Spring Security :

**Internal Flow of Spring Security (DAO Authentication)**

1. **Initial Request:**

* When a client hits a protected API for the first time, the request is intercepted by the Security Filter Chain (SFC).
* The SFC consists of multiple filters and is configured using a security configuration class.

1. **Filter Chain Processing:**

* The Security Filter Chain checks if the incoming request is targeting a secured resource.
* If the resource is secured, the user is redirected to the login page (custom or default).

1. **Login and Authentication:**

* The user enters their username and password and submits the login form.
* The SFC captures the login data and creates an Authentication object, which is passed to the Authentication Manager (AM).

1. **User Details Retrieval:**

* The AM calls the loadUserByUsername() method from the UserDetailsService implementation.
* This method fetches the user details from the database and returns a UserDetails object.

1. **Authentication Provider Selection:**

* The AM uses an AuthenticationProvider to validate the credentials.
* In this case, we use the DAO Authentication Provider, suitable for database-based authentication.
* We create a DAO Authentication Provider bean and register it with the AM.

1. **Credential Verification:**

* The AM passes the Authentication object and the UserDetails object to the DAO Authentication Provider.
* The provider compares the credentials.
* If they match, it returns a valid Authentication object to the AM.

1. **Security Context Setup:**

* The AM passes the valid Authentication object to the SFC.
* The SFC creates a SecurityContext and stores the Authentication object inside it using the SecurityContextHolder.
* A session object is created, containing a Session ID and the Authentication object.
* This Session ID is sent back to the user and stored in their browser cookies.

1. **Subsequent Requests:**

* On future requests, the browser sends the Session ID.
* The filters check if the session is valid.
* If valid, the corresponding SecurityContext is restored, and the user is granted access to the API.

**Invalid User Scenario:**

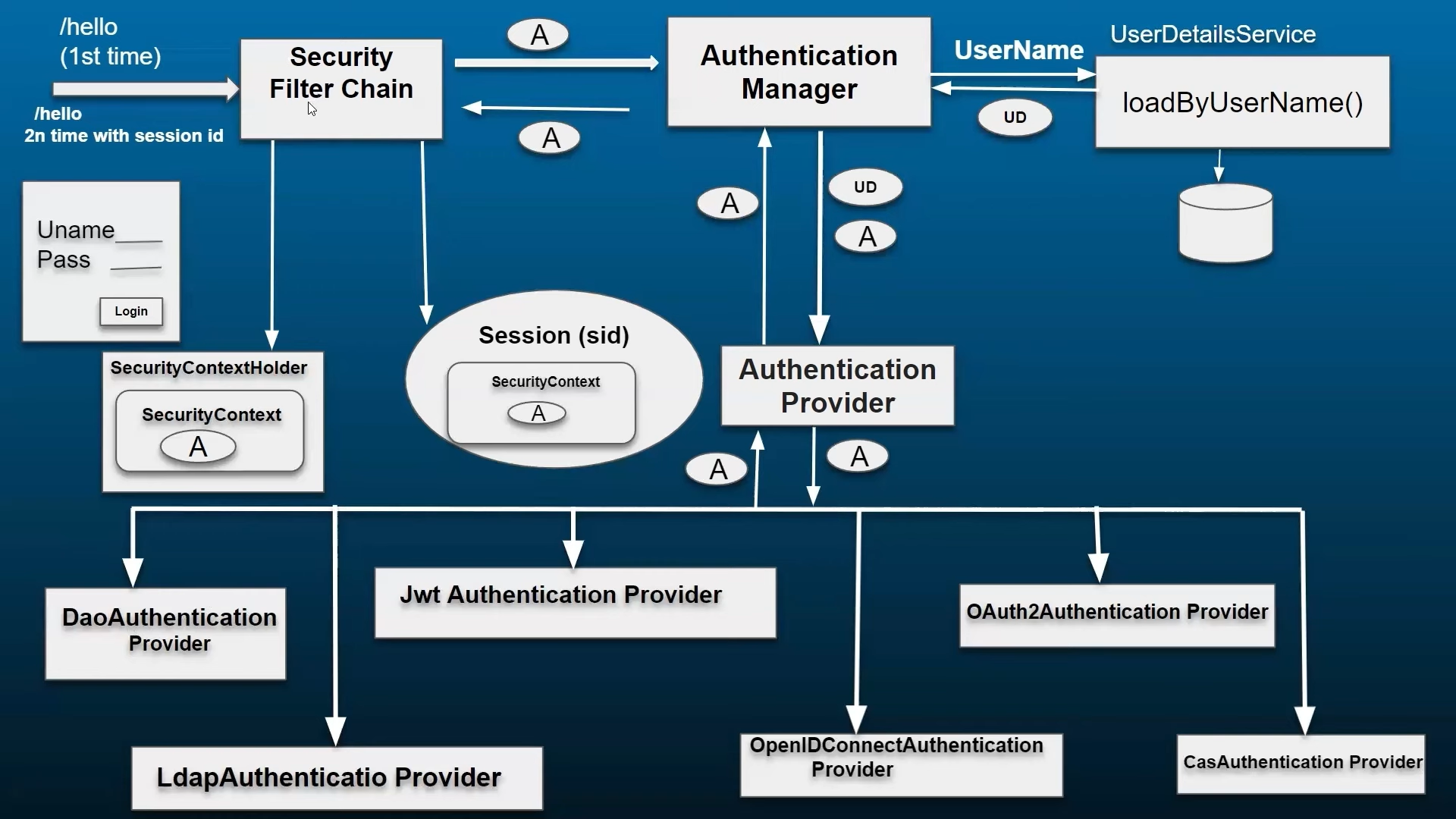
* If the username is not found in the database, the AM throws a UsernameNotFoundException.
* The SFC handles this exception and redirects the user back to the login page.

