**What is spring security?**

**Ans: -** Spring Security is a powerful and highly customizable framework that focuses on providing authentication and authorization to Spring-based applications.

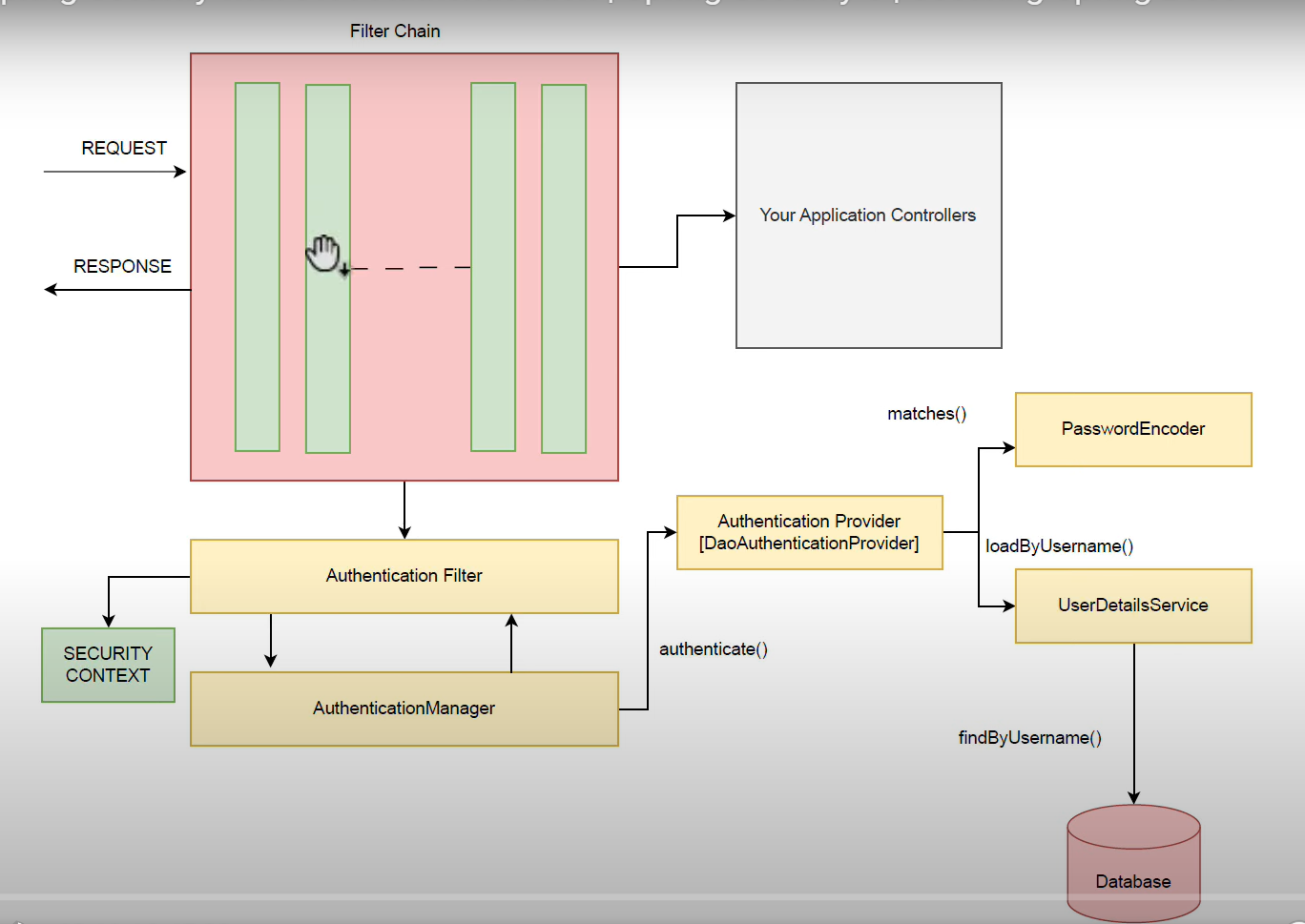
**What is authentication?**

**Ans:** - Authentication is verifying the user using the username and password.

**What is authorization?**

**Ans: -** After Authentication what user can accruable from the web site.

**How Spring Security Works.**

****

**What is filter chain.**

In Spring Security, a filter chain is a sequence of filters that are applied to each incoming HTTP request. These filters work together to perform various security-related tasks, such as authentication, authorization, and session management.

