provided filter that ensures the `doFilterInternal` method is executed \*\*once per request\*\*. Let’s break down the role of servlets and how they are used in this code.

---

### \*\*What Are Servlets?\*\*

1. \*\*Servlet Definition\*\*:

- A \*\*servlet\*\* is a Java class that handles HTTP requests and generates HTTP responses.

- It acts as a middle layer between the client (browser, mobile app, etc.) and the server (application logic, database, etc.).

- Servlets are part of the `javax.servlet` (or `jakarta.servlet` in newer versions) package.

2. \*\*Servlet Lifecycle\*\*:

- \*\*Initialization\*\*: The servlet is initialized when the application starts or when the first request is made.

- \*\*Request Handling\*\*: The servlet processes each incoming request and generates a response.

- \*\*Destruction\*\*: The servlet is destroyed when the application stops or the servlet is no longer needed.

3. \*\*Servlet in Spring Boot\*\*:

- In Spring Boot, servlets are abstracted away, and developers typically work with higher-level components like controllers, filters, and interceptors.

- However, servlets are still used under the hood to handle HTTP requests and responses.

---

### \*\*How Servlets Are Used in This Code\*\*

The `JwtFilter` class is a \*\*filter\*\* that intercepts every incoming HTTP request. Filters are part of the servlet API and are used to perform tasks like authentication, logging, or modifying requests/responses before they reach the controller.

#### \*\*Key Servlet Components in the Code\*\*

1. \*\*`HttpServletRequest`\*\*:

- Represents the HTTP request sent by the client.

- Provides methods to access request headers, parameters, body, etc.

- In this code, it is used to:

- Extract the `Authorization` header (`request.getHeader("Authorization")`).

- Pass the request along the filter chain (`filterChain.doFilter(request, response)`).

2. \*\*`HttpServletResponse`\*\*:

- Represents the HTTP response that will be sent back to the client.

- Provides methods to set response headers, status codes, body, etc.

- In this code, it is passed along the filter chain but not directly modified.

3. \*\*`FilterChain`\*\*:

- Represents a chain of filters that the request must pass through before reaching the controller.

- The `doFilter` method is called to pass the request and response to the next filter or the controller.

4. \*\*`OncePerRequestFilter`\*\*:

- A Spring-provided abstract class that ensures the `doFilterInternal` method is executed \*\*once per request\*\*.

- It simplifies the implementation of filters by handling the boilerplate code.

---

### \*\*Step-by-Step Flow of the Code\*\*

#### 1. \*\*Request Interception\*\*

- When a client sends an HTTP request to the server, the `JwtFilter` intercepts it before it reaches the controller.

#### 2. \*\*Extract Authorization Header\*\*

- The `Authorization` header is extracted from the request:

```java

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

```

- Example of an `Authorization` header:

```

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

```

#### 3. \*\*Check for Bearer Token\*\*

- The code checks if the `Authorization` header is present and starts with `Bearer `:

```java

if (authHeader == null || !authHeader.startsWith("Bearer ")) {

filterChain.doFilter(request, response);

return;

}

```

- If the header is missing or invalid, the request is passed along the filter chain without further processing.

#### 4. \*\*Extract and Validate Token\*\*

- The JWT token is extracted from the header:

```java

String token = authHeader.substring(7); // Remove "Bearer " prefix

```

- The username is extracted from the token:

```java

String userName = jwtUtil.extractUserName(token);

```

#### 5. \*\*Load User Details\*\*

- If the username is valid and no authentication is set in the `SecurityContextHolder`, the user details are loaded:

```java

UserDetails userDetails = userDetailsService.loadUserByUsername(userName);

```

#### 6. \*\*Validate Token\*\*

- The token is validated using the `jwtUtil`:

```java

if (userDetails != null && jwtUtil.validateToken(token)) {

```

#### 7. \*\*Set Authentication in SecurityContextHolder\*\*

- If the token is valid, an `Authentication` object is created and set in the `SecurityContextHolder`:

```java

UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities()

);

SecurityContextHolder.getContext().setAuthentication(authToken);

```

#### 8. \*\*Pass Request Along the Filter Chain\*\*

- Finally, the request is passed along the filter chain:

```java

filterChain.doFilter(request, response);

```

---

### \*\*Role of Servlets in This Code\*\*

1. \*\*Request Handling\*\*:

- The `HttpServletRequest` object provides access to the incoming request, including headers, parameters, and body.

2. \*\*Response Handling\*\*:

- The `HttpServletResponse` object allows the filter to modify the response before sending it back to the client.

3. \*\*Filter Chain\*\*:

- The `FilterChain` object ensures that the request passes through all filters in the chain before reaching the controller.

4. \*\*Security Context\*\*:

- The `SecurityContextHolder` is used to store the authenticated user's details, which can be accessed later in the application.

---

### \*\*Example Flow\*\*

#### \*\*Input (HTTP Request)\*\*:

```http

GET /api/resource HTTP/1.1

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

```

#### \*\*Steps\*\*:

1. The `JwtFilter` intercepts the request.

2. Extracts the token from the `Authorization` header.

3. Validates the token and extracts the username.

4. Loads the user details and sets the authentication in the `SecurityContextHolder`.

5. Passes the request to the next filter or controller.

#### \*\*Output\*\*:

- If the token is valid, the request proceeds to the controller with the authenticated user details.

- If the token is invalid, the request is rejected, and the client receives a `401 Unauthorized` response.

---

### \*\*Why Servlets Are Important Here\*\*

- \*\*Request/Response Handling\*\*: Servlets provide the foundation for handling HTTP requests and responses in Java web applications.

- \*\*Filters\*\*: Filters like `JwtFilter` rely on the servlet API to intercept and process requests before they reach the controller.

- \*\*Security\*\*: Servlets enable the implementation of security mechanisms like JWT authentication by providing access to request headers, parameters, and the ability to modify responses.

---

### \*\*Summary\*\*

- \*\*Servlets\*\* are the backbone of HTTP request/response handling in Java web applications.

- In this code, the `JwtFilter` uses servlet components (`HttpServletRequest`, `HttpServletResponse`, `FilterChain`) to:

1. Intercept requests.

2. Extract and validate JWT tokens.

3. Set authentication in the `SecurityContextHolder`.

4. Pass requests along the filter chain.

- Understanding servlets helps you debug and customize filters and other components in your Spring Boot application.