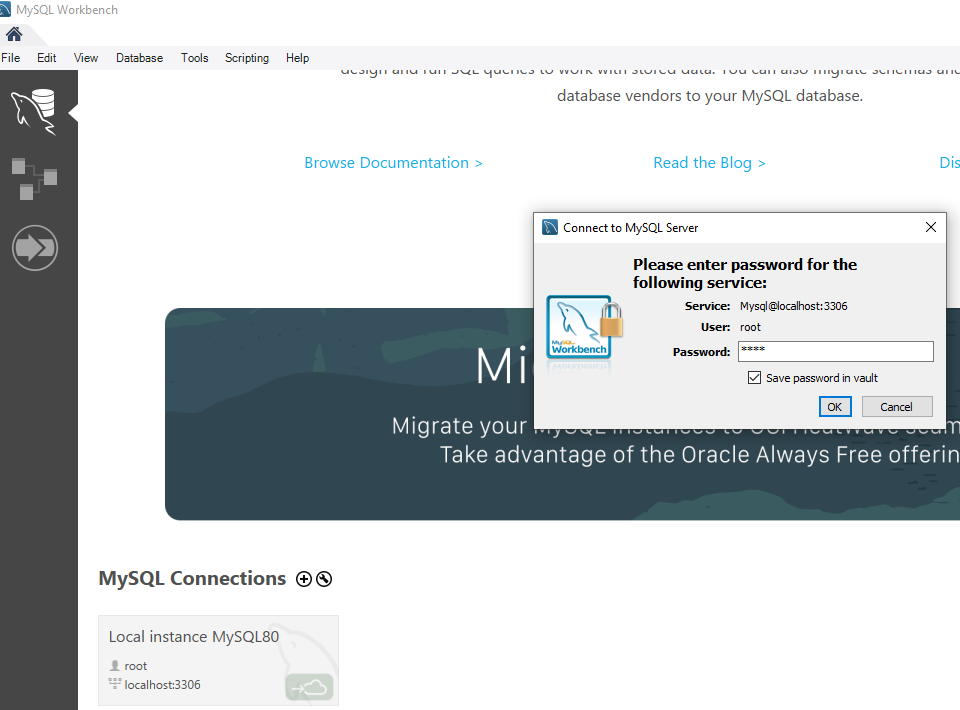
========== Spring Security Fundamentals =========  
Installed Required Software:

1. Install Java Version > 1.8
2. Eclipse or STS or Intellij
3. Mysql community server
4. Mysql workbench client
5. Postman client

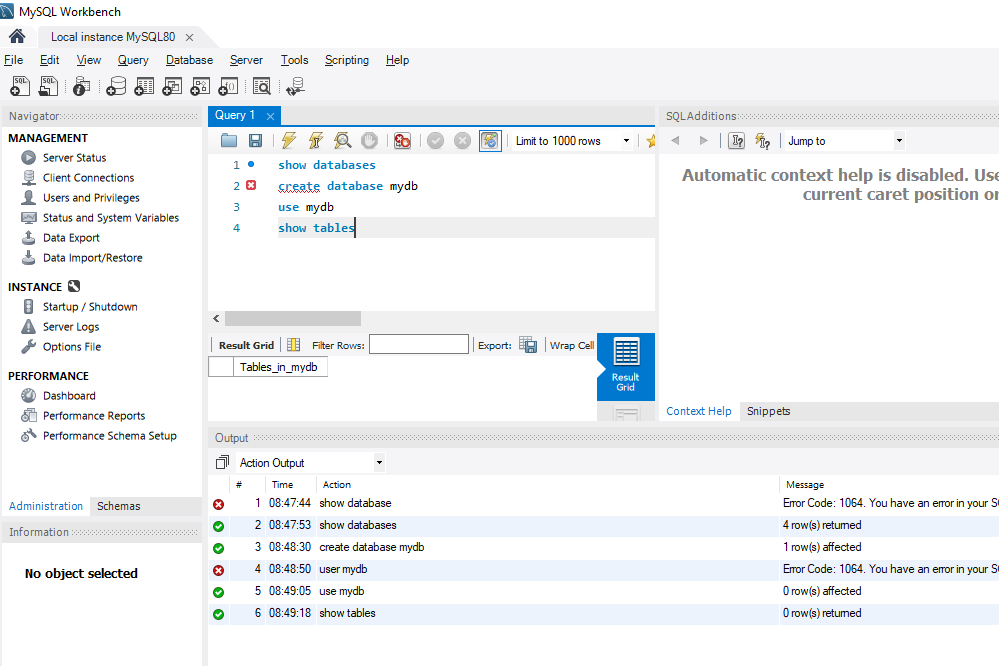
Once all is install you have to connect mysql workbench client with mysql community server, In my call mysql server username = root and password = root.



Once work bench client is install then create database(mydb) and then create tables like below:

Query:

1. **show databases**
2. **create database mydb**
3. **use mydb**
4. **show tables**



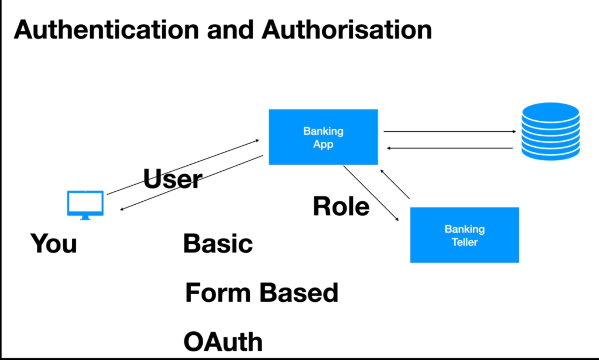
# The Fundamentals:

## What is Security:

There are the two thing Authentication and Authorization:

**Authentication**- Only application kaa access he ki nhi uske liye just check krte he ki user authenticated user he ki nhi that is called authentication.

**Authorization**- Ye next step he to check user ko kon kon ki services yaa products ka access he. To check user roles .. that is called authorization.

There are different types of authentication like – Basic authentication, Form based authentication, OAuth(open authentication and authorization) authentication, and also custom authentication.  


**Confidentiality:**

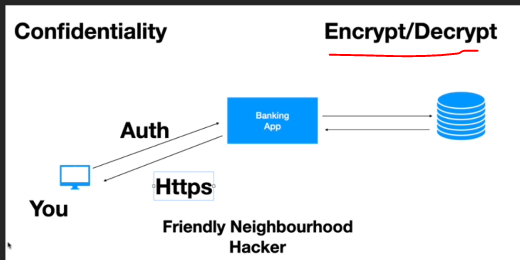
Means application(bank application) jo bhi data user ya client ko send kr rhi he vo hacker ke dwara hack to nhi kiya jaa rha yaa or data is no venerable to hacker.

Means you login to the application or friend Neighbourhood hacker hacks your application over the network.

Is sab ko roken ke liye confidentiality comes into pichhar – we use encryption and decryption.

Simplest way to use encryption and decryption by using HTTP’S. Once the communication is encrypted by using certain key(public Key) and send’s user details and the application will use the private key which will decrypt those details. Ager hacker get the those details uske bad bhi hack nhi kr payega kyoki unless private key nhi hogi uske pas.

Data dekh hi nhi payega hacker kyoki private key nhi hogi to.

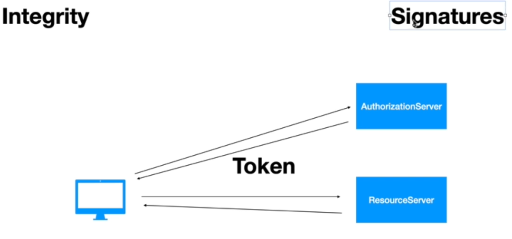
  
  
**Integrity:**  
Means hacker ne data hack kiya he or us data me kuchh change kr ke bhej rha he to server re verity krega “**Signature**” ka use kr ke that is called integrity maintain he ki nhi application. Means to maintain integrity we use Signature.

OAuth and JWT use authorized server and resource serve for signature integrity.

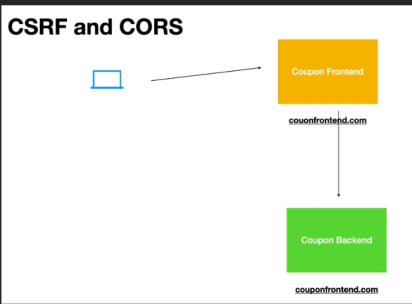
When Authorized server create a Token and gives to the application and then application to send that Token to the resource server to fetch any data.

Resource server verify krta he that Token by the signature.

Means authorized server will use private to generate Token.



**CSRF (Cross Site Request Forging) and CORS (Cross Origin Resource Sharing) :  
CSRF** is prevent request coming from other site on your behalf. Means koi dusari site request bhejati he apki jagah pr to usko rokta he.

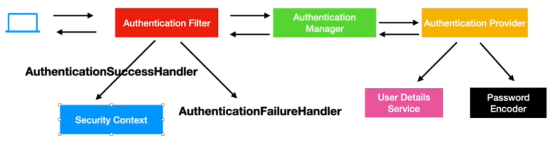
**CORS** prevent same server communication, means apne two modules bnaye he UI and Backend. To un dono ke bich communation ho uske liye CORS use krte he means enable krna pdta he.

## The Key Security Components:

When rest full client send request tab kya kya hoga he in terms of security and token uses. Jab RESTFull send request very first component is called Authentication Filter in spring security that intercept restfull request.

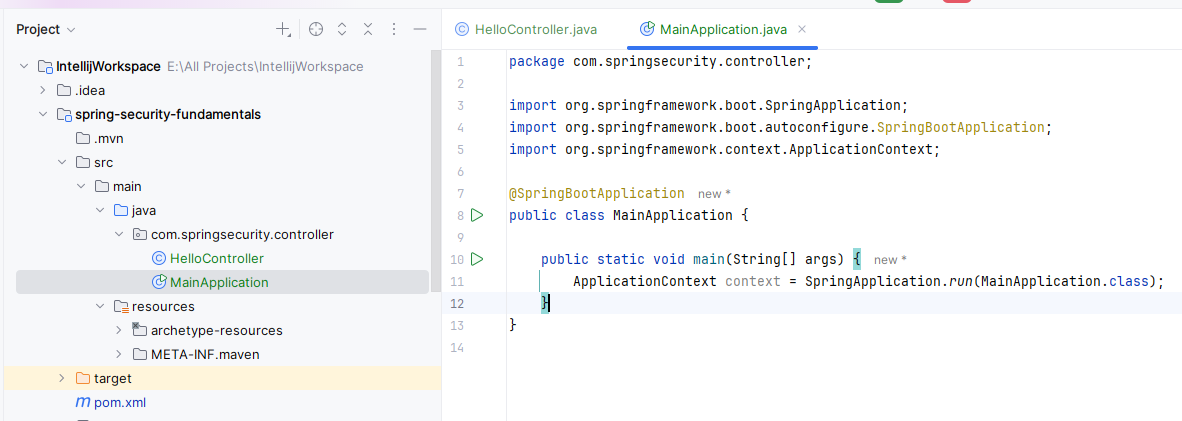
There are 6 components:

1. Authentication Filter- Is a servlet filter class, current user is authenticated or not. Once the user authentication send success response then it will stores data to the Security Context Holder.
2. Authentication Manger
3. Authentication Provider – this will not fetch data from database or LDAP or any source. It uses User details service and Password Encoder. If user authenticated by using user details service and password provider then only it will send back data to the authentication manager.
4. User Details Service
5. Password Encoder
6. Security Context



## Spring Security In Action – Example

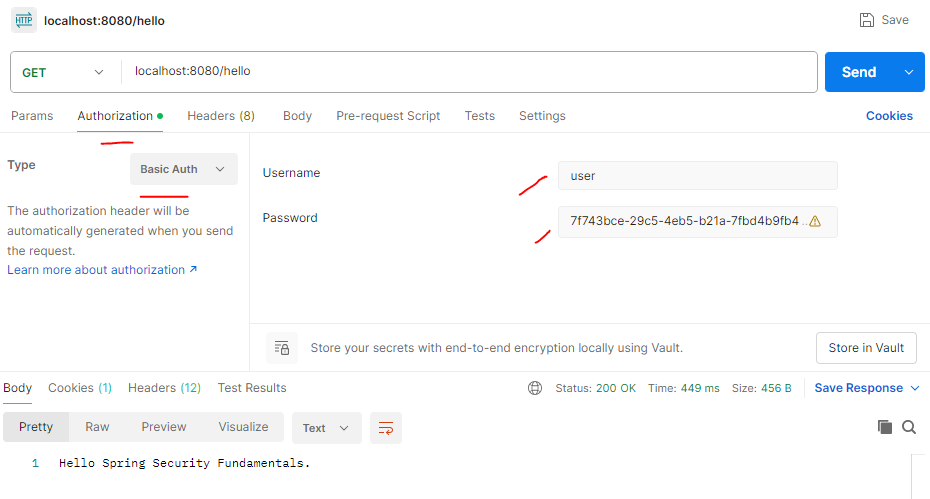
First Spring Security Project, Below are the code you have to add in your POM.xml and Java.

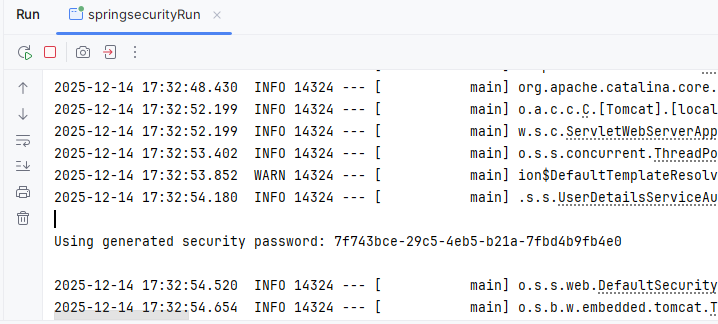






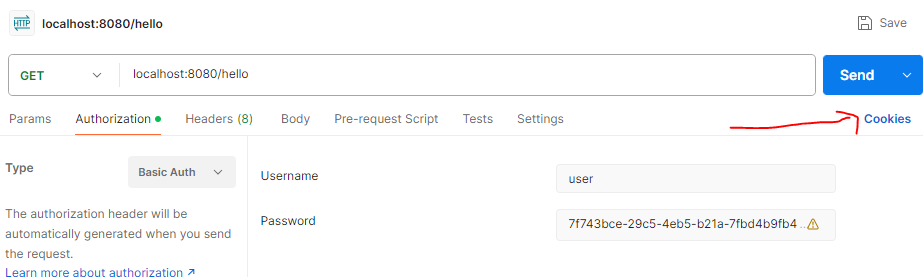
Once you start application then spring security will generate password in console for user login.

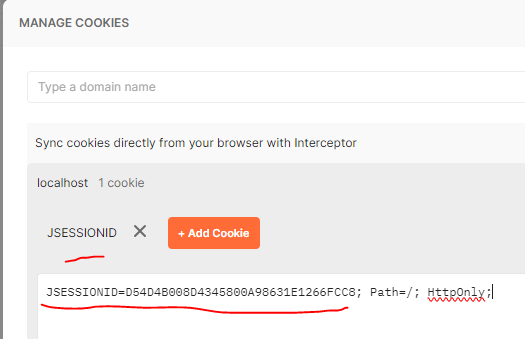




Once your login on PostMan or any other tool by passing username and password and JSON Cookies(JssessonID) then that login details are stored in SecurityContext and jab bhi request heat hogi to JSONCookies(JssessonID) and send hogi username or password ke sath me until ki server restart nhi huo.

JsessionId aane ke bad Application Filter or SecurityContext use same JsessionID for next time.

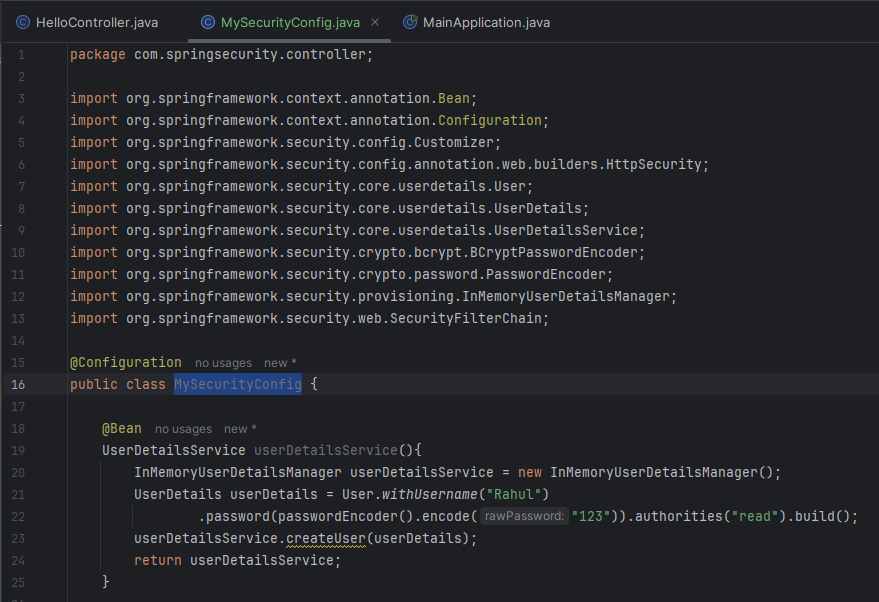




Bove are the default implementation of spring security for security.

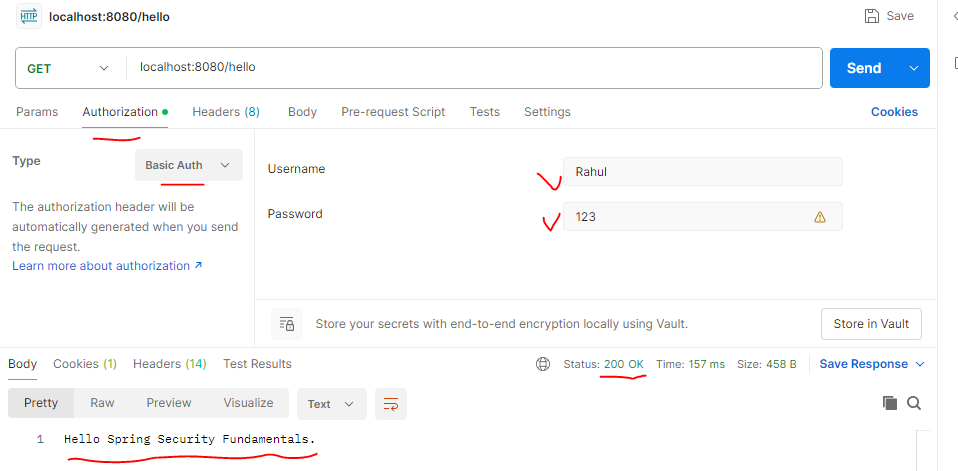
## Create Custom Security Configuration:

Create our custom MySecurityConfig.java in the same package where you have controller:

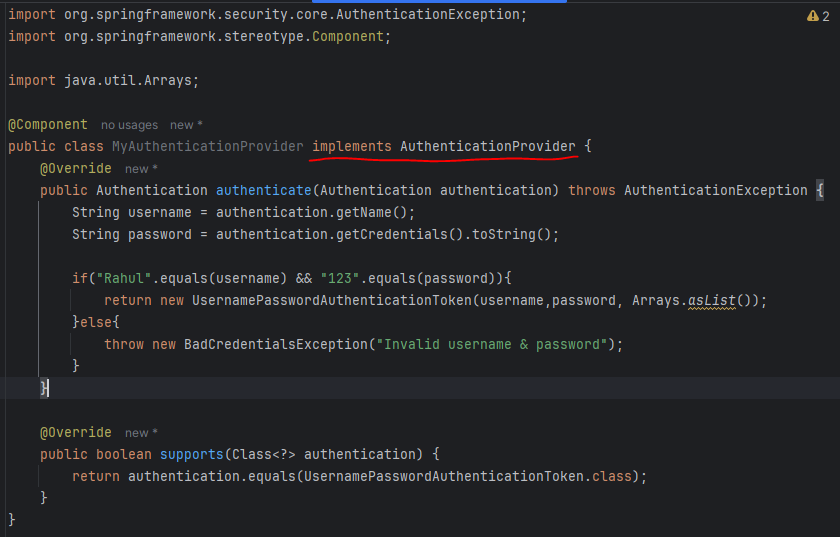


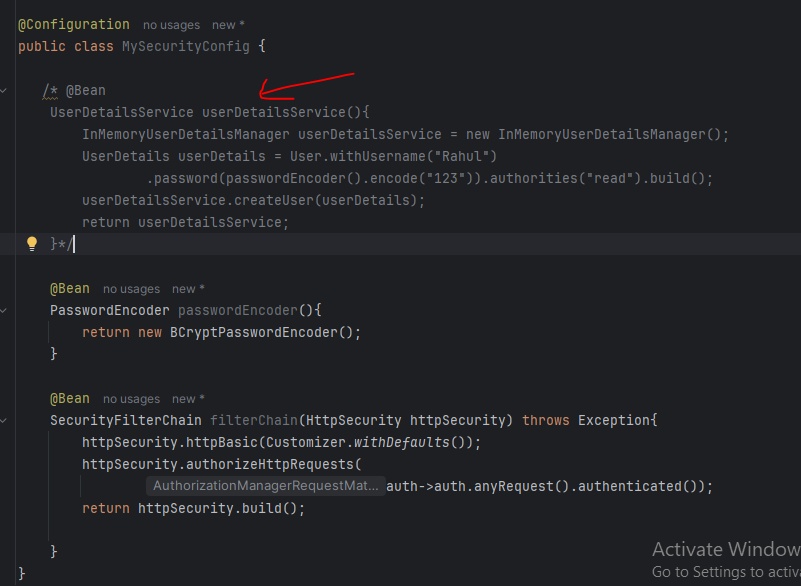


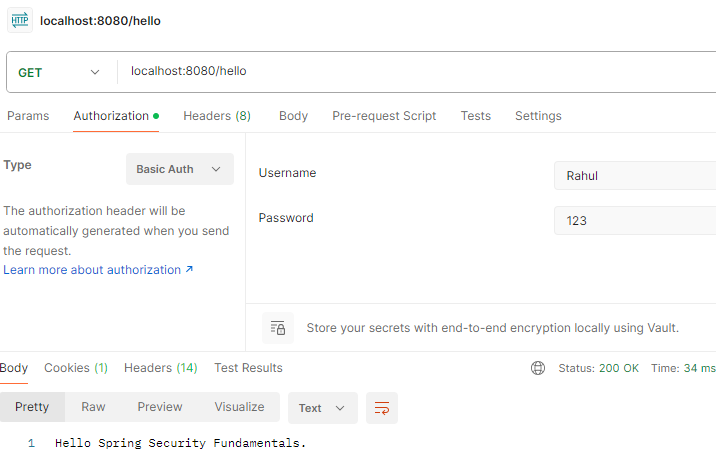
Run the Server – and now this time you will not see spring security generated password:  