**Implementation**

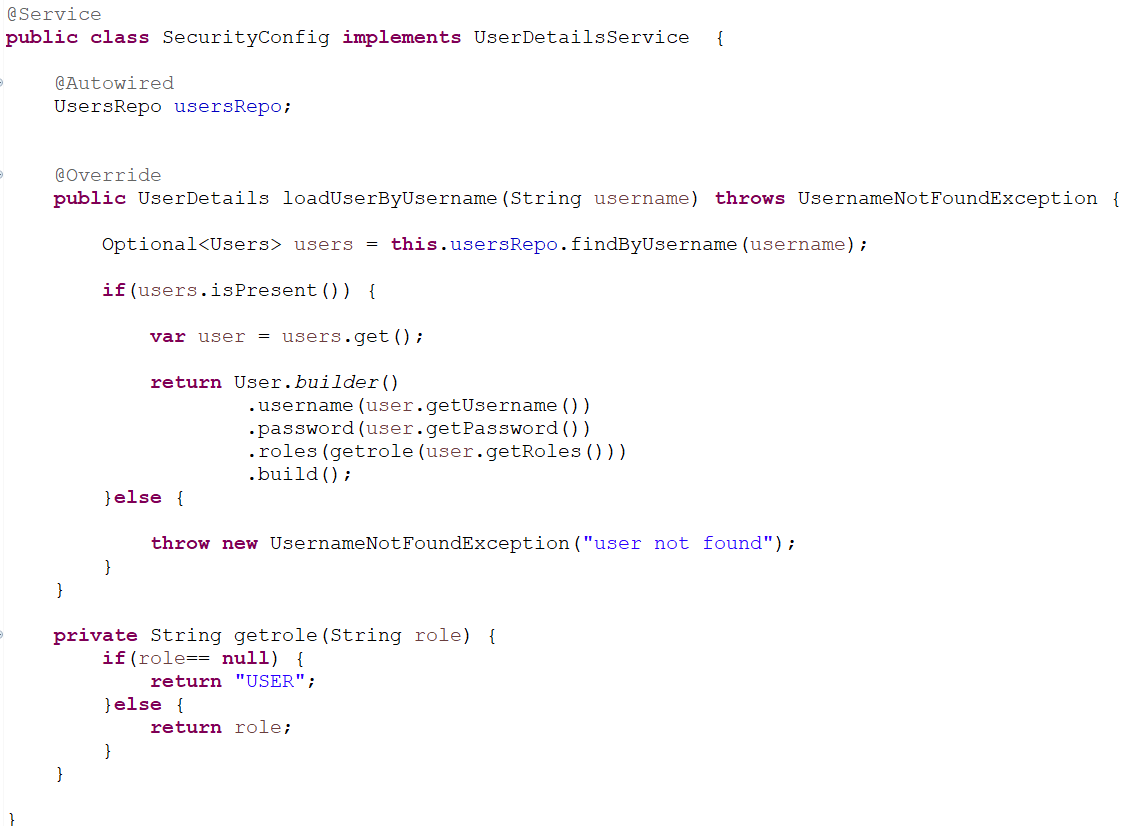
* Add dependency of spring security.
* Go to application propertyes
  + Add
    - Spring.secutiry.user.name=
    - Spring.security.user.password=