**Invalid Password Scenario:**

* If the username is correct but the password is invalid, the DAO Authentication Provider throws a BadCredentialsException.
* The SFC catches this and again redirects the user to the login page with an appropriate error message.

**Disadvantages :**

1. Tightly Coupled with UserDetailsService:
2. Limited to Username/Password Authentication (Scalability Issues)
3. Not Suitable for Stateless Applications (e.g., REST APIs):
4. Poor Fit for Microservices:
5. Password Handling Responsibility: You must configure a PasswordEncoder correctly.
6. Limited Customization without Overriding Core Classes:



**JWT Token with Spring Security – Enterprise Application Flow**

**Overview:**

JWT (JSON Web Token) is used to securely transfer user identity information between client and server. In an enterprise app using Spring Security, JWT helps secure REST APIs by verifying identity and maintaining sessionless authentication.

**JWT Structure:**

A JWT token has 3 parts:

* Header: Algorithm type (e.g., HS256)
* Payload: Claims like subject (username), expiration time, user ID, etc.
* Signature: Created using a secret key and the specified algorithm.

**Authentication Flow:**

1. Authentication API (/authenticate) – Not Secured

* The client sends username and password to /authenticate.
* Spring Security Configuration allows this API without a token.

2. Request Intercepted by Spring Security Filter Chain (SFC)

The configuration defines:

* Which APIs are secured (require token)
* Which APIs are not secured (like /authenticate)

3. AuthenticationController (AC)

Developer implements this controller.

It performs:

1. Creates an Authentication object (UsernamePasswordAuthenticationToken) with the provided credentials.
2. Passes it to AuthenticationManager (AM).

4. AuthenticationManager (AM)

Uses AuthenticationProvider (AP) to validate credentials.

Steps:

1. AM fetches user details from the UserDetailsService (UDS) using username.
2. UDS calls DAO layer to get user info.
3. AP validates username and password.
4. If valid, returns authenticated Authentication object.

5. Back to AuthenticationController (AC)

AC receives the authenticated object.

AC then calls JWT Utility (JwtUtil):

* Passes username (or user details)
* JwtUtil generates a JWT token (encoded with secret key)

AC returns this token to the client.

**Using JWT Token to Access Secured APIs**

6. Client Accesses a Secured API

* Sends JWT token in Authorization header:

Authorization: Bearer <token>

7. Request Intercepted Again by Spring Security

* If the API is secured, request passes through JWT Filter.

8. JWT Filter Logic

Extracts token from the header.

Calls JwtUtil:

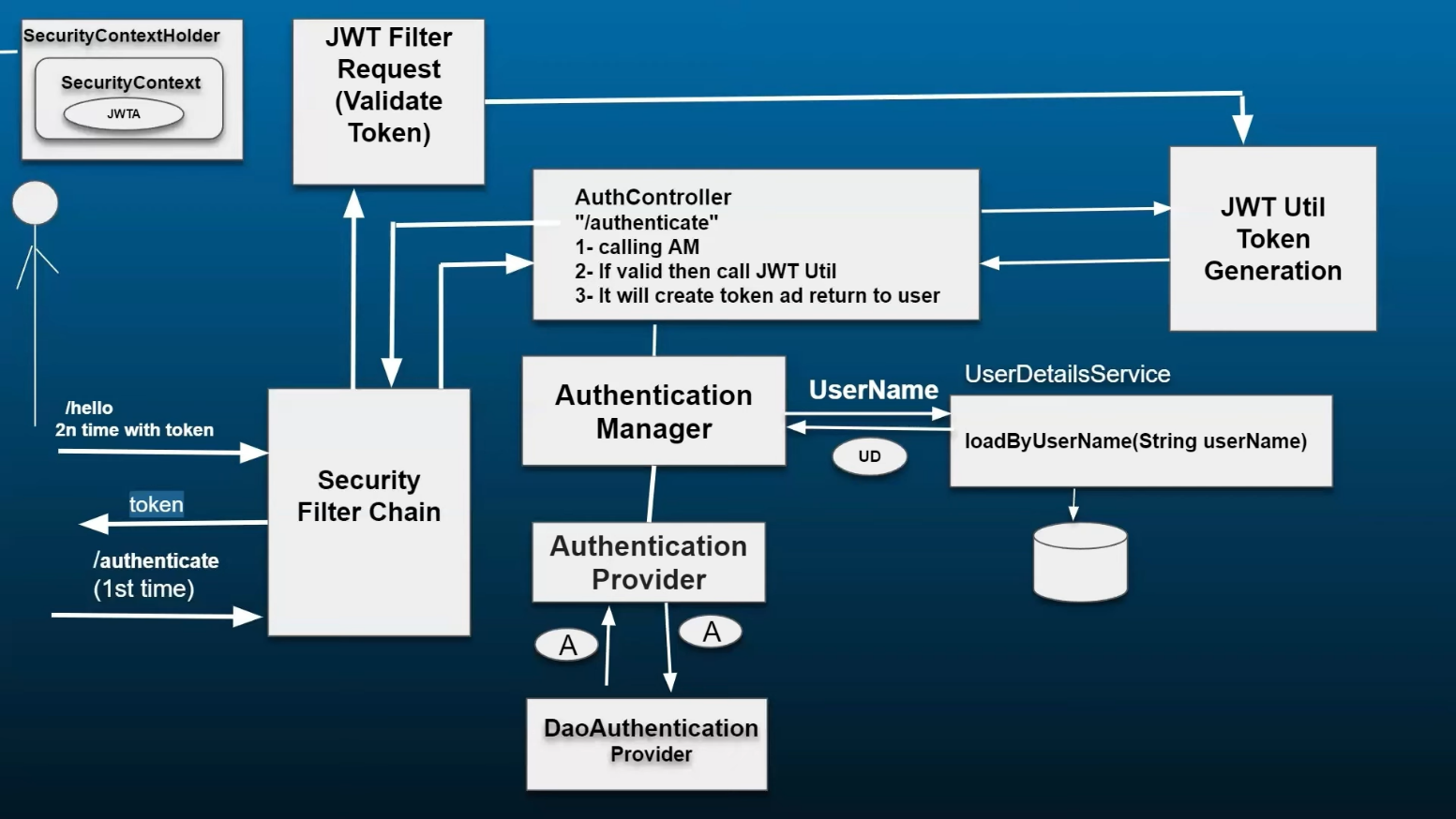
* Validates the token (checks signature, expiration)
* Decodes the token to extract user information

If valid:

* Creates a UsernamePasswordAuthenticationToken using extracted user info
* Sets it in SecurityContextHolder
* The request is then allowed to proceed to the controller

**JWT Utility (JwtUtil) Responsibilities**

* Generate JWT Token (on authentication)
* Validate JWT Token (on subsequent requests)
* Decode payload to extract user details



**Basic Level**

1. **What is Spring Security?**  
   Spring Security is a powerful and customizable authentication and access-control framework for Java applications, especially Spring-based apps. It handles authentication, authorization, password encoding, CSRF protection, session management, etc.
2. **How is Spring Security different from JAAS or container-managed security?**  
   Spring Security is more flexible, easily testable, and integrates seamlessly with Spring. Unlike JAAS or container-managed security, it supports method-level security and custom authentication flows.
3. **What are the main features of Spring Security?**
   * Authentication and Authorization
   * Password Encoding
   * Protection against CSRF, Session Fixation
   * OAuth2, OpenID Connect
   * Method-level security
   * Integration with LDAP, JWT, and more
4. **What is the default authentication mechanism in Spring Security?**  
   By default, Spring Security uses HTTP Basic authentication or form-based login with in-memory users.
5. **How do you enable Spring Security in a Spring Boot application?**  
   Add the dependency:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

