Spring JWT Security

Topics:

1. What are Security Attacks?
2. Internal Working of Spring Security
3. Authentication vs Authorization
4. Default Behavior of Spring Security
5. Core Components of Spring Security
6. Configuring Security Filter Chain
7. What is JWT?
8. Authenticating Requests Using JWT
9. JWTAuthFilter Control Flow (Step-by-Step)
10. Exception Handling in Spring Security

By Nayeem Shaik

24/7/25

Spring Security Fundamentals:-

Security Attacks:-

1. Cross-Site Request Forgery (CSRF)

→ An attack where a logged-in User is tricked into Performing Actions without knowing (like transferring money).

→ The web application Trusts the user's brower and executes the Malicious request.

Ex:- A banking website without CSRF protection → attacker sends a hidden transfer request → Unknowingly Transfers money.

How to Prevent?

1. CSRF Tokens → Generates a Unique, unpredictable token for each user session.
2. Stateless with JWT → Use JWT authentication instead of Session-based Authentication.

Note:-

"CSRF exploits the trust a site has in the user's browser, while XSS exploits the trust a User has in a Site".

2. Cross-Site Scripting (XSS)

→ An attacker injects malicious Scripts (JavaScript) into web pages, affecting other users.

→ Can steal Cookies, hijack sessions, or deface websites.

How to Prevent?

1. Input Validation & Sanitization → Check all user inputs and remove harmful Code.

Note:-

"Stored XSS is more dangerous than reflected XSS because it is Saved on the Server and affects multiple users".

3. SQL Injection.

→ An attacker inserts malicious SQL code into input fields to manipulate database queries.

→ Can expose (or) modify Sensitive data.

How to Prevent?

1. Use Prepared Statements/ Parameterized Queries.
2. Use ORM Frameworks like Hibernate instead of direct SQL.
3. Validate & Sanitize Inputs.

Note:- Ex:- SELECT * FROM users WHERE username = '' OR '1'='1';

"SQL injection happens when User input is directly Concatenated into SQL queries without Validation."

"

CSRF → Targets actions by tricking the browser.

XSS → Injects Scripts to steal (or) manipulate data. "

25/7/25

Internal working of Spring Security.

Spring Security is a powerful framework in Spring for Authentication (who you are) and Authorization (what you Can Access).

→ Protects applications from unauthorized Access.

→ we Can add just adding dependency To Our Project.

* Spring Boot auto-configurationes Security with sensible defaults using the "WebSecurityConfiguration" class.

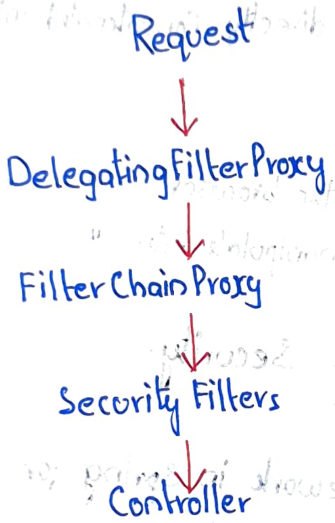
	Authentication	Authorization
Meaning	→ Verifying the identity of a User	→ Checking what an authenticated user Can (or) Cannot do.
How	→ Usually via Username/Password (or) Tokens.	→ Based on roles/ Permissions.
Example	→ Logging into Gmail using Credentials	→ whether you Can read emails (or) delete them.

Internal Working of Spring Security:-

1. SecurityFilterAutoConfiguration → Registers a DelegatingFilterProxy named SpringSecurityFilterChain.

2. DelegatingFilterProxy → Delegates the request to a FilterChainProxy.

3. FilterChainProxy → Uses a SecurityFilterChain to execute multiple Security Filters in a specific order (like authentication, CSRF Protection, etc).



Default Behaviour of Spring Security:-

1. Creates "SpringSecurityFilterChain" to handle all requests.
2. HTTP Basic Authentication is enabled by default.
3. Default Login Page is automatically generated.
4. Default User Credentials:-

   Username → user

   Password → Printed in the console at startup

5. Password encryption → stored using Bcrypt hashing.
6. Logout Features → Enabled Automatically.
7. CSRF Protection Enabled by default.
8. Session Fixation Protection Prevents session hijacking.

