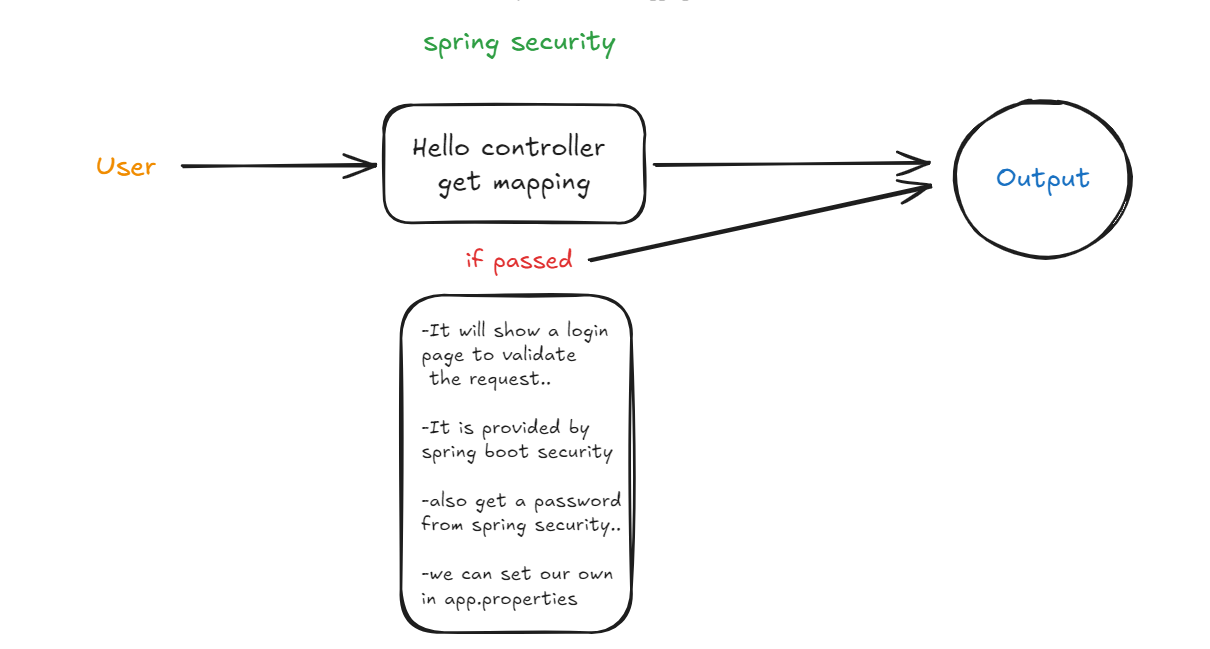
**Authentication using Spring Security-**

-This document outlines the design and implementation details for building a JWT-based authentication system in Spring Boot. The system aims to provide secure login and protect APIs by ensuring that only authenticated users can access protected resources.

**1) Spring Security Getting Started**

-added 3 dependencies 1) spring web, 2) spring security, 3) spring dev tools

-created a one get endpoint and exposed it..