1. **What is the purpose of the WebSecurityConfigurerAdapter class?**  
   This was the main class for customizing security configurations. As of Spring Security 6, it’s replaced by lambda-based SecurityFilterChain.
2. **What are the common annotations used in Spring Security?**
   * @EnableWebSecurity: Enables Spring Security config.
   * @PreAuthorize: Adds pre-authorization to methods.
   * @Secured: Declares method access rules using roles.
3. **What is the difference between authentication and authorization?**
   * Authentication: Verifying who the user is (login).
   * Authorization: Verifying what the user is allowed to do (permissions).
4. **What is the use of the UserDetails and UserDetailsService interface?**  
   UserDetails defines the core user info.  
   UserDetailsService fetches user info from a custom source (DB, LDAP, etc.).
5. **How do you define in-memory authentication in Spring Security?**

@Bean

public InMemoryUserDetailsManager userDetailsService() {

UserDetails user = User.withDefaultPasswordEncoder()

.username("user")

.password("password")

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

**Intermediate Level**

1. **How do you secure REST APIs using Spring Security?**

* Use JWT or Basic Auth for stateless APIs.
* Configure SecurityFilterChain to allow/deny specific endpoints.
* Disable CSRF for stateless services.

1. **What is method-level security? How do you enable it?**  
   Annotate your class with @EnableGlobalMethodSecurity(prePostEnabled = true) and use:

* @PreAuthorize("hasRole('ADMIN')")
* @PostAuthorize(...)

1. **How can you customize the login page in Spring Security?**

http.formLogin()

.loginPage("/custom-login")

.defaultSuccessUrl("/home");

1. **What is the use of PasswordEncoder?**  
   It encodes passwords and checks them securely. Examples:

* BCryptPasswordEncoder
* Pbkdf2PasswordEncoder

1. **What is CSRF? How does Spring Security handle it?**  
   CSRF (Cross-Site Request Forgery) is a web attack.  
   Spring Security auto-generates a CSRF token and requires it in non-GET requests (e.g., POST, PUT).
2. **How does Spring Security handle sessions and session fixation attacks?**  
   It invalidates the old session and creates a new one post-login to prevent session fixation.
3. **How do you configure roles and authorities in Spring Security?**  
   Use .roles("ADMIN") in memory or fetch from DB using UserDetailsService.
4. **What are filters in Spring Security?**  
   Filters are classes that inspect HTTP requests before they reach controllers. SecurityFilterChain defines their order.
5. **How to exclude specific URLs from Spring Security filters?**

http.authorizeRequests()

.antMatchers("/public/\*\*").permitAll();

1. **What is the order of filters in the Spring Security filter chain?**

* SecurityContextPersistenceFilter
* UsernamePasswordAuthenticationFilter
* BasicAuthenticationFilter
* ExceptionTranslationFilter
* FilterSecurityInterceptor, etc.

**Advanced Level**

1. **What is OAuth2 and how is it integrated in Spring Security?**  
   OAuth2 is an authorization framework.  
   Spring Security offers spring-security-oauth2-client and spring-security-oauth2-resource-server to support OAuth2 login and token verification.
2. **What is JWT and how can you implement JWT-based authentication?**  
   JWT (JSON Web Token) is a stateless, compact way to transmit user info.  
   Common flow:

* Authenticate → generate JWT → send to client
* Client sends JWT → validate using OncePerRequestFilter

1. **How does Spring Security integrate with Spring Boot's auto-configuration?**  
   With spring-boot-starter-security, many defaults are configured automatically, such as login page, password encoding, and filters.
2. **How do you implement a custom authentication provider?**

* Implement AuthenticationProvider
* Override authenticate() method
* Register using AuthenticationManagerBuilder

1. **What is the difference between GrantedAuthority and Role?**

* Role is a high-level group (e.g., ADMIN).
* Authority is a specific permission (e.g., READ\_PRIVILEGES).

1. **How do you handle security in microservices using Spring Security?**  
   Use JWT or OAuth2 for stateless authentication.  
   Combine with Spring Cloud Gateway + centralized authentication.
2. **How can you integrate Spring Security with LDAP?**  
   Use LdapAuthenticationProvider and configure UserDetailsContextMapper for mapping LDAP attributes to Spring roles.
3. **Explain the stateless authentication concept in Spring Security.**  
   In stateless auth, no server-side session is stored. The client sends a token (like JWT) with every request. The server validates and proceeds.
4. **How do you secure WebSockets with Spring Security?**

* Use AbstractSecurityWebSocketMessageBrokerConfigurer
* Enable SimpMessageType access rules
* Integrate HTTP session with WebSocket handshake

1. **How do you prevent brute force attacks using Spring Security?**

* Track failed login attempts
* Lock account after N attempts
* Add CAPTCHA after failed attempts
* Use rate-limiting libraries like Bucket4j