Internal Flow

1. Client Request → Sent to the server.

2. DelegatingFilterProxy → Intercepts the request.

3. FilterChainProxy → Decides which filters to apply.

4. Authentication Filter → Validates credentials.

5. Authorization Filter → Checks roles/Permissions.

6. Controller Execution → if allowed, the request reaches the Controller.

7. Response Returned → if not allowed, Spring Security returns 401.

28/7/25

Core Spring Security Components.

1. UserDetails

→ The UserDetails interface represents a User in the Spring Security framework.

→ it provides methods to get user information such as Username, password / Authorities.

Purpose → To encapsulate user information, including Authentication and Authorization details.

Implementation → You can use it to extend your UserEntity.

2. UserDetailsService

→ The UserDetailsService interface is a Core Component in Spring Security that is used to retrieve user-related data.

→ it has a single method : loadUserByUsername.

Purpose → To fetch user details from a datasource based on the username.

Implementation → You typically implement this interface to load user details, such as username, Password, and roles, from your own user repository.

3. InMemoryUserDetailsManager

→ The InMemoryUserDetailsManager is a Spring Security provided implementation of UserDetailsService that stores user information in memory

Purpose → To store user details in memory, typically for testing (or) small Applications

→ You define users directly in the Configuration.

4. PasswordEncoder

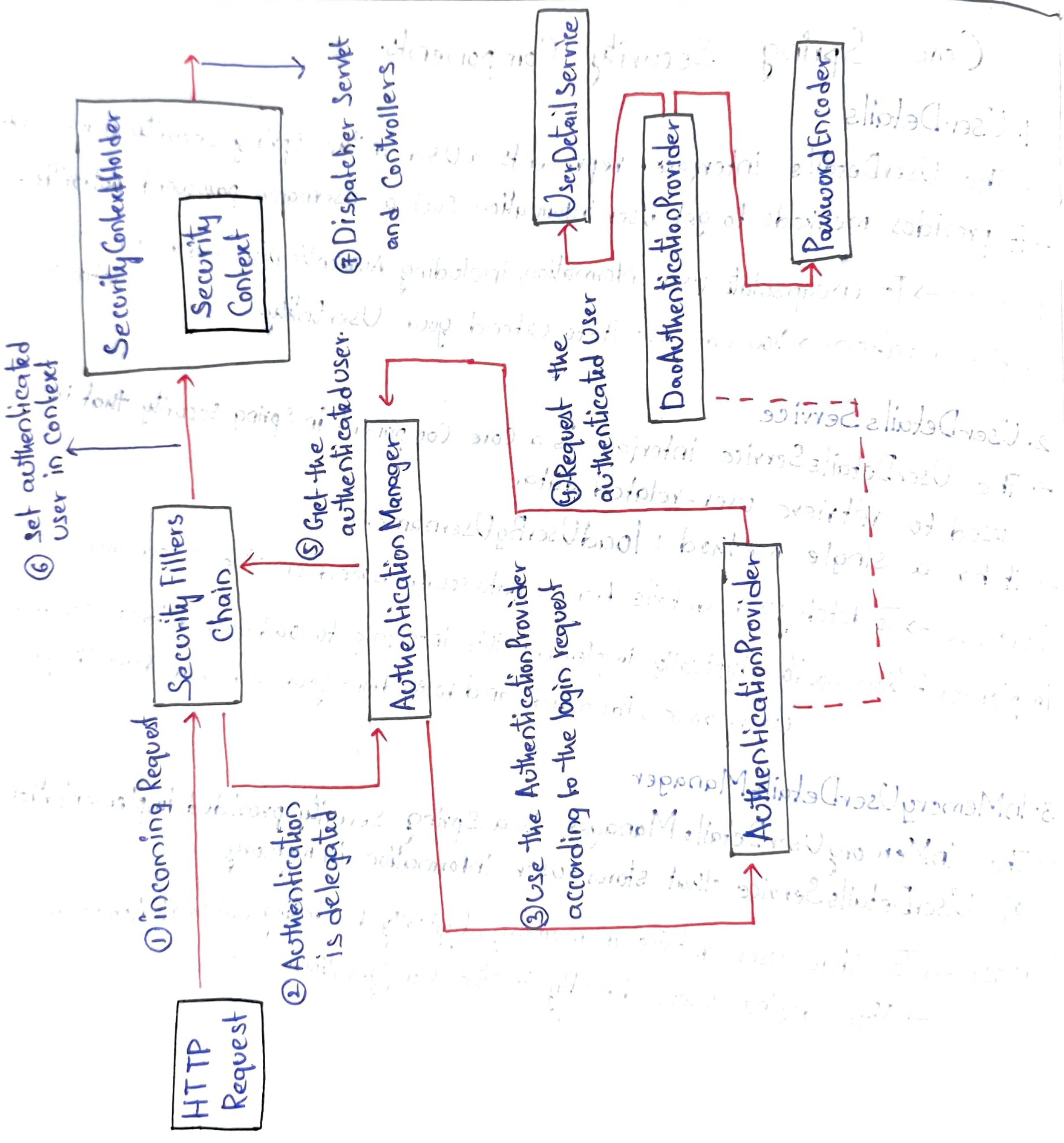
→ The PasswordEncoder interface is used for encoding and validating Passwords.

