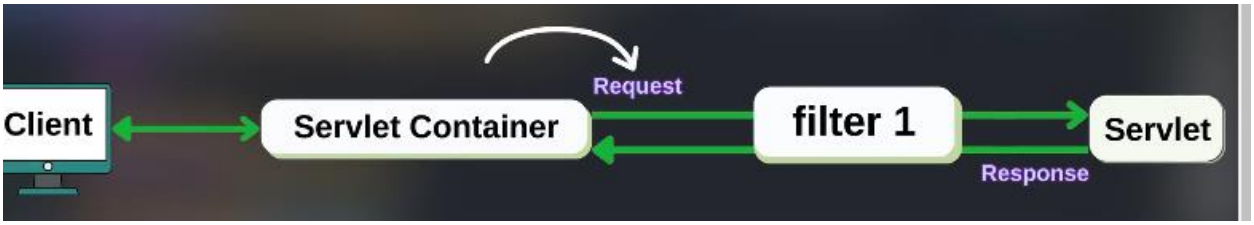
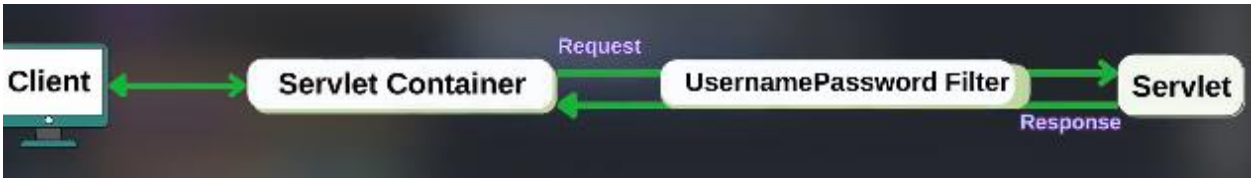
9.Creating A JWT Filter

We created the token, but on the server side we need to verify during sending the request.

By default, your Spring Security verifies using UsernamePasswordAuthentication. Add filter between that



By default spring security uses some filter



Now we add one more filter.

1. Adding a JWT Filter to the Security Configuration

Example:

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(request -> request
            .requestMatchers("/register", "/login").permitAll()
            .anyRequest().authenticated()
        )
        .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

Explanation:

- **SecurityFilterChain Bean**:
  - Configures the HTTP security settings for the application.
  - Allows specific routes (e.g., /register and /login) to be accessed without authentication while securing all other endpoints.
- **Session Management**:
  - Configures the application to be **stateless** (using SessionCreationPolicy.STATELESS), as JWT-based authentication doesn't rely on server-side session storage.
- **Adding JwtFilter**:
  - **addFilterBefore()** is used to add the custom JwtFilter to the security filter chain.
  - The filter is executed **before** the UsernamePasswordAuthenticationFilter, which handles basic username-password authentication.

2. Creating the JwtFilter Class

Example:

```
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;

public class JwtFilter extends OncePerRequestFilter {

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

    }
}
```

Explanation:

- **JwtFilter**:
    - Extends OncePerRequestFilter, ensuring that the filter is executed once per request.

3. Wiring the JwtFilter in SecurityConfig

Example:

```
@Autowired
private JwtFilter jwtFilter;
```

Explanation:

- **Dependency Injection**:
    - The JwtFilter is declared as a Spring-managed bean and injected into the SecurityConfig class using the @Autowired annotation.
    - This ensures that the JwtFilter is available and properly integrated into the Spring Security filter chain.

Key Concepts:

1. CSRF Disable:

- Disables Cross-Site Request Forgery (CSRF) protection. This is common in stateless APIs secured by tokens.

2. Session Management (Stateless):

- Configures the application to not use HTTP sessions, as JWT tokens are self-contained and hold all necessary authentication information.

3. Filter Chain:

- Filters are a core part of Spring Security, enabling the application to intercept HTTP requests and apply security logic (e.g., token validation).

4. JWT Validation:

- Ensures only authenticated and authorized users can access protected endpoints. Validation can include:
    - Checking the token's signature.
    - Decoding the token to extract claims.
    - Validating the token's expiration.