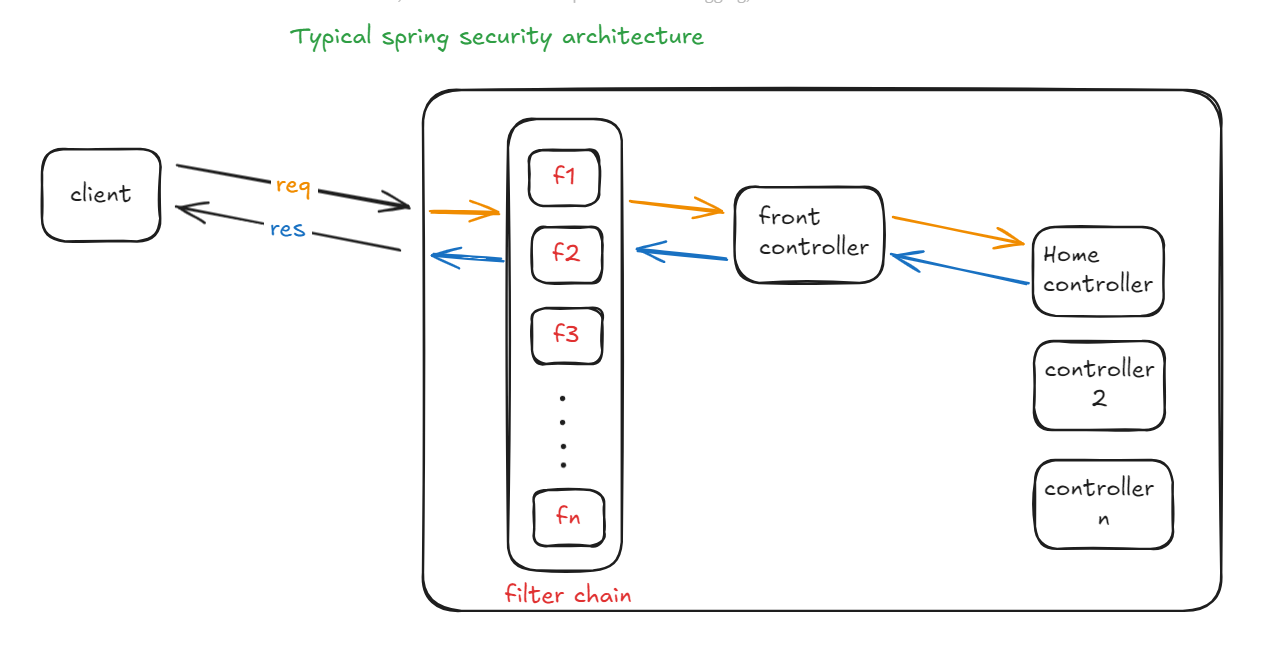


-spring security gives us a login and logout form when we hit any resource on server..

-as shown in the figure when we add security dependency then by default springboot provides us built in security..

**2) Setting our own username and password**

-before going to setup own loginid and password need to understand the spring security architecture-



-Before applying any username password by our side it is set by username password authentication filter.

-in our case one of the filter checking that user is logged in or not if not then shows a login form..

-Now in the application.properties file set our own username and password..

Eg.. spring.security.user.username = \*\*\*\*\*

Spring.security.user.password = \*\*\*\*\*

-That is how we can set our own username and password for spring security..

**3) Applying security using CSRF Token**

**-**why Cross-Site Request Forgery (CSRF) token

-CSRF (Cross-Site Request Forgery) is a type of attack where a malicious website tricks a user (who is authenticated on another site) into making unwanted requests to that site on their behalf. To prevent this, CSRF tokens are used.

A diagram of a software program

Description automatically generated

Fig: without csrf token

-Protection Against Forged Requests: When a user is logged into a site, they have an authenticated session, and browsers automatically include their authentication cookies with every request. A CSRF attack takes advantage of this by tricking the user into unknowingly submitting a form or making a request to the site they're logged into. The CSRF token ensures that the request is coming from the legitimate source (the actual user interacting with the application) and not from a third-party site.

-That is why csrf token comes into the picture and used to handle request and response to the server.

-With each request need to send \_csrf token to validate the request.

-spring security provides feature to implement the csrf request with tokens for every request.