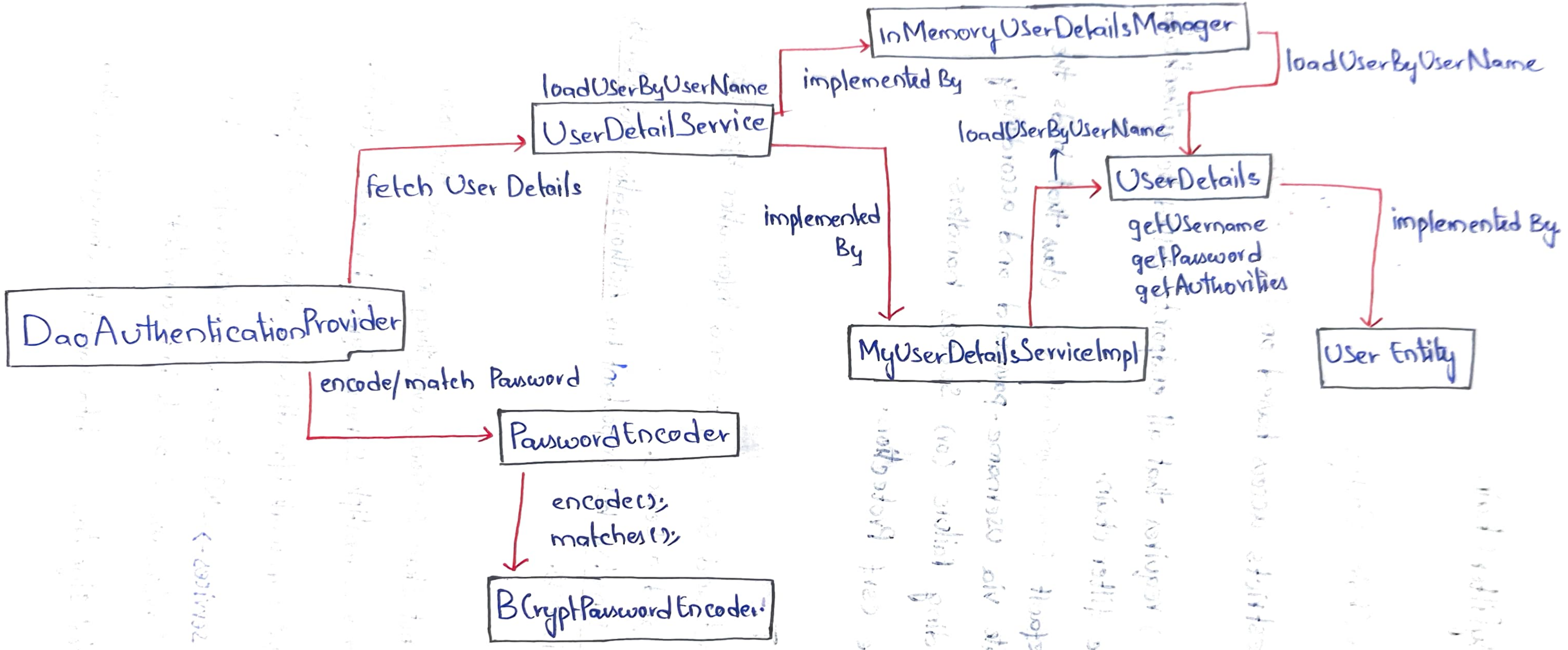
→ it has methods for encoding raw passwords and Matching encoded Passwords.