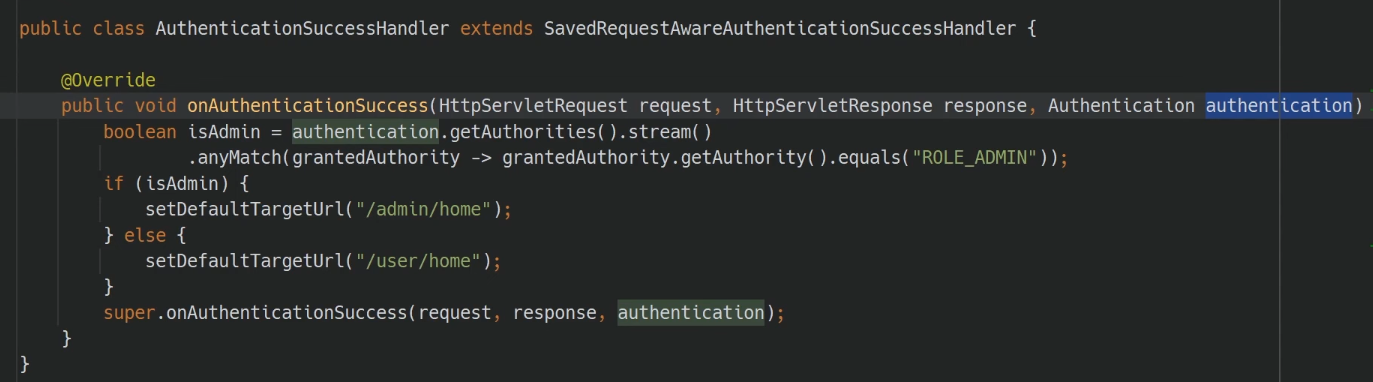
**@PreAuthorize(“hasRole(‘USER’)”)**: Is user to check the authorization before execution of a method.

**@EnableWebSecurity:** @EnableWebSecurity tell the spring this is security class and enable the security to the web applications.

**@Configuration:** Tags the class as a source of bean definitions for the application context.

**@EnableMethodSecurity**: It will tell to the spring there is an annotation of **@PreAuthorize.**

****

****

****

****

**What Authentication Provider Can Do?**

* In Spring Security, the Authentication Provider is a core interface that plays a crucial role in the authentication process. It is responsible for performing the actual authentication logic and producing an authentication object if the authentication is successful.
* Can throw exceptions like BadCredentialsException, AccountExpiredException, etc., if authentication fails.

**JWT Authentication & Authorization with Spring Security**

**YouTube link: -** <https://youtu.be/HYBRBkYtpeo?si=qTJJdihkZileyv_9>

* Spring Boot JWT authentication and authorization is a popular way to secure REST APIs. It uses JSON Web Tokens (JWT) to authenticate users and authorize their access to specific resources.

Three Dependency required for JWT web security.

* Jjwt-impl
* Jjwt-api
* Jjwt-jackson

**What is JWT web token: -** An open standard that securely transmits information between parties as a JSON object.

**what is a secret key in JWT and what it will do**

In JSON Web Tokens (JWT), a **secret key** is a cryptographic key used to sign and verify the authenticity of the token. Here's a breakdown of what it does:

**Purpose of the Secret Key in JWT:**

1. **Signing the Token**: When a JWT is created, it is typically signed using either a symmetric key (HMAC algorithms like HS256) or an asymmetric key pair (RSA or ECDSA algorithms like RS256 or ES256). If a symmetric algorithm like HS256 is used, the secret key is shared between the party issuing the token and the party verifying the token.
2. **Verification of the Token**: When a JWT is received by the server, the server uses the same secret key to verify that the token has not been tampered with. This process ensures the integrity of the token. If the signature matches, it indicates that the token is authentic and was issued by a trusted party.

**How it Works:**

* **Creation**: When a server generates a JWT, it creates the payload (the data, like user ID or roles), encodes it, and then signs it using the secret key. The signature ensures that the token cannot be modified without knowing the secret key.
* **Validation**: When the JWT is sent back to the server (e.g., via an HTTP request), the server decodes the token and verifies the signature using the secret key. If the signature is valid, the server trusts the data in the token.