-generated csrf token using HttpServletRequest request

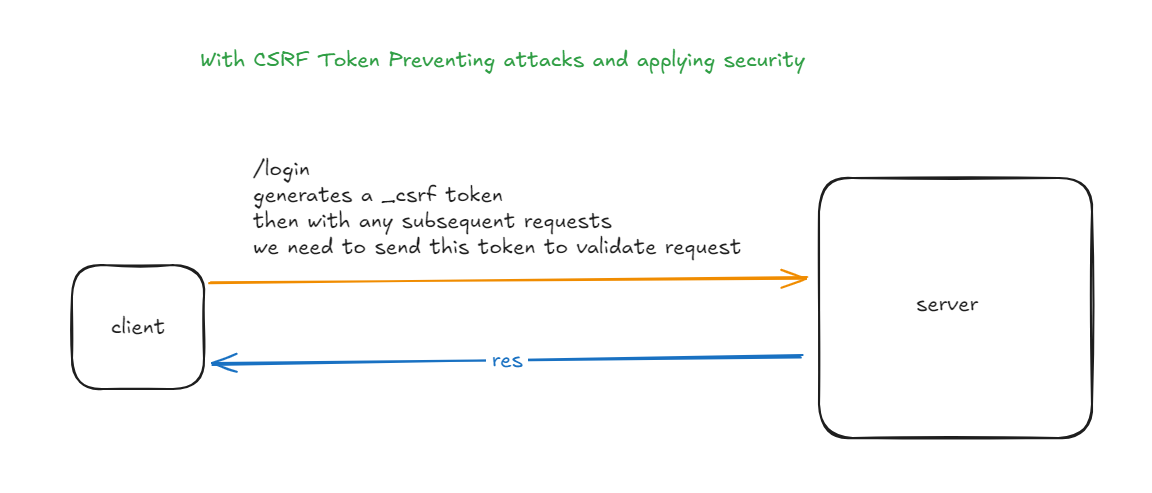


Fig : with csrf token

**4) Spring security custom configurations**

-Now need to update a configuration that spring security provides us for the custom implementations like jwt authentication.

-following configurations are done in security config file.

- Disabled default CSRF protection to bypass security for development purposes.

- Enabled basic authentication with custom username and password stored in application properties.

- Configured session management to be stateless every request brings new session.

- Ensured all requests are authenticated.

-This section configures Spring Security for a stateless, secured application. It disables CSRF protection (since it's likely using a token-based system like JWT), enforces authentication for all requests, uses basic authentication, and ensures that sessions are not stored (stateless sessions).

**5) Spring Security User Authentication with Database Integration**

-Added dependencies like JPA, MySQL connector.

-I don’t have MySQL server installed on my system only have workbench.. I used docker I pulled MySQL server image from the docker hub and run a instance of MySQL server on my system and connected it to the workbench.

**docker run --name mysql-container -e MYSQL\_ROOT\_PASSWORD=rootpassword -e MYSQL\_DATABASE=yourdb -p 3306:3306 -d mysql:latest**

-Later connected with spring boot using jpa and MySQL connector.

Flow 🡪

1. User Login:

- When a user tries to log in, Spring Security uses the configured `AuthenticationProvider` to authenticate the user.

- The `MyUserDetailsService` is called, which loads the user from the database using `UserRepo`.

- If the user exists, the details are passed to Spring Security for validation. If the password matches (stored in plain text here), the user is authenticated.

2. Database Interaction:

- The user details (username and password) are stored in a `Users` table in the database.

- The repository `UserRepo` is responsible for querying the database and retrieving the user based on the username provided during login.