Purpose → To Security hash passwords before storing them and to verify hashed Passwords during authentication.

Common Implementations:-

1. BCryptPasswordEncoder.
2. Pbkdf2PasswordEncoder.
3. SCryptPasswordEncoder.


HTTP Request
① incoming Request
Security Filters Chain
② Authentication is delegated
AuthenticationManager
③ Use the AuthenticationProvider according to the login request
AuthenticationProvider
④ Request the authenticated User
DaoAuthenticationProvider
UserDetail Service
PasswordEncoder
⑤ Get the authenticated user
⑥ set authenticated user in context
SecurityContextHolder
Security Context
⑦ Dispatcher Servlet and Controllers


InMemoryUserDetailsManager
loadUserByUserName
loadUserByUserName
UserDetailService
implemented By
loadUserByUserName
fetch User Details
UserDetails
implemented By
getUsername
getPassword
getAuthorities
DaoAuthenticationProvider
MyUserDetailsServiceImpl
User Entity
encode/match Password
PasswordEncoder
encode();
matches();
BCryptPasswordEncoder

30/7/25

Configuring SecurityFilterChain

Default SecurityFilterChain Config:-

- authorizeRequests() restricts access based on RequestMatcher implements.
- ~~authoriz~~ authenticated() requires that all endpoints called be authenticated before Proceeding in the filter chain.
- formLogin() calls the default FormLoginConfigurer class that loads the login Page to authenticate via username-password and accordingly redirects to Corresponding failure (or) Success handlers.
- csrf() to cofigure the csrf Protection.

Understanding JWT:-

1. JWT is a Small, Compact and Safe way to send information b/w Two Parties (usually a client and a Server).
2. This info is usually not sensitive but important for Authorization and identity (like user ID, roles, Permissions).

Why Use JWT?

Stateless → Server does not store sessions, everything is inside the Token.

Scalable → Easily works across multiple Servers (great for large Systems).

Cross-domain Support → JWT Can work b/w different domains (or) Systems.

Decentralized systems/Microservices → JWT is best for microservices where Services are Separate but Connected.

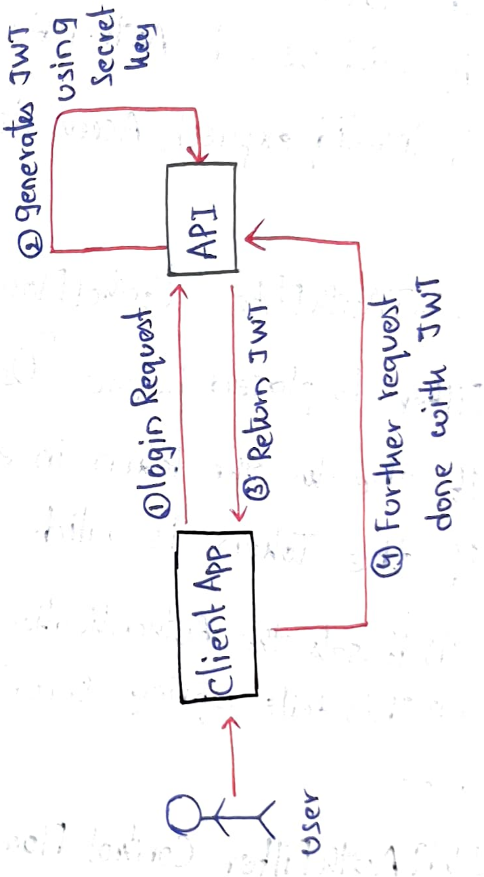
Highly Secure → Uses digital signature so data Cannot be changed or faked.

