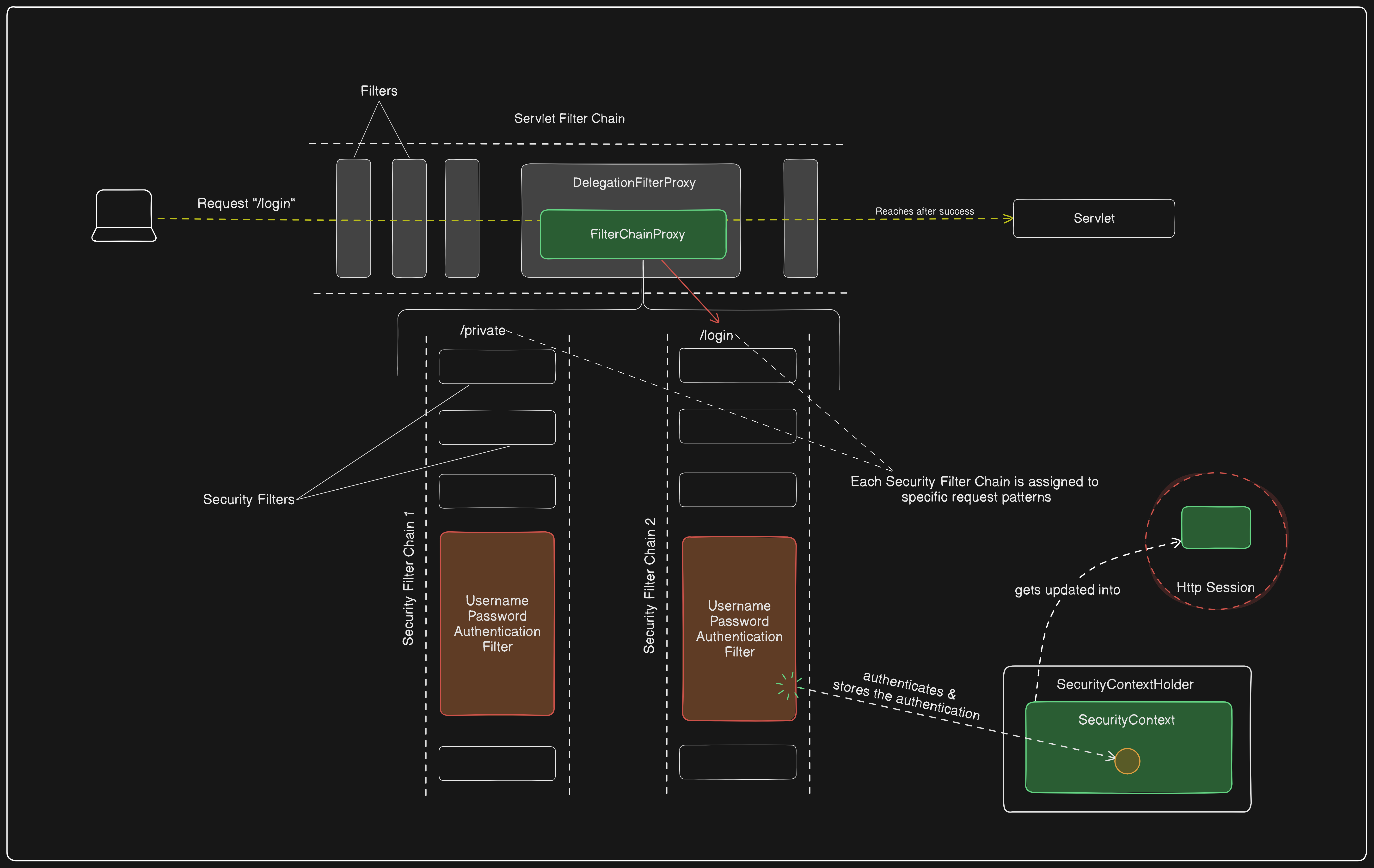
Spring Security :

Security Filter Chain:

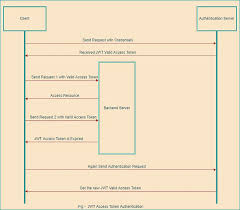
The SecurityFilterChain contains a list of Security Filters (Objects) each tailored to specific security requirements.

The request passes through all the security filters serially, if any filter fails to authenticate the request, an Unauthorized or Forbidden response is returned to the client.

If the request successfully passes through the filters, the request is then allowed to access the application resources.



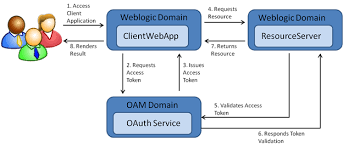
JWT authentication:



Instead of storing user credentials information somewhere on the server that can easily be accessed by attackers over the internet, we use JWT. When we use that, we maintain user secrets with different cryptographic algorithms and encode that to authenticate users.