A diagram of a system

Description automatically generated

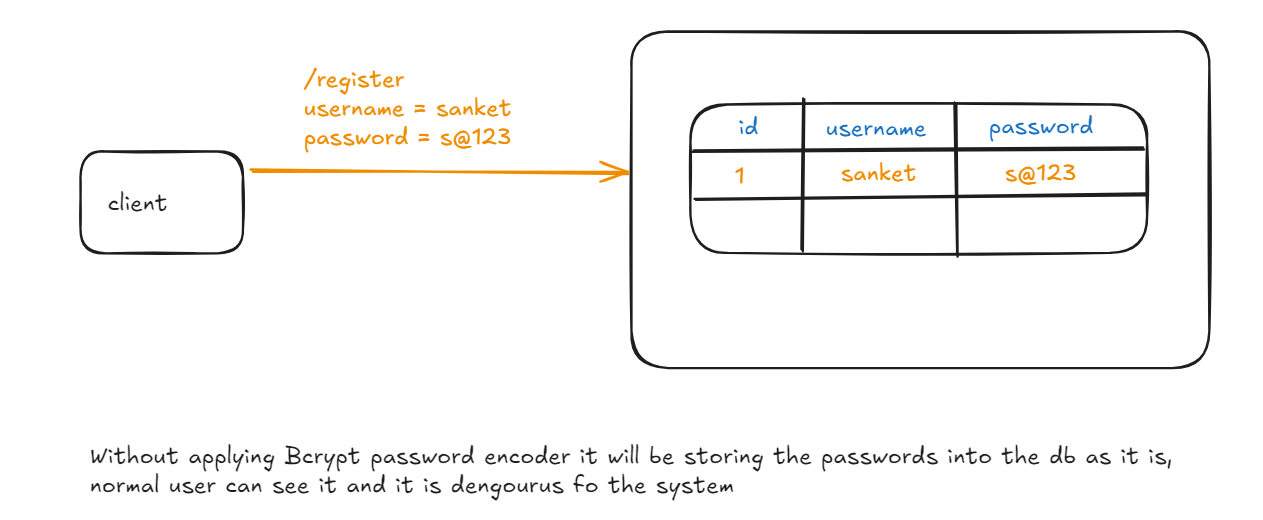
**Low Level Diagram🡪**

**A diagram of a company

Description automatically generated with medium confidence**

**6) Applying the Bcrypt password encoder to the system**

Why password encoder?