Structure of JWT:-

1. Header – Info about the algorithm
2. Payload – Actual data like user ID, roles etc.
3. Signature – Digital Signatures to make sure Token isn't Tampered.

JWT Creation Flow:-

1. User logs in with username/password.
2. if valid, Server Creates a JWT with:-
   - * User info
   - * Timestamp
   - * Expiry Time
3. JWT is sent to the Client (browser).
4. Client stores it (in local Storage (or) Cookies).


② Generates JWT using Secret key
API
① login Request
③ Return JWT
④ Further request done with JWT
Client App
User

JWT Verification:-

Every time the user makes a request:-

1. Client sends the JWT in the request header.
2. Server verifies:-
   - * is the signature valid?
   - * is the token expired?
   - * Does the Token Contain required roles?
3. if valid, Access is granted.

JWT Dependencies in SpringBoot:-

To Use JWT in SpringBoot.

1. Add required dependencies (like jjwt, Spring-Security, etc.).
2. Configure SecurityFilterChain and Custom Filters.
3. Use AuthenticationManager to verify login.
4. Generate JWT on Succesful login.
5. Intercept requests and validate JWT before Proceeding.

1/8/25

Authenticating requests using JWT:-

* JWT Authentication Workflow in SpringBoot

1. Customer Filter (e.g., JWTAuthFilter) intercepts the requests.
2. it extracts the token from Authorization:- Bearer < token > header.
3. The Token is validated (signature, expiration, etc.).
4. if valid, user is Authenticated and request proceeds.
5. if invalid/expired, Access is denied (401 Unauthorized).

*JWTAuthFilter Control Flow (How Spring Security Handles it).

1. Filter is placed before "UsernamePasswordAuthenticationFilter".
2. it checks the token in each incoming request.
3. if the Token is Valid.
   (i) it sets the Authentication object in the Security Context.
   (ii) This tells Spring Security that the user is Authenticated.

2/8/25

JWTAuthFilter Control Flow Explained.

1. HTTP Request → Client sends request.
2. Security Filters → Filters decide if Authentication is needed.
3. JwtAuthFilter → Validates JWT for Secured requests.
4. Login Controller → Handles login and issues token.
5. AuthenticationManager → Authenticates Credentials.
6. SecurityContextHolder → Stores the authenticated user.
7. DispatcherServlet → Routes to Controller.
8. Response → Returns data (or) token.

"In Spring Security with JWT, all requests first go through filters. For Secured APIs, the JwtAuthFilter checks and validates the JWT token from the header. if valid, it adds the user to the Security Context, and the request continues. For login requests, the Credentials are verified, and a JwT token is generated and returned".

Spring Security Exception Handling:-

① Authentication Exception:-

→ These exceptions Occur when user fails to log in (or) Session is invalid.

Common Causes:-

- ✱ Wrong username / Password.
- ✱ Expired account / session.
- ✱ Missing Credentials.

→ Use HTTP Status Code:- 401 UNAUTHORIZED.

Common Exceptions:-

1. AccountExpiredException → User Account is expired.
2. BadCredentialsException → Wrong Username (or) Password.
3. CredentialsExpiredException → Password expired.
4. AuthenticationCredentialsNotFoundException → No authentication details Provided.
5. SessionAuthenticationException → Session-related failure.

② JwtException (Jwt Token Specific):-

Common Exceptions:-

1. ExpiredJwtException → Token has expired.
2. MalformedJwt Exception → Token is wrongly structured (or) tampered.
3. Signature Exception → Token Signature is invalid
4. UnsupportedJwtException → Token format not Supported.
5. IllegalArgumentException → Token is null/empty (or) incorrect argument passed.