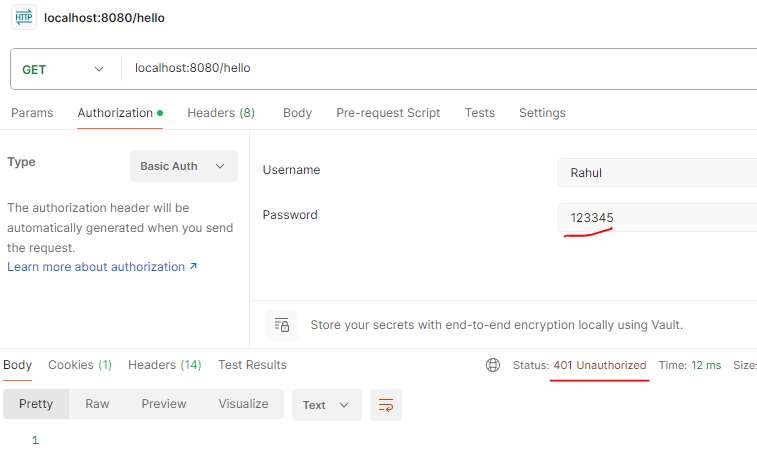



## Create Custom AuthenticationProvider:

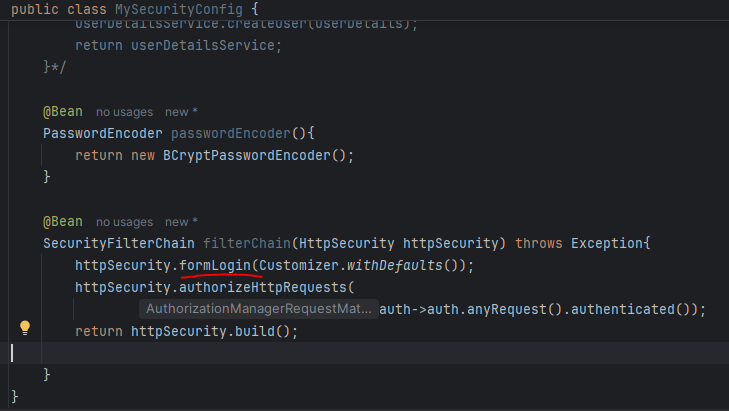


Now to use our own AuthenticationProver(above) we need to comment out custom userdetailsService bean, Like below :  


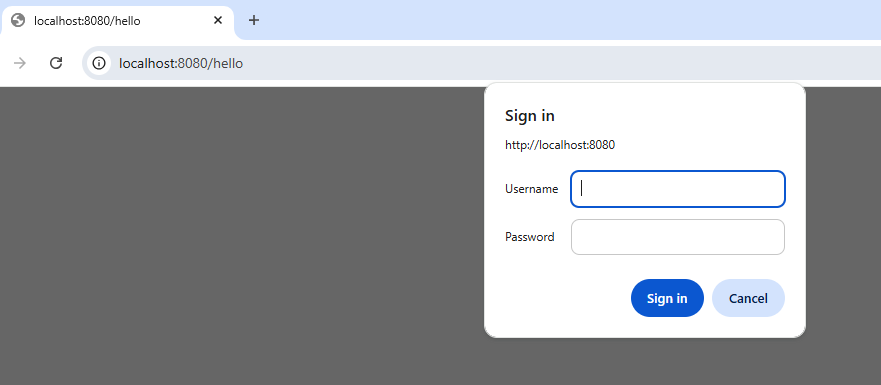




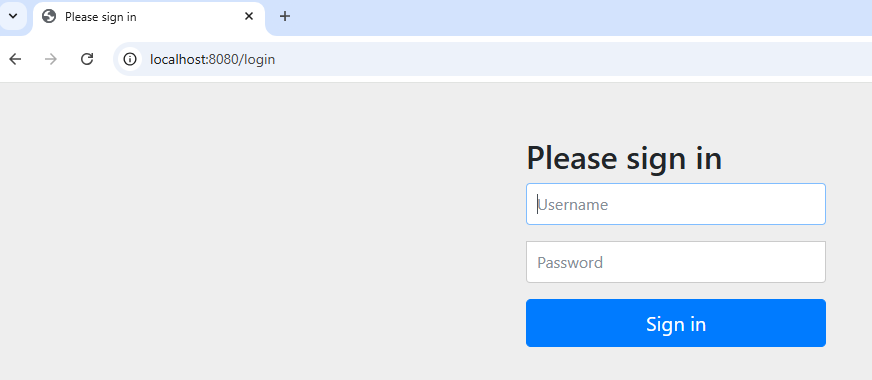
## Form Based Login :

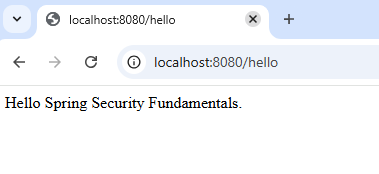


Let see login form in browser:



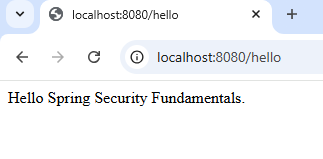
Or

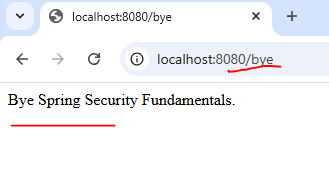




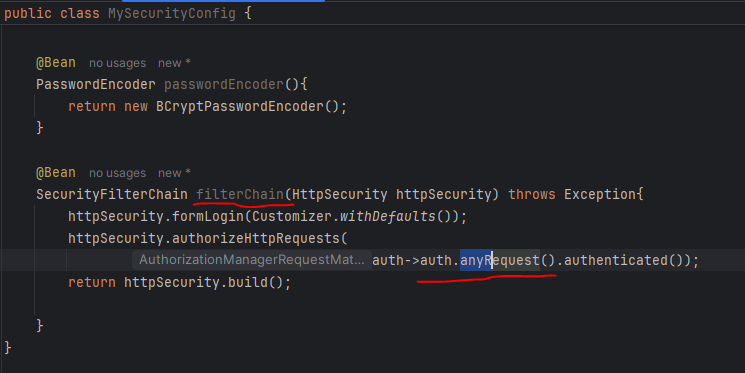
## Few More Methods ( permit):

Now see hmne login kiya aik api me (hello me)

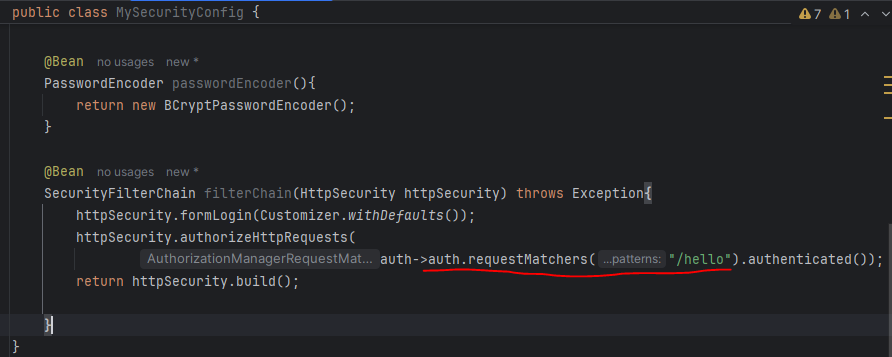


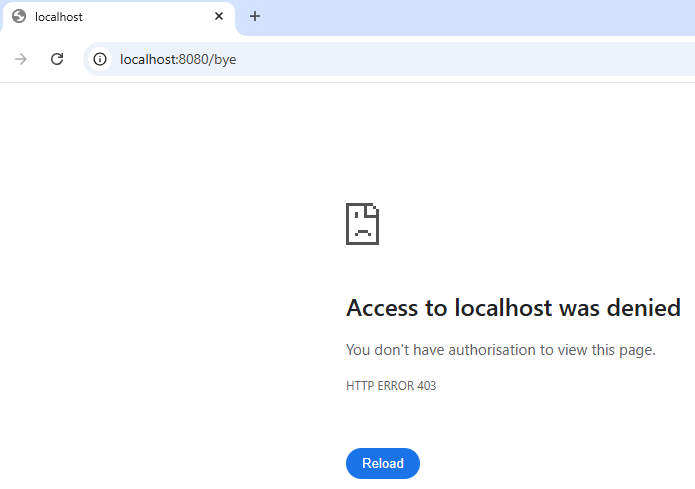
Now hitting /bye api :  


In this case its still fetching data - . means all request are accessible due to below code: anyRequest() method



Now If you want to restrict anyRequest(), Then we can use another method called:  
requestMatchers().



After giving requestMatcher permission only “/hello” then we are able to access only “/hello” but not the bye its giving below error:  


## Other Filter Classes:

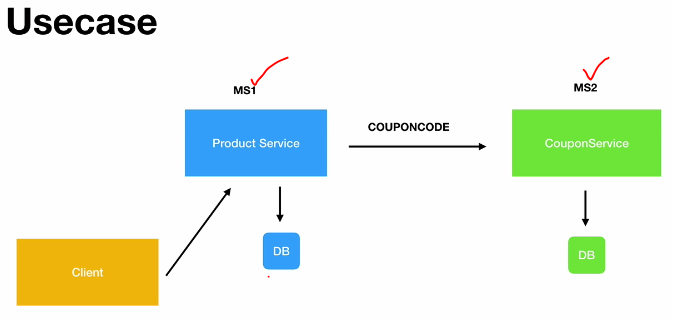
GenericFilerBean class:  


If you have guaranty that your filter logic will execute only once then you can go for:  
OncePerRequestFilter class:



# Create Microservices:

## Usercase:



**DataBase:**  
use mydb;

create table product(

id int AUTO\_INCREMENT PRIMARY KEY,

name varchar(20),

description varchar(100),

price decimal(8,3)

);

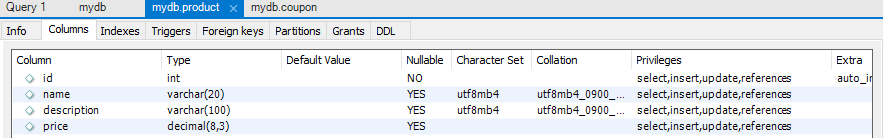
create table coupon(

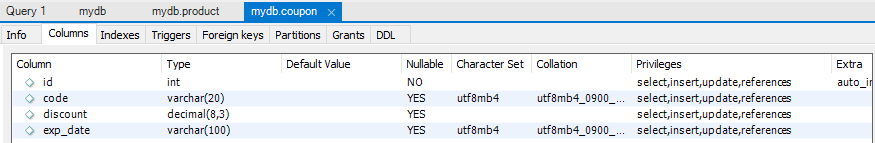
id int AUTO\_INCREMENT PRIMARY KEY,

code varchar(20) UNIQUE,

discount decimal(8,3),

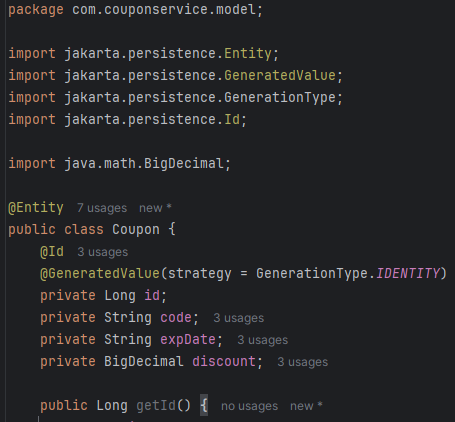
exp\_date varchar(100)

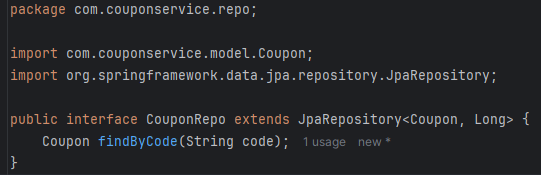
);  


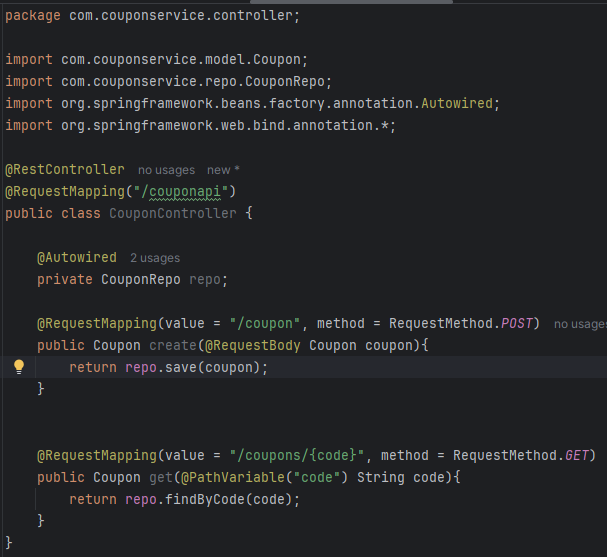


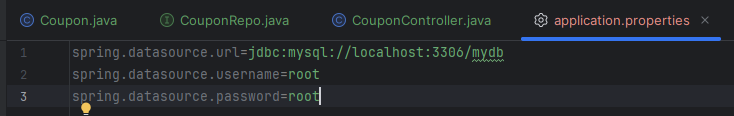
## Create coupon service module:

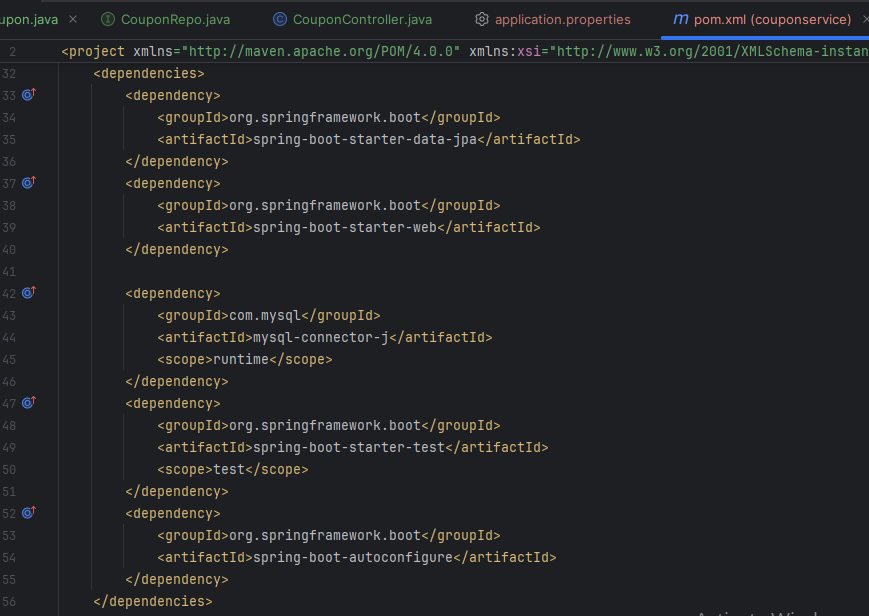
using - <https://start.spring.io/>

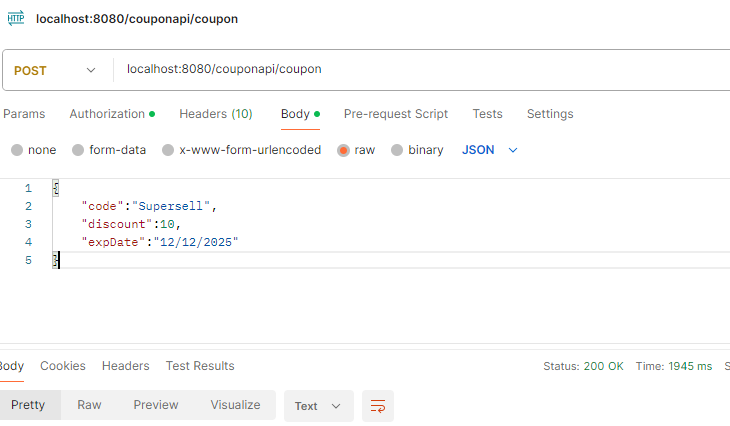


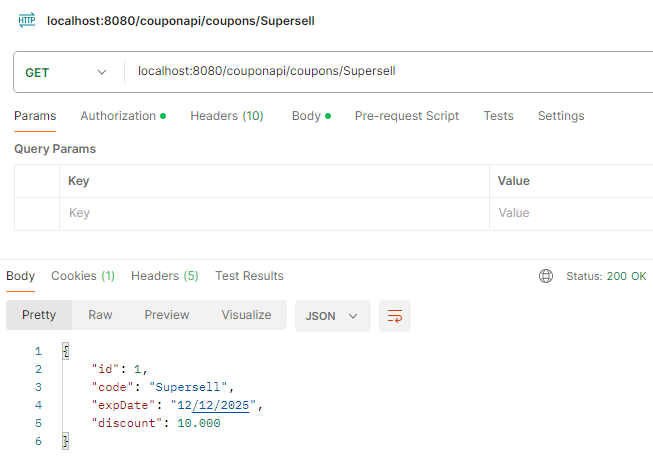






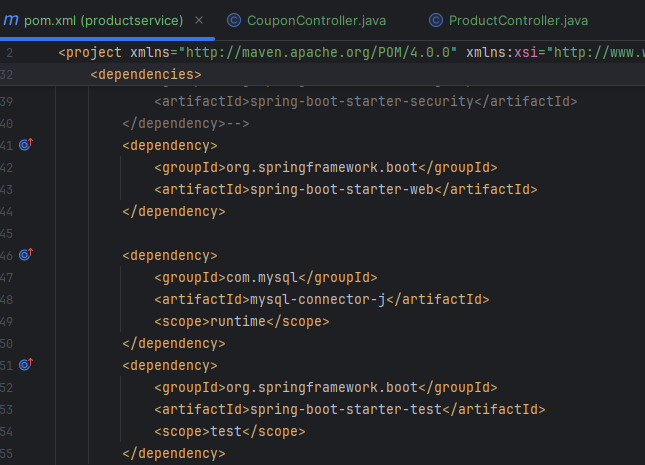


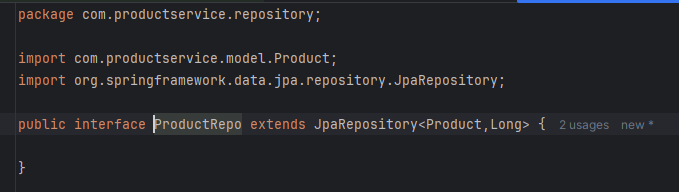


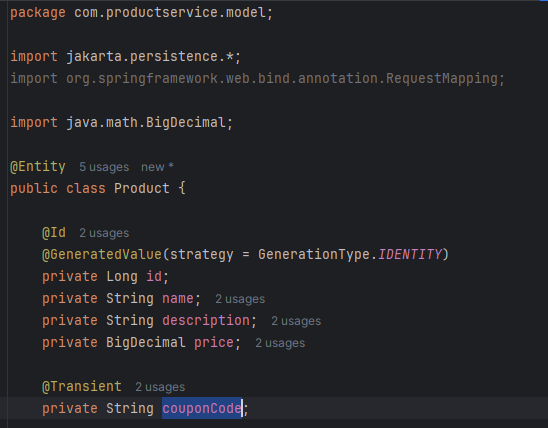


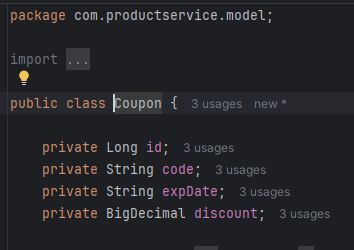
## Create product service module:

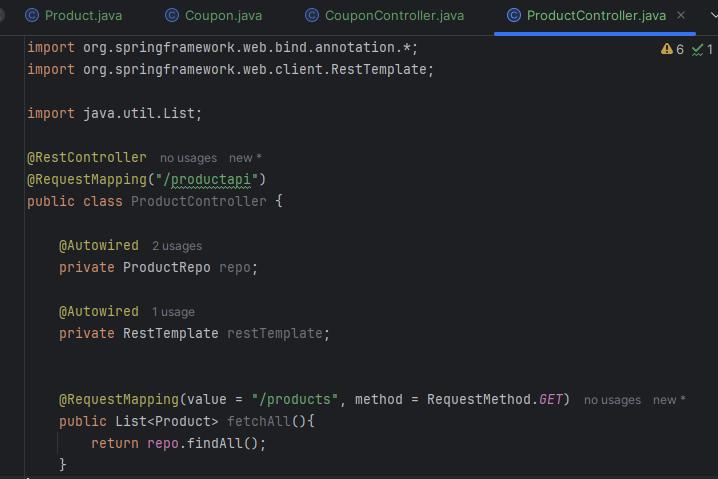
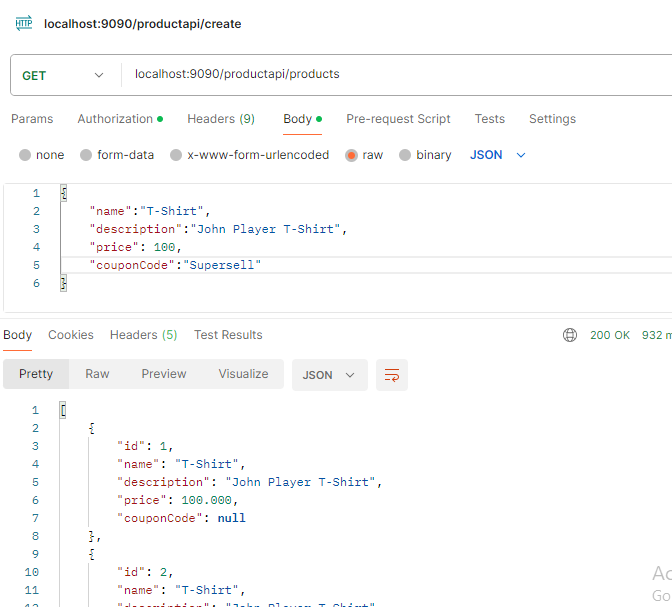
using - <https://start.spring.io/>







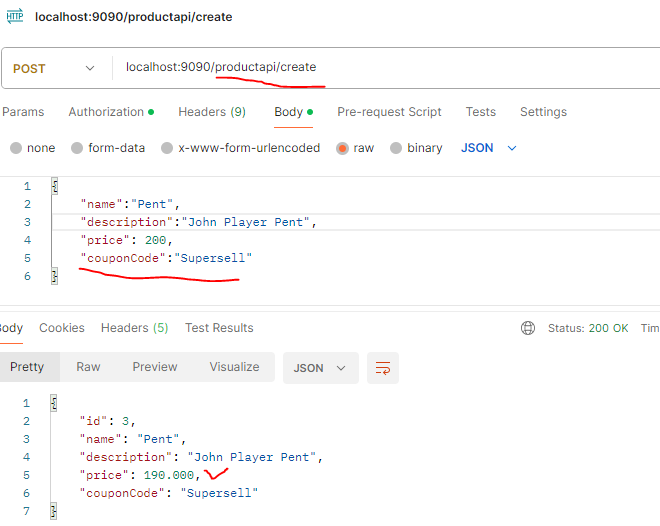
 

## Microservices Call:

Now product service se couponCode pass kr ke product ke price me discount calculate kr ke create krenge new product.

Coupon Service Running on Port 9091, And Product Service Running on Port 9090





# Secure Rest API:

AB ye jo user services bnai he(product and coupon) service lo secure kran he.

User Role ke hisab se product add hoga.

Admin role hi coupon ko create or edit or delete kr ska he bas.

We are using two roles(Admin and User).

## DataBase:

use mydb;

CREATE TABLE USER

(

ID INT NOT NULL AUTO\_INCREMENT,

FIRST\_NAME VARCHAR(20),

LAST\_NAME VARCHAR(20),

EMAIL VARCHAR(20),

PASSWORD VARCHAR(256),

PRIMARY KEY (ID),

UNIQUE KEY (EMAIL)

);

CREATE TABLE ROLE

(

ID INT NOT NULL AUTO\_INCREMENT,

NAME VARCHAR(20),

PRIMARY KEY (ID)

);

CREATE TABLE USER\_ROLE(

USER\_ID int,

ROLE\_ID int,

FOREIGN KEY (user\_id)

REFERENCES user(id),

FOREIGN KEY (role\_id)

REFERENCES role(id)

);

insert into user(first\_name,last\_name,email,password) values ('doug','bailey','doug@bailey.com','$2a$10$U2STWqktwFbvPPsfblVeIuy11vQ1S/0LYLeXQf1ZL0cMXc9HuTEA2');

insert into user(first\_name,last\_name,email,password) values ('john','ferguson','john@ferguson.com','$2a$10$YzcbPL.fnzbWndjEcRkDmO1E4vOvyVYP5kLsJvtZnR1f8nlXjvq/G');

insert into role values(1,'ROLE\_ADMIN');

insert into role values(2,'ROLE\_USER');

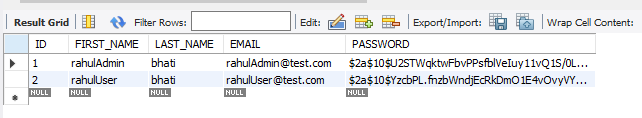
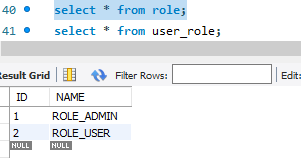
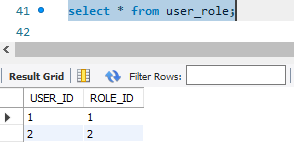
insert into user\_role values(1,1);

insert into user\_role values(2,2);

select \* from user;

select \* from role;

select \* from user\_role;

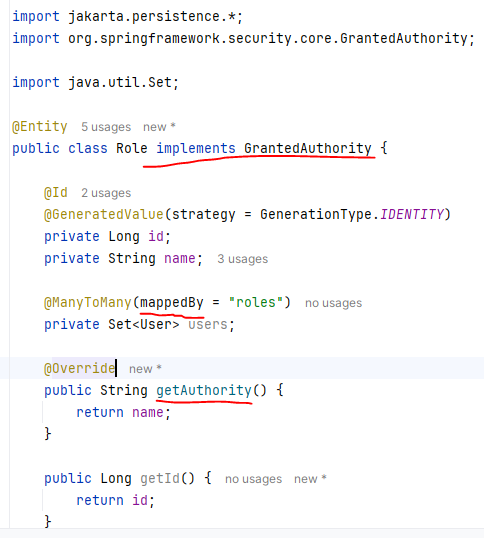
## Secure API:

Now Secure Coupon Service: by adding spring security dependency in pom.xml

*-<dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <scope>test</scope>  
</dependency>*

**User and Role Model and there Mapping:**



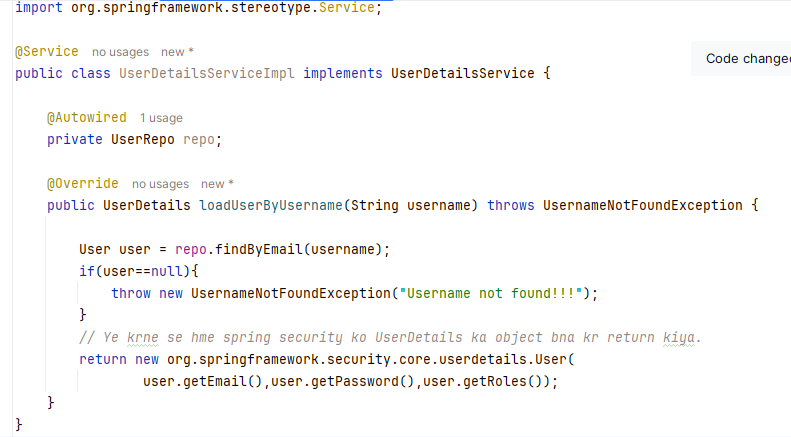


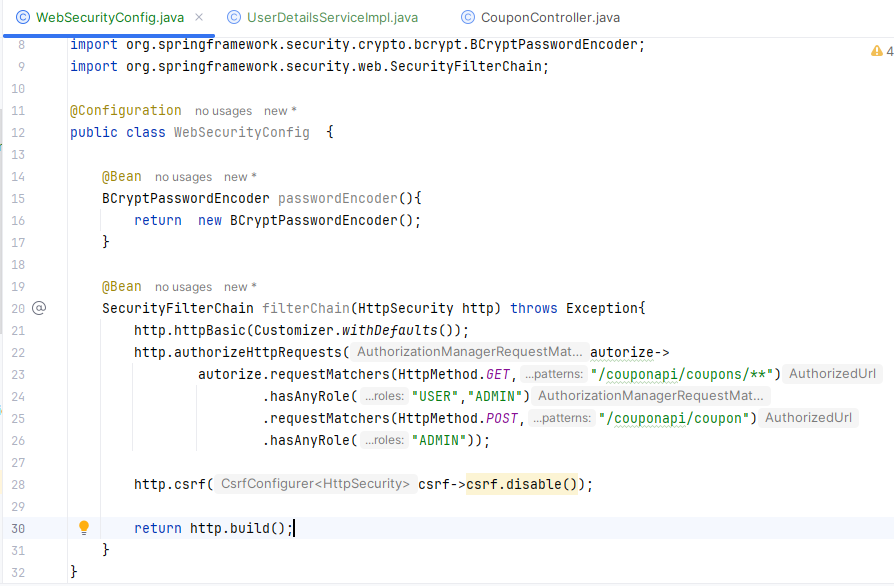
**Custom UserDetailsService**

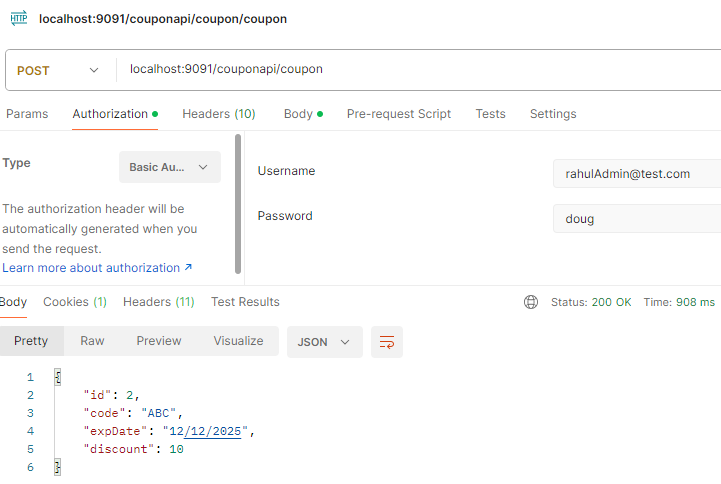
***// Ye krne se hme spring security ko UserDetails ka object bna kr return kiya.***

**Kyoki spring boot ko springboot ka User class chahiye.**

Isliye hm email(username) pass kr ke data get kr rhe he database me se or pass work rhe he spring boot User class ko.

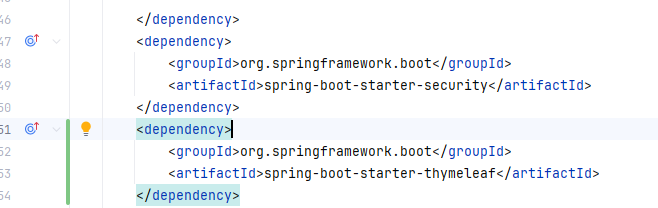


Configure WebSecurityConfig so that only admin role user can create the coupon and other user only can see the Get api or coupon details:  


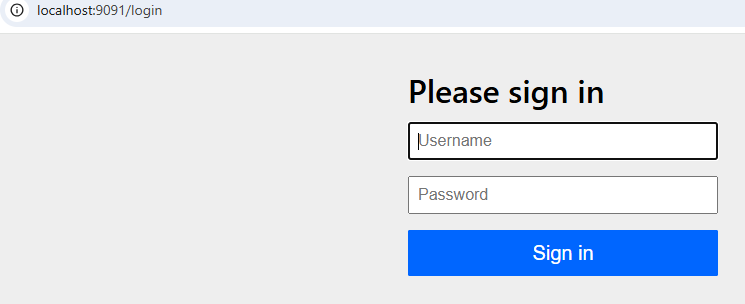


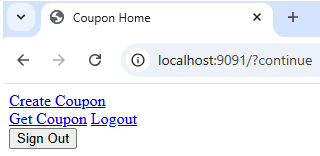
# Secure a WebApp:

**Thymeleaf Dependency:**







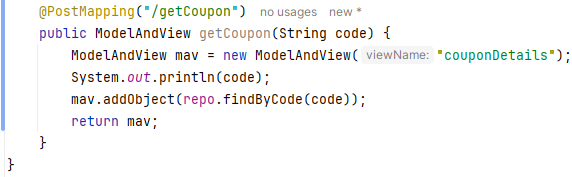


Till now we have implemented login and index page.

Now we are going to give the permission to the USER and ADMIN role.

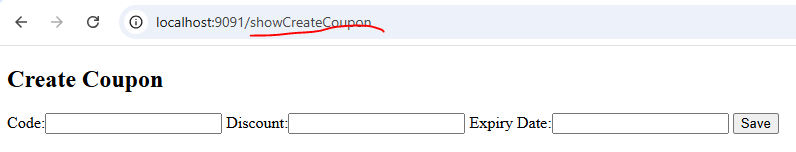
Admin only can create coupon and save it.

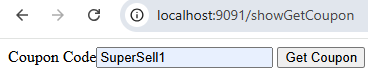
User role only able to get the coupon details.

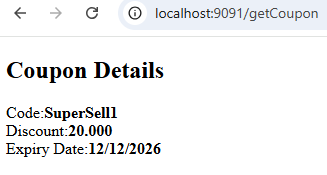
 



Admin Role can create role:







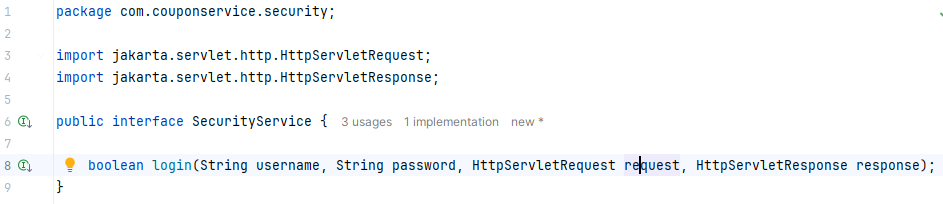
## Implement Custom Login:



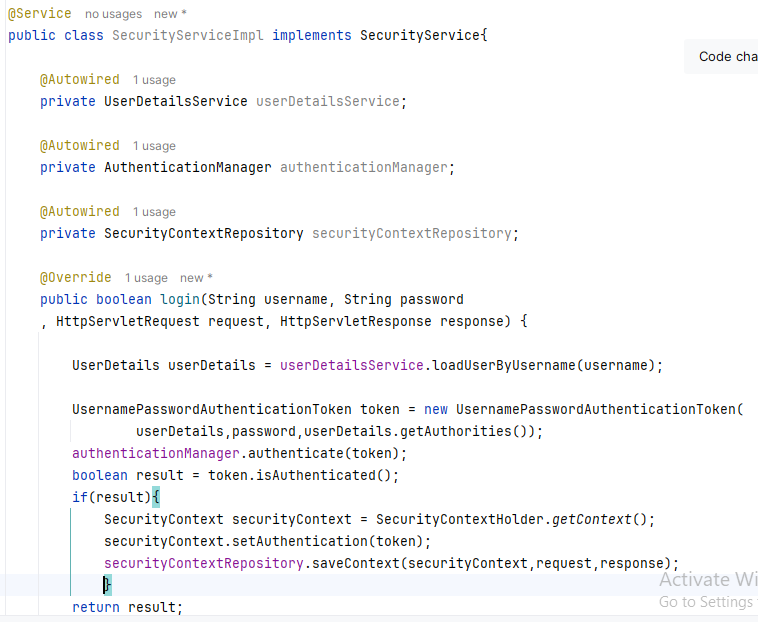
**UserController.java**



**SecurityService Interface:**



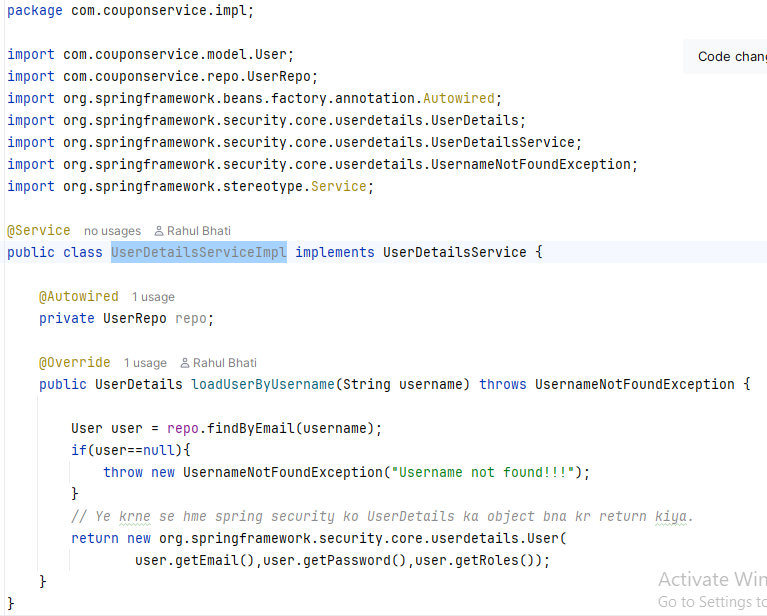
**SecurityServiceImpl.java:**

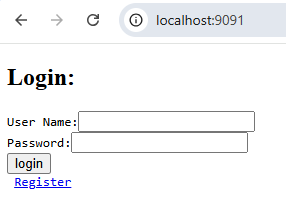
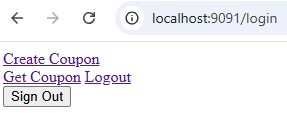


**WebSecurityConfig.java:**

**Custom UserDetailsServiceImpl.java:**



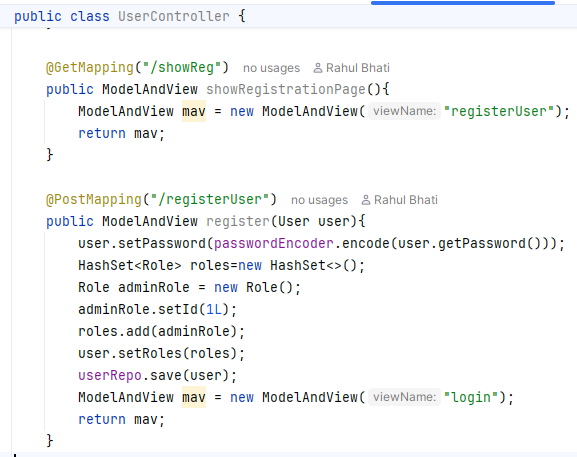
 

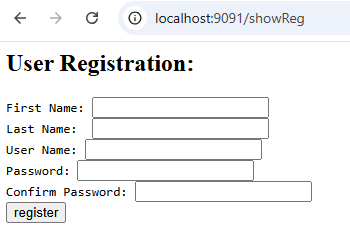
## Implement Registration:

**WebSecurityConfig:**



**UserController:**

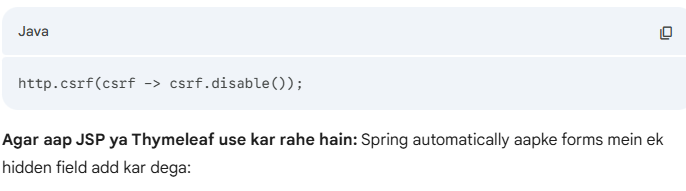
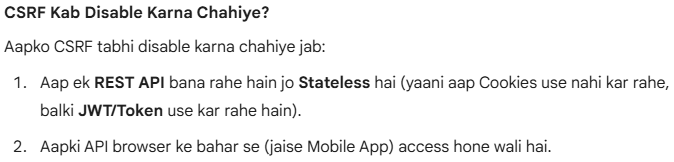




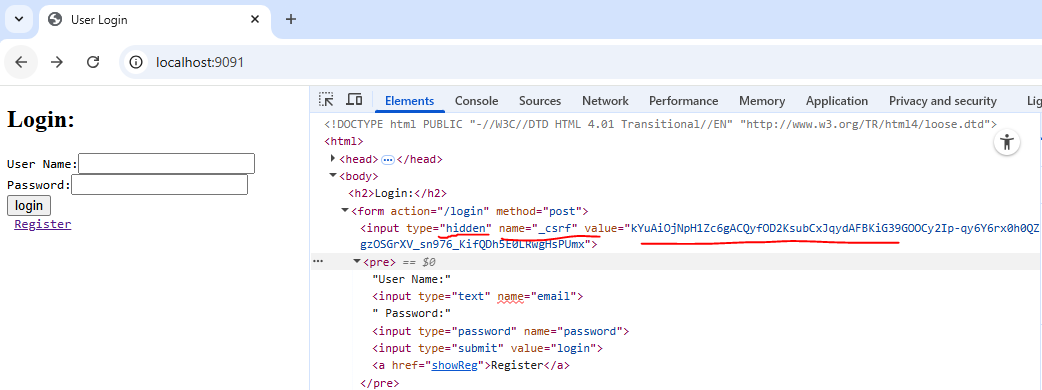
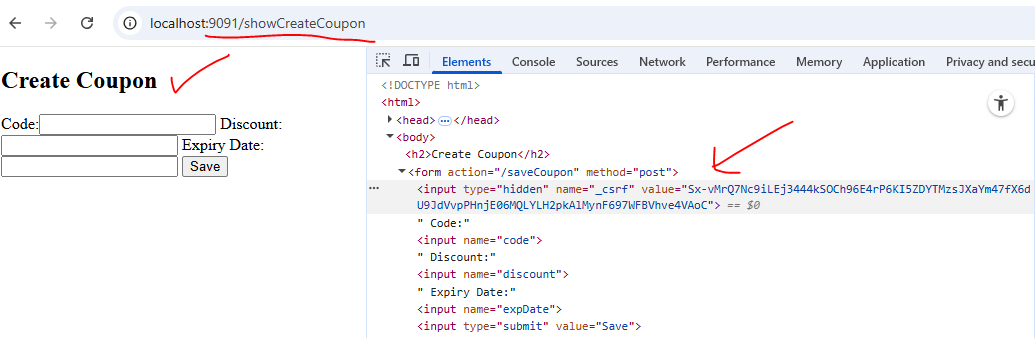
# Spring Security CSRF: (Cross Site Request Forgery):

**CSRF** ka matlab hai **Cross-Site Request Forgery**. Ye ek tarah ka web attack hai jisme ek hacker aapki browser session ka fayda uthakar aapki taraf se kisi aur website par galat request bhej deta hai.

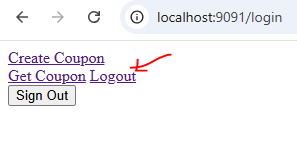
 

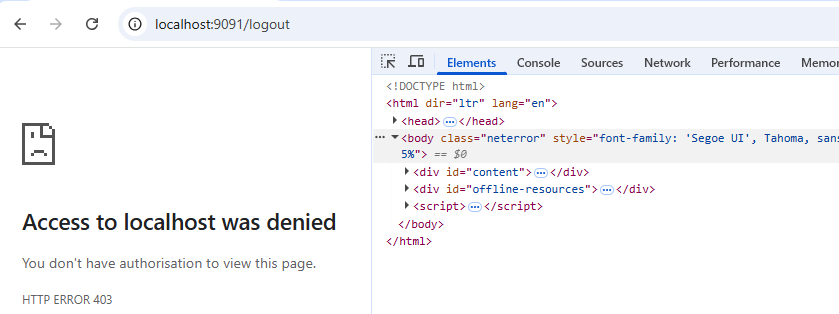
  

Every Incomming request has this hidden csrf value token.

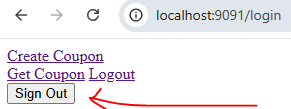
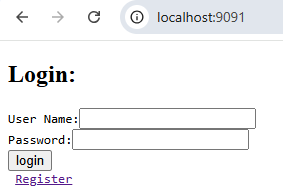
 

Jab hm logout kisi href link pr click krte he to will get error



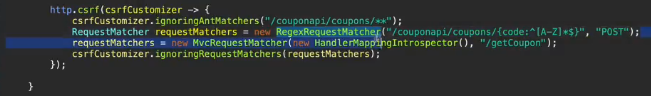


To logout button pr click krte he to it will work :

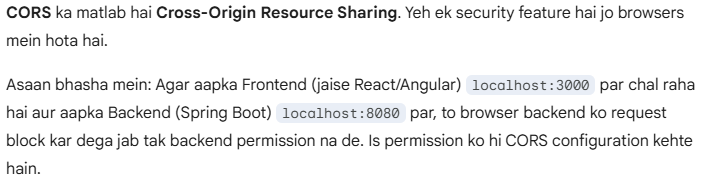
 

If you want to ignore certain URL for CSRF then you can do some configuration to skip csrf configuration.

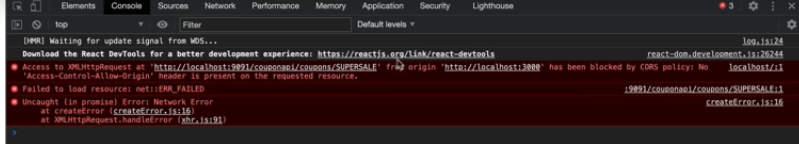
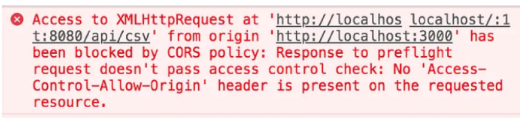
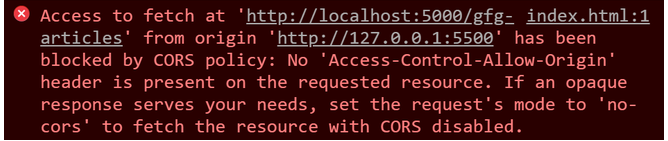


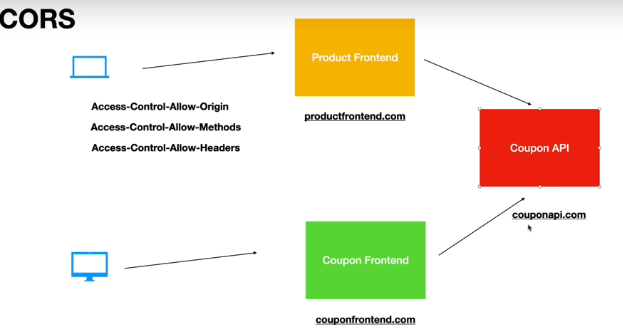
Latest code me kuchh is tarah ka changes he:  


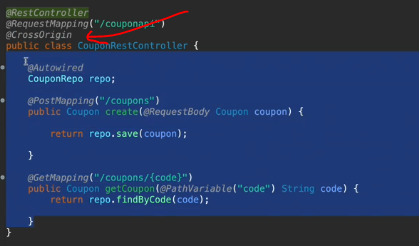
# Spring Security CORS: (Cross-Origin Resource Sharing):

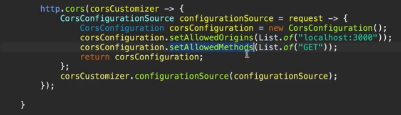


**Issue are below:**

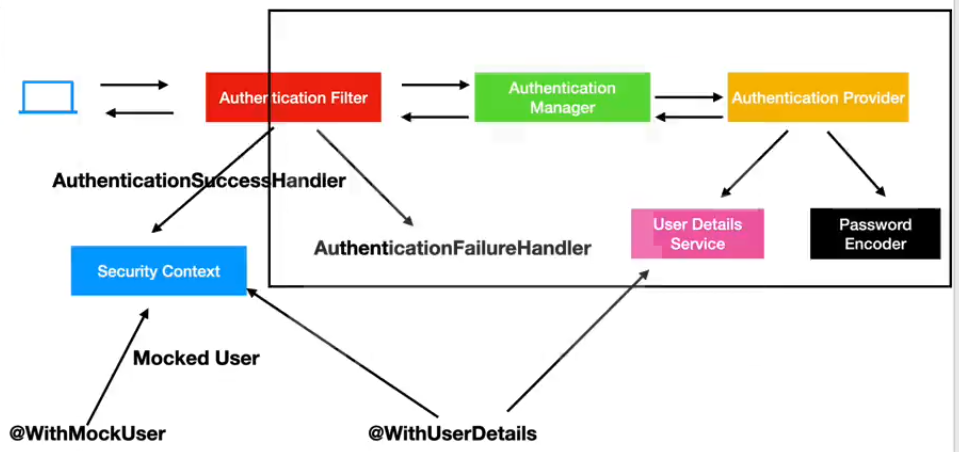
  



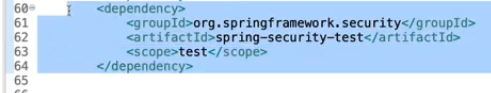
Configuration in your controller only need to add annotation - **@CrossOrigin**

**You can also customize CORS like below:**

# Spring Security Testing:



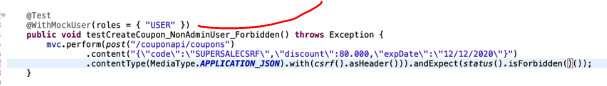
**Need to add dependency:**



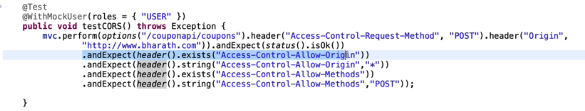


**Test CSRF Roles:**





**Test for CORS support:**



**@MockUserDetails:**  

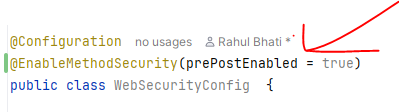

# Spring Method Level Security:



Spring Security 6.x aur Spring Boot 3.x mein **Global Method Security** ka tarika badal gaya hai. Pehle hum @EnableGlobalMethodSecurity use karte the, lekin ab ise **deprecate** karke **@EnableMethodSecurity** laya gaya hai.

Iska maqsad ye hai ki aap Controller ya Service level ke specific **methods** par security apply kar sakein, na ki sirf URL patterns par.



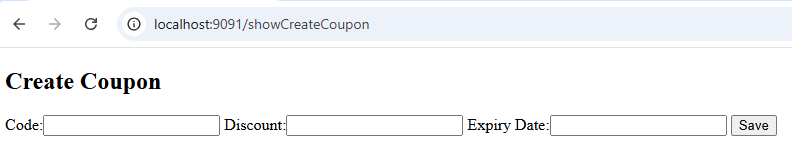
**Enable Method Security Annotation:**  


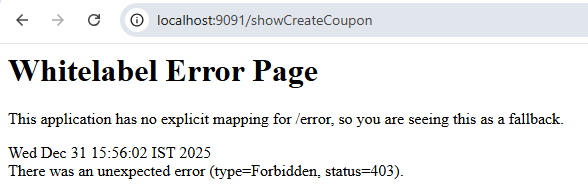


## @PreAuthorize:

@PreAuthorize Spring Security ka sabse powerful annotation hai. Iska kaam method ke execute hone se **theek pehle** access check karna hota hai. Agar condition true hai, tabhi method chalega, warna 403 Forbidden error aa jayega.

I have done the @PreAuthorize for create coupon api. Only Admin can create the coupon:  


When I logged in as Admin:  


But when I logged in as normal User: then I will get below error:  


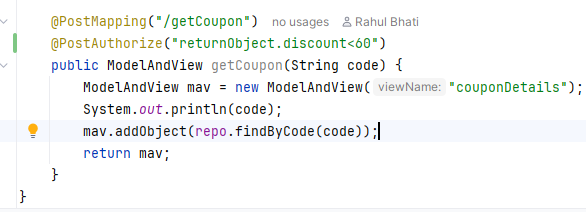
## @PostAuthorize:

@PostAuthorize ka use tab kiya jata hai jab aapko method **execute hone ke baad** security check lagani ho.

Iska sabse bada fayda ye hai ki aap method ke **return value** (jo data method ne fetch kiya hai) ke basis par decide kar sakte hain ki user ko wo data milna chahiye ya nahi.



Ager Coupon discount < 60 he to hme return nhi krvana he to we can do like below:

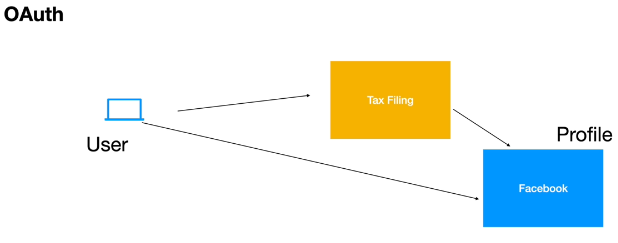


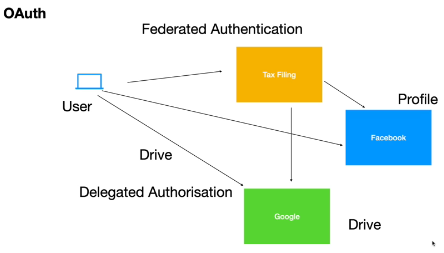
# OAuth and JWT Concepts:

**OAuth 2.0** ek standard protocol hai jo **"Delegated Authorization"** ke liye use hota hai.

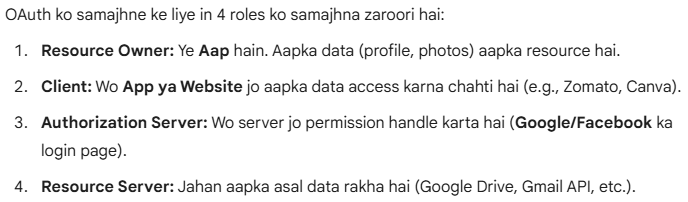
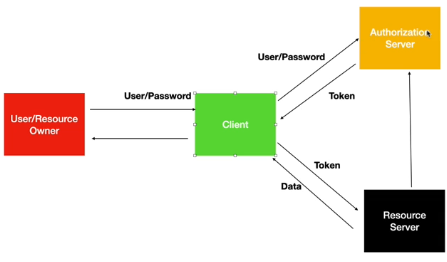
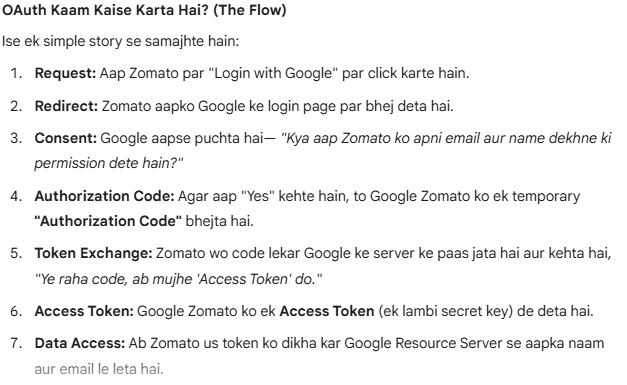
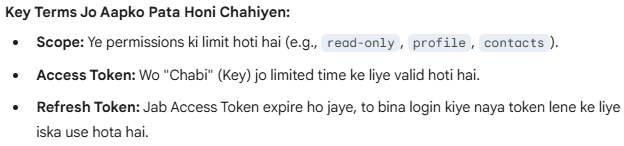
Iska asaan matlab hai: **Apna password diye bina, kisi teesri app (Third-party app) ko apne data ka access dena.**

Sabse common example hai **"Login with Google"** ya **"Login with Facebook"**. Jab aap kisi nayi website par Google se login karte hain, to aap us nayi website ko apna Google password nahi dete, balki Google se ek "Permission" dilwate hain.





## OAuth ke 4 Main Characters (Actors):

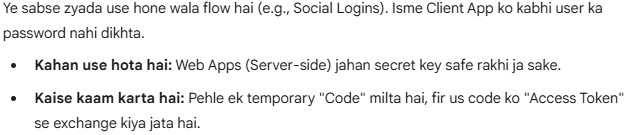
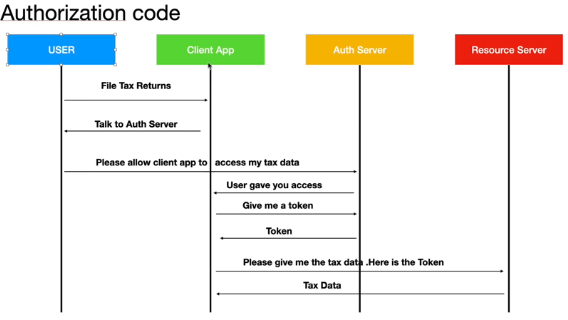
   

## Grant Type:

OAuth 2.0 mein **Grant Type** ka matlab hai wo "Rasta" ya "Tarika" (Method) jiske zariye ek Client App ek **Access Token** hasil karti hai.

Alag-alag scenarios ke liye alag-alag Grant Types bane hain. Spring Security 6 mein inka sahi istemal samajhna bahut zaroori hai.

### 1. Authorization Code Grant (Sabse Secure):

### 2. Client Credentials Grant:

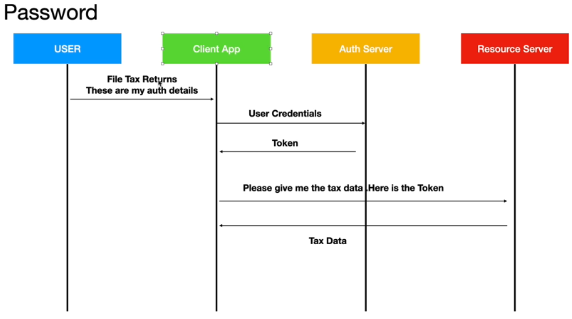




### 3. Password (Resource Owner Password) Grant — **[DEPRECATED]:**

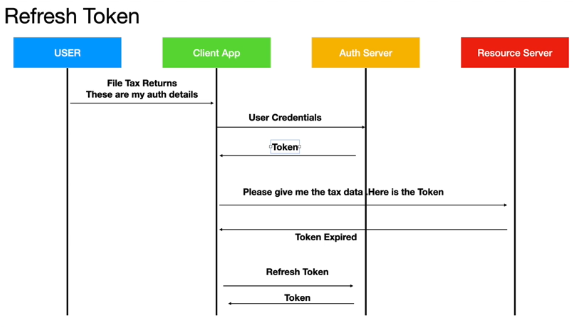
Isme user apna username aur password direct Third-party app ko de deta hai.

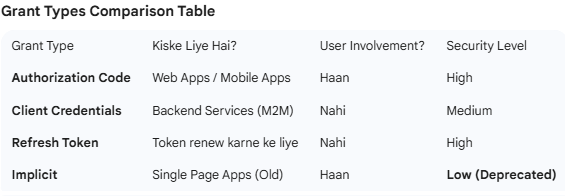
* **Status:** Ab ise use nahi karna chahiye (Security risk). Iski jagah **Authorization Code with** PKCE (Proof Key for Code Exchange) use hota hai.



### Refresh Token Grant:

Jab Access Token expire ho jata hai (man lijiye 1 ghante baad), to user ko baar-baar login na karna pade, isliye Refresh Token ka use karke naya Access Token liya jata hai.

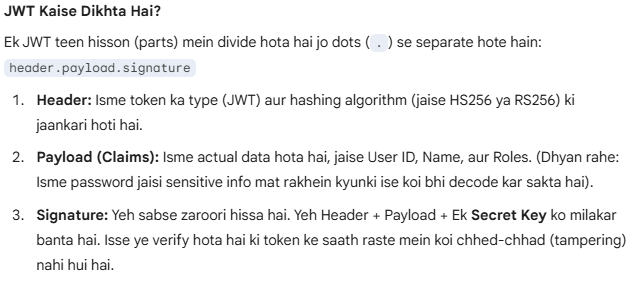




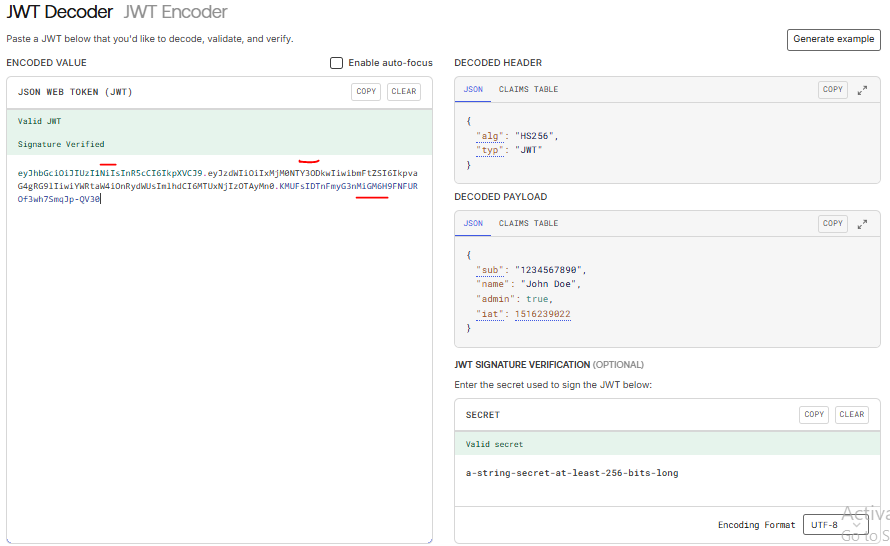
# JWT Introduction:

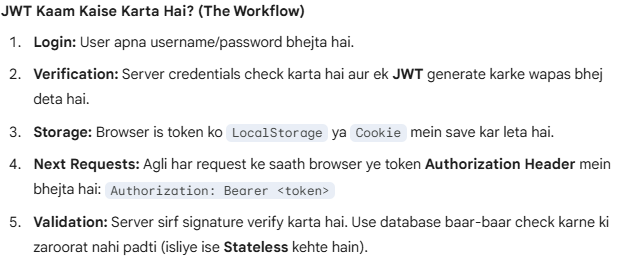
**JWT** ka matlab hai **JSON Web Token**. Yeh ek open standard (RFC 7519) hai jo do parties ke beech mein information ko ek **secure JSON object** ke roop mein transmit karne ke kaam aata hai.

Spring Boot aur Modern Web Apps mein JWT ka sabse bada use **Stateless Authentication** ke liye hota hai.



<https://www.jwt.io/>



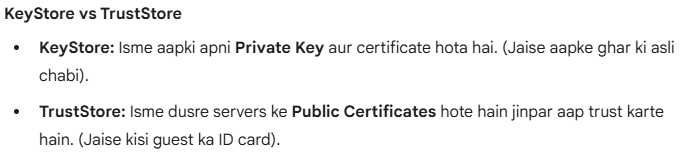


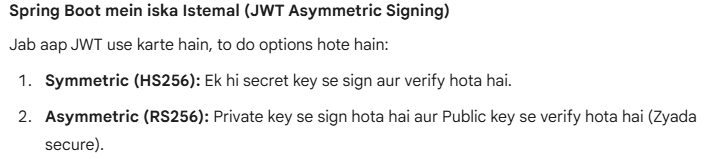
# The java keytool:

Java mein **keytool** ek command-line utility hai jo **KeyStore** (ek tarah ka digital locker) ko manage karne ke kaam aati hai. Ise JWT signing (Private/Public keys) aur HTTPS (SSL certificates) setup karne ke liye use kiya jata hai.

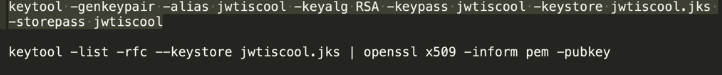
Spring Boot development mein iska sabse bada use **Self-signed certificates** aur **JWT asymmetric keys** banane ke liye hota hai.

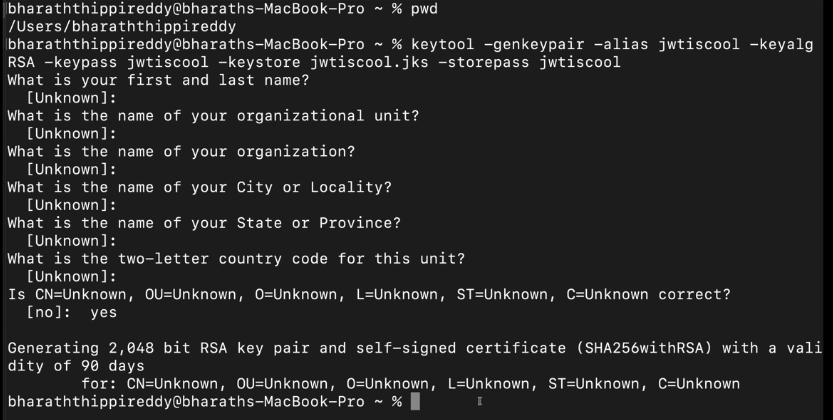






**Generate Asymmetric Key:**



It will generate jks key file in the folder :  


# Spring Authorization Server:

Spring Security mein **Authorization Server** ka matlab ek aisa dedicated service hai jo users ko authenticate karta hai aur OAuth2/OpenID Connect tokens issue karta hai.

Pehle Spring ne apne purane authorization server ko "deprecated" kar diya tha, lekin ab ek naya project **"Spring Authorization Server"** laya gaya hai jo Spring Boot 3 aur Java 17/21 ke liye optimize kiya gaya hai.

<https://spring.io/projects/spring-authorization-server>