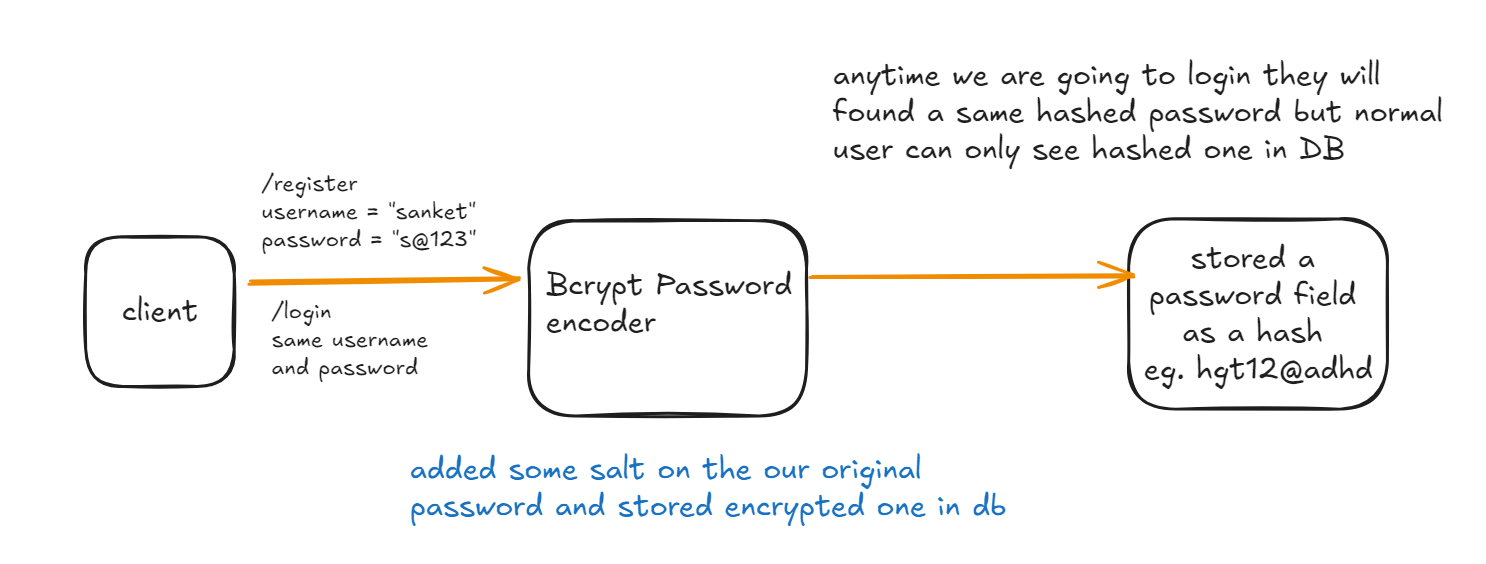


After Applying the Bcrypt password encoder-

A diagram of a computer system

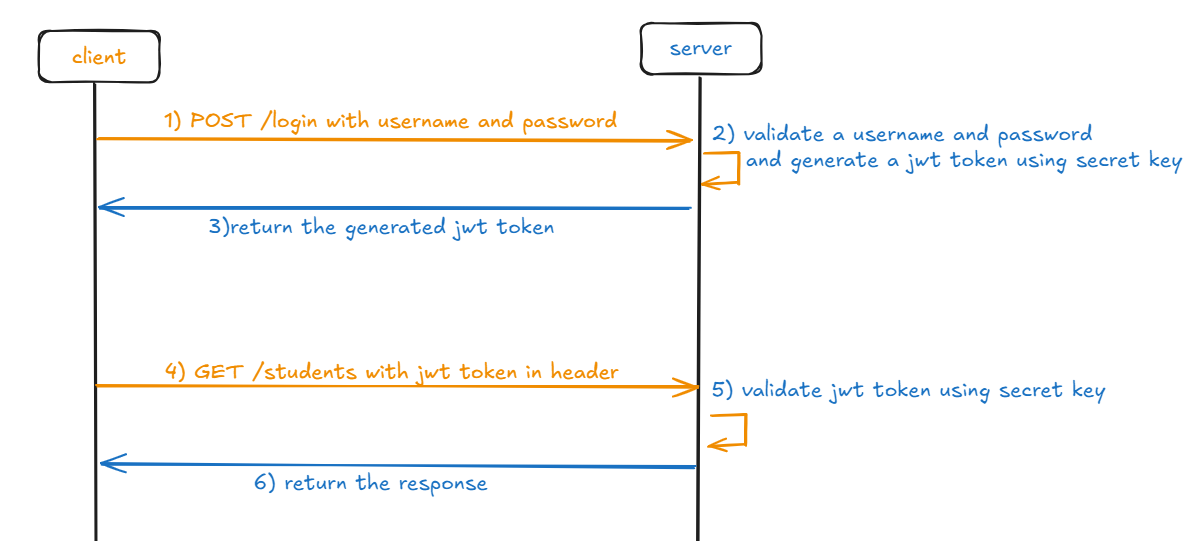
Description automatically generated

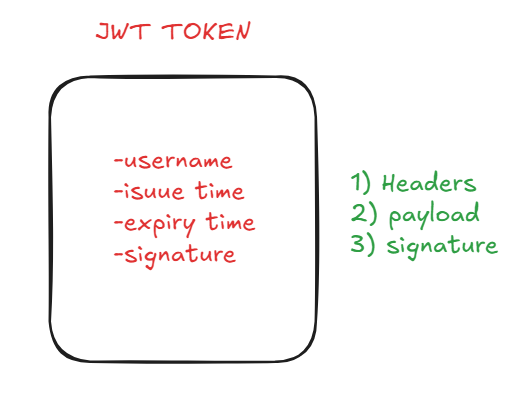
-This is how every time bcrypt are doing their work of encryption.

****

**7) What is JWT?**

- JWT (JSON Web Token) is a compact, URL-safe token used to securely transmit information between parties. In your application, it is used to authenticate users by generating a token upon successful login, which is then sent with each subsequent request. The server validates the token to ensure the user's identity and grants access to protected resources without needing to re-authenticate.





A diagram of a system

Description automatically generated with medium confidence

Fig: Implementation workflow authentication system using JWT