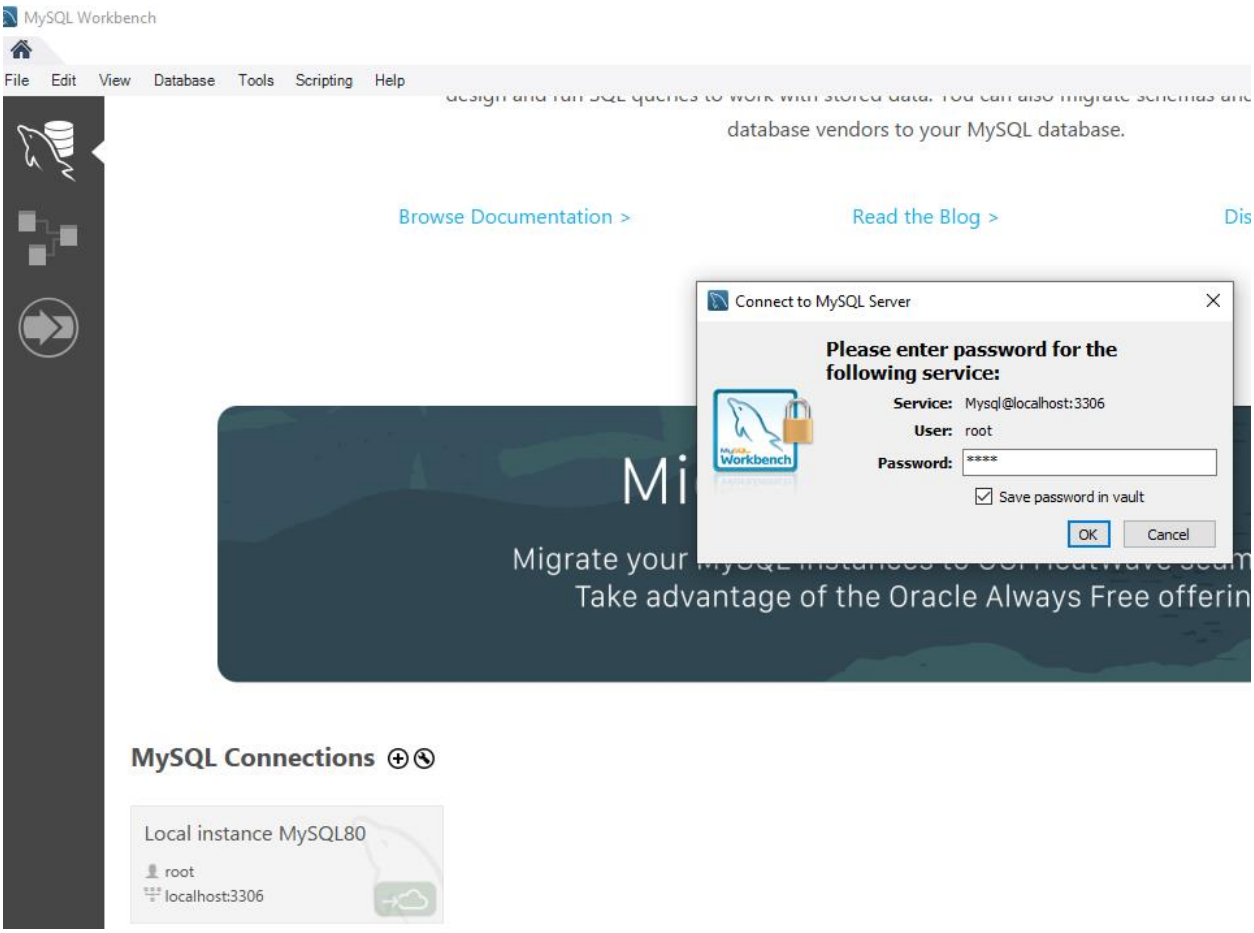
========== Spring Security Fundamentals =========

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.