**Example:**

In the case of **HS256** (HMAC with SHA-256):

* **Secret key**: my\_secret\_key\_123
* The JWT signature is created using this key.
* When the token is sent back to the server, the server will use the same my\_secret\_key\_123 to verify the token’s authenticity.

**What Happens if the Secret Key is Leaked:**

If the secret key is compromised, an attacker can forge valid JWTs, potentially gaining unauthorized access to resources or impersonating users. Therefore, keeping the secret key secure is crucial.

**Summary:**

* The **secret key** is used to ensure the **authenticity** and **integrity** of JWTs.
* It is involved in the **signing** (when the token is issued) and **verification** (when the token is validated) processes.
* A leaked secret key can compromise the security of your JWT system.

**What is Authentication Manager and what it will do: -**   
  
The Authentication Manager is a core interface in the Spring Security framework. It is responsible for handling authentication requests and returning an authentication object if the authentication is successful. Here's an overview of what it does and how it works:

**Authentication Manager:**

* Typically calls the authenticate method of one or more Authentication Provider instances.
* If the first Authentication Provider cannot authenticate, it moves on to the next provider in the list.
* If none of the providers can authenticate the user, it will throw an Authentication Exception.

**1. Authentication Requests Handling:**

* The Authentication Manager processes authentication requests. These requests typically contain credentials such as a username and password.
* It takes an authentication object as input, which contains the user's credentials.

**2. Authentication Providers:**

* The Authentication Manager delegates the actual authentication process to one or more Authentication Provider instances. Each Authentication Provider can handle a specific type of authentication (e.g., username/password, token-based, etc.).
* If one of the Authentication Providers successfully authenticates the user, it returns a fully populated Authentication object.

**3. Success and Failure Handling:**

* If authentication is successful, the Authentication Manager returns an authentication object containing details about the authenticated user (e.g., username, roles, etc.).
* If none of the Authentication Providers can authenticate the user, the Authentication Manager throws an Authentication Exception.

**4. Common Implementations:**

* “Provider Manager”: The most commonly used implementation of Authentication Manager. It holds a list of authenticationProviders and tries each one in turn until authentication is successful or all providers are exhausted.

**5. Use in Applications:**

* Authentication Manager is typically used in web applications to authenticate users during the login process.
* It can be configured with multiple authentication providers, allowing for flexible authentication mechanisms (e.g., database-backed authentication, LDAP, OAuth2, etc.).
* Often integrated with security filters like UsernamePasswordAuthenticationFilter to intercept login requests and delegate them to the AuthenticationManager.

****

**When to use authentication Manager**?  
  
Authentication Manager is used in various scenarios where you need to manage and verify the authentication of users within an application. Below are some common use cases:

**1. Custom Authentication Logic:**

* When you need to implement custom authentication logic that goes beyond the default username and password checks provided by Spring Security.
* For example, you might want to authenticate users against a third-party system, custom database queries, or a specific token-based authentication method.

**2. Custom Login Pages and Authentication Flows:**

* If you're building a custom login page or flow, where you need to manually authenticate a user, you would use Authentication Manager.
* This is particularly common in Single Page Applications (SPAs) using Angular or React, where the login process might involve an API call to the backend.

**How to create secret key:** -



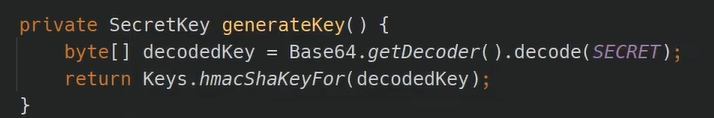
* Create a class in the test and write a test case and run the above code it will give secret key in the console.