* First, the user sends a request to the authentication server with credentials like the user name and password.
* Then, the authentication server will validate that information, and whatever information is provided by the user will be correct and successfully authenticated so the auth server can issue the JWT Valid Access Token to the user.
* Next, the user sends the first request to the backend server with a valid JWT Access Token, and the server will provide the requested resource to the user.

**OAuth2**



**Key Components of OAuth2**

* **Resource Owner**: The end user who owns the data the client application wants to access.
* **Client (Application)**: The application requesting access to the resource. In Spring Boot, this is your application.
* **Authorization Server**: Authenticates the user and issues an access token to the client. Examples include Google's OAuth2 service.
* **Resource Server**: Hosts the protected resources and verifies the access token provided by the client.

**How OAuth2 Works in Spring Boot**

In a Spring Boot application, OAuth2 can be integrated to handle both authentication and authorization. The framework relies on Spring Security, which provides robust mechanisms for securing applications.

Here’s how OAuth2 fits into the Spring Boot architecture:

1. **Client Registration**: You need to register the application with the OAuth2 provider (e.g., Google, GitHub) to obtain a client ID and client secret. These credentials are used to identify the application to the authorization server.
2. **OAuth2 Client Setup in Spring Boot**: In Spring Boot, configure the OAuth2 client settings in the application.properties or application.yml file. This includes specifying the client ID, client secret, authorization server URLs, and scopes of access. The Spring Security OAuth2 client automatically handles the redirection to the authorization server and manages the exchange of authorization codes for access tokens.
3. **Authentication Flow**:
   * When a user tries to access a secured resource, Spring Security intercepts the request and checks if the user is authenticated.
   * If the user is not authenticated, Spring Security redirects them to the OAuth2 provider's login page.
   * After successful authentication, the OAuth2 provider redirects the user back to the application with an authorization code.
   * The Spring Security OAuth2 client exchanges this code for an access token, which is then used to authenticate the user within the application.

* client-id and client-secret are your Google OAuth credentials.
* scope specifies the data you are requesting from Google.
* redirect-uri is where Google will send the user after successful authentication.

@Configuration

@EnableWebSecurity

**public** **class** **SecurityConfig** **extends** WebSecurityConfigurerAdapter {

@Override

**protected** void configure(HttpSecurity http) **throws** Exception {

http

.authorizeRequests()

.antMatchers("/", "/login\*\*").permitAll() *// Allow unauthenticated access to home and login pages*

.anyRequest().authenticated() *// Require authentication for all other requests*

.and()

.oauth2Login() *// Enable OAuth2 login*

.defaultSuccessUrl("/dashboard", **true**); *// Redirect to dashboard after successful login*

}

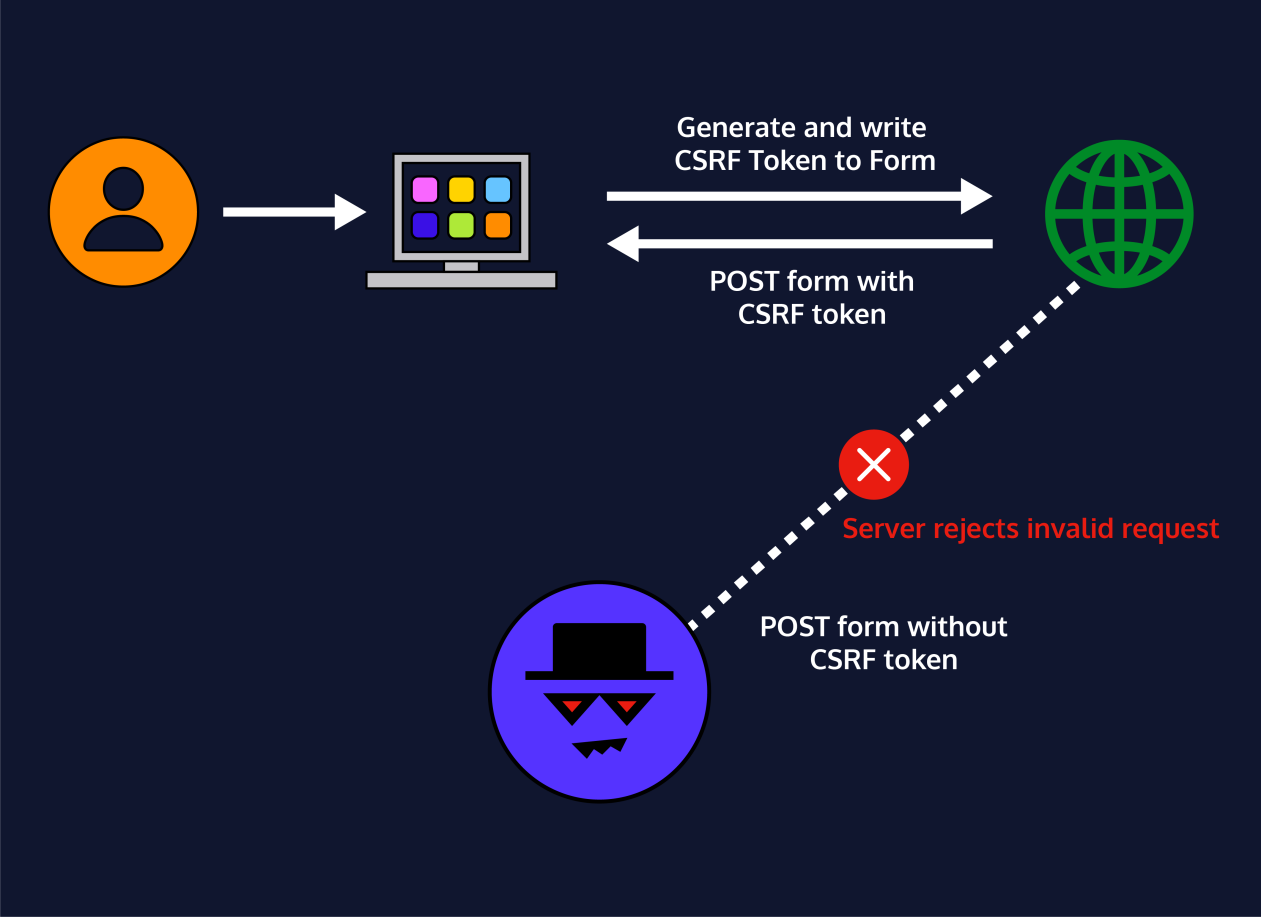
}

* **antMatchers("/", "/login\*\*").permitAll():** Allows unauthenticated access to the home and login pages. This ensures that users can access these pages without logging in.
* **anyRequest().authenticated():** Requires authentication for all other requests, ensuring that users need to be logged in to access other parts of the application.
* **oauth2Login():** Configures the application to use OAuth2 login.
* **defaultSuccessUrl("/dashboard", true):** Redirects users to the dashboard page after a successful login.

CSRF

### **Cross-Site Request Forgery**

Cross-Site Request Forgery is a serious vulnerability that results from poor session management. If the requests sent by an application aren’t unique, it’s possible for an attacker to craft a special request and send that to a user. If the user interacts with the crafted request, and sessions aren’t handled properly, an attacker may be able to assume the session identity of that user and carry out requests on their behalf.



Basic authentication :

Basic authentication is an HTTP authentication scheme that uses a username and password sent in an Authorization header, which is then Base64 encoded. While it's a simple method, it is considered less secure because Base64 encoding is easily reversible, so basic authentication should always be used over a secure connection like [HTTPS](https://www.google.com/search?sca_esv=6b9acc621195338b&sxsrf=AE3TifPbdM_8Q-Puduh7r_O_uk5iWKD3TQ:1756901313937&q=HTTPS&sa=X&ved=2ahUKEwj6sOO1x7yPAxX_MlkFHc5BGqsQxccNegQIJBAB&mstk=AUtExfBegITXLSenxxpkLlmZCKI97m_BqVhXPALCczBEcQ07Lg_JtOfe8ISDlC4B6P8XFmIza0KFK5bn9T1SuLUb_-dik6ukrg52BTT-UZd7j7fpNvC8BXUvfu08vjMAGi7z6GNjrRzYrIvm_3fb8WkhQYAAnv18ussOVVIJoKD2Y6mXunY&csui=3" \t "https://www.google.com/_blank) or [TLS](https://www.google.com/search?sca_esv=6b9acc621195338b&sxsrf=AE3TifPbdM_8Q-Puduh7r_O_uk5iWKD3TQ:1756901313937&q=TLS&sa=X&ved=2ahUKEwj6sOO1x7yPAxX_MlkFHc5BGqsQxccNegQIJBAC&mstk=AUtExfBegITXLSenxxpkLlmZCKI97m_BqVhXPALCczBEcQ07Lg_JtOfe8ISDlC4B6P8XFmIza0KFK5bn9T1SuLUb_-dik6ukrg52BTT-UZd7j7fpNvC8BXUvfu08vjMAGi7z6GNjrRzYrIvm_3fb8WkhQYAAnv18ussOVVIJoKD2Y6mXunY&csui=3" \t "https://www.google.com/_blank)