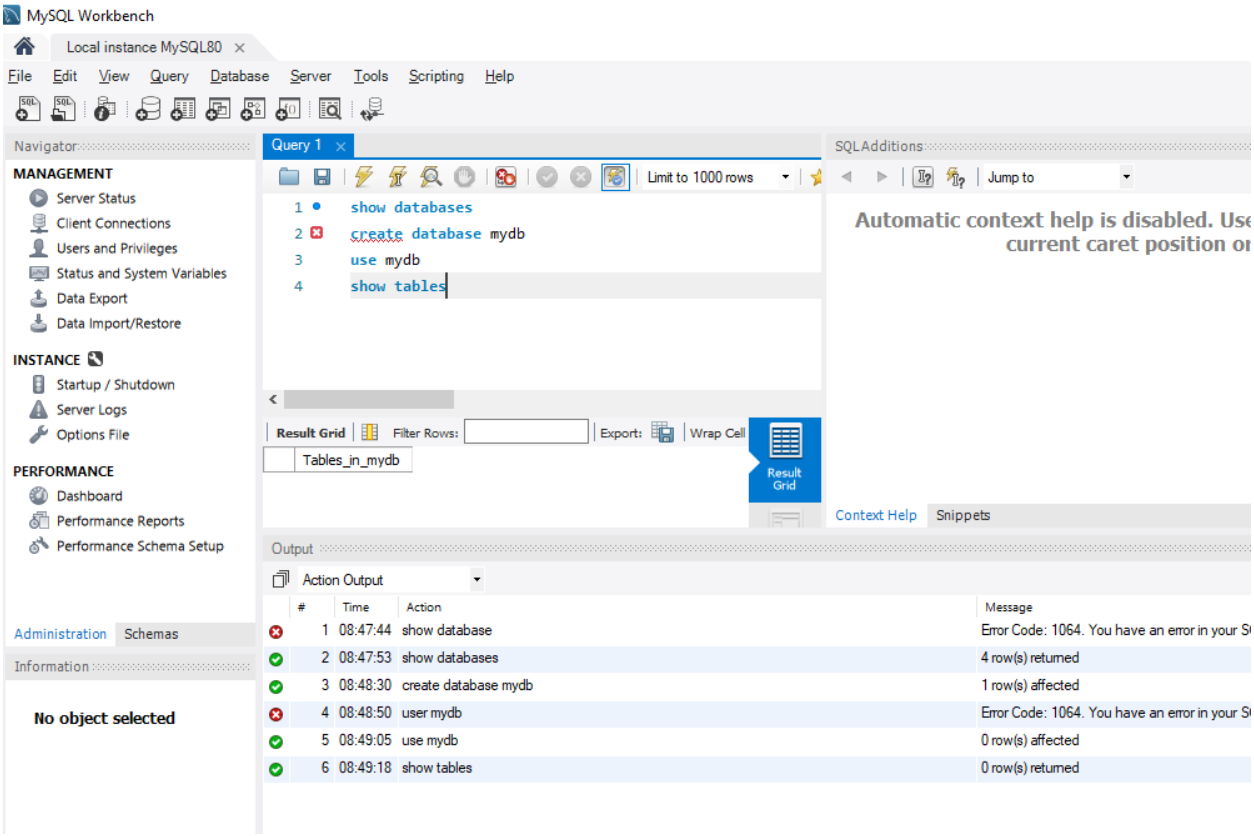


Once work bench client is install then create database(mydb) and then create tables like below:

Query:

1. **show databases**
2. **create database mydb**

3. **use mydb**
4. **show tables**



The Fundamentals:

What is Security:

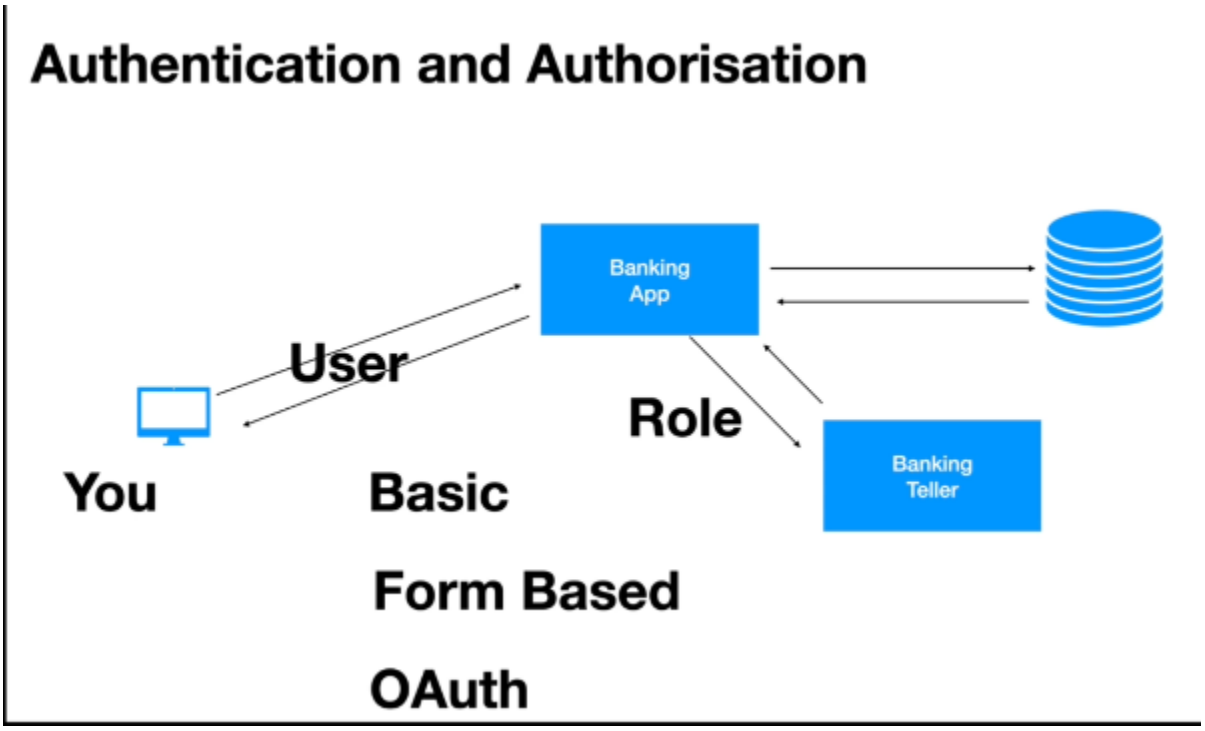
There are the two thing Authentication and Authorization:

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:

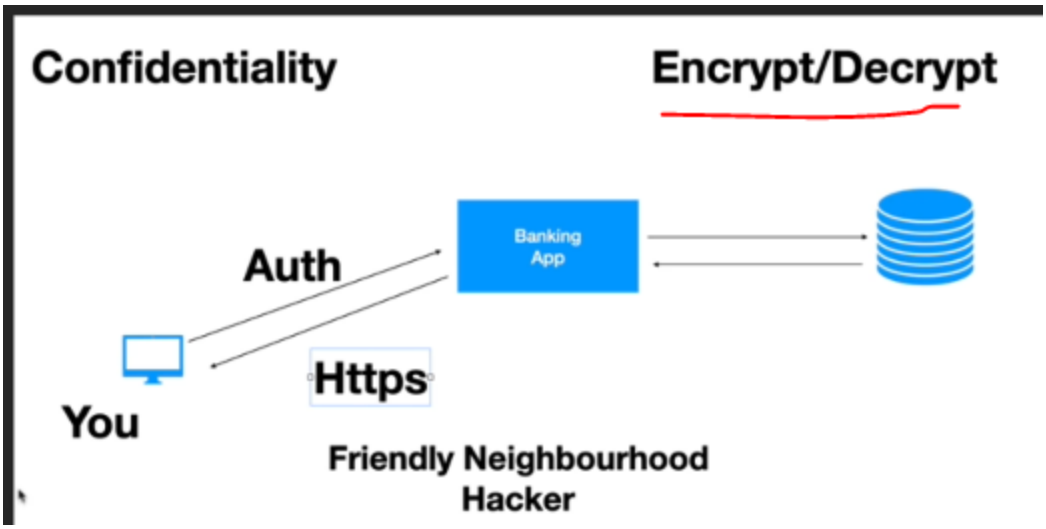
Means application(bank application) jo bhi data user ya client ko send kr rhi he vo hacker ke dwara hack to nhi kiya jaa rha yaa or data is no venerable to hacker.

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.



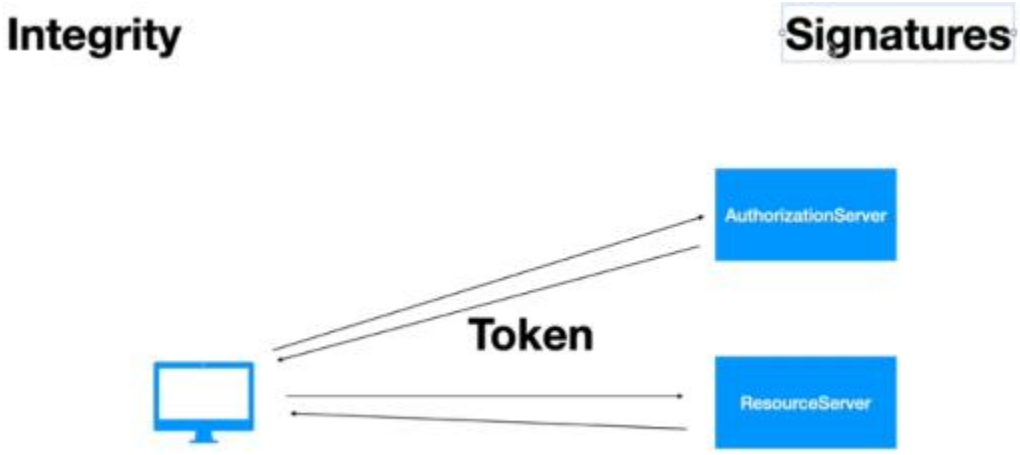
Integrity:
Means hacker ne data hack kiya he or us data me kuchh change kr ke bhej rha he to server re verity krega "**Signature**" ka use kr ke that is called integrity maintain he ki nhi application.
Means to maintain integrity we use Signature.

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.