**Steps to implementation: -**

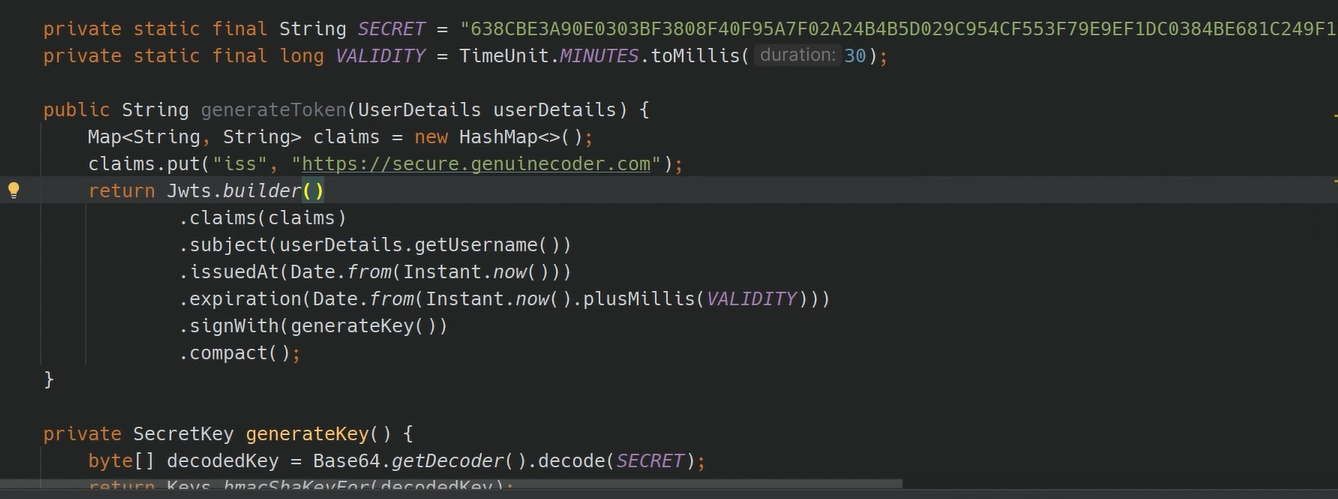
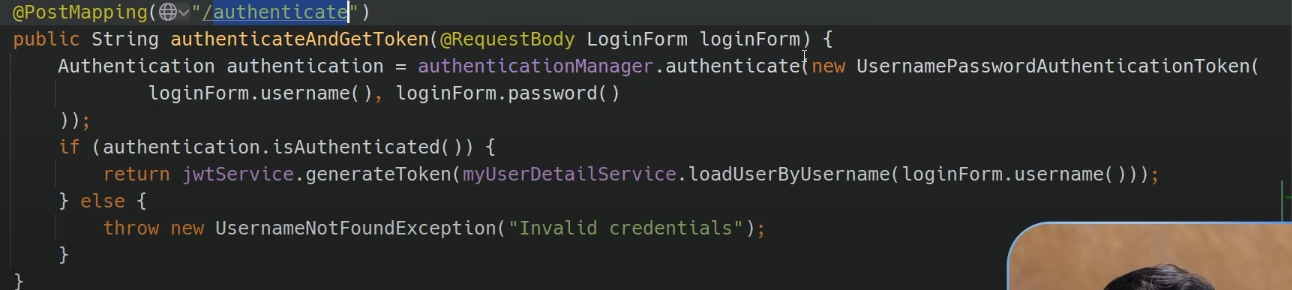
1. Store the secret key in a variable has a final key
2. Create a token with return type is string and userDetails in parameters.
   * **Set the values**
     + **Subject()**: - subject() menace for username or token name.
     + **issuedAt()**: - issuedAt() menace Token started time.
     + **Expiration()**: - expiration() menace Token expiration time.
     + **signWith()**: - signWith() menace secret key.
       - * **To generate the super key**

First, we must convert encoded format into decoded format, the result in byte array.

**Key.hmacShaKeyFor** is used to convert byte array in to secret key.



* + - **claims: -** claims menace adding extra data to the token.
  + Finaly convert the token in the string using **compact().**

1. ****
2. Calling the generated token when the login is required.  
     
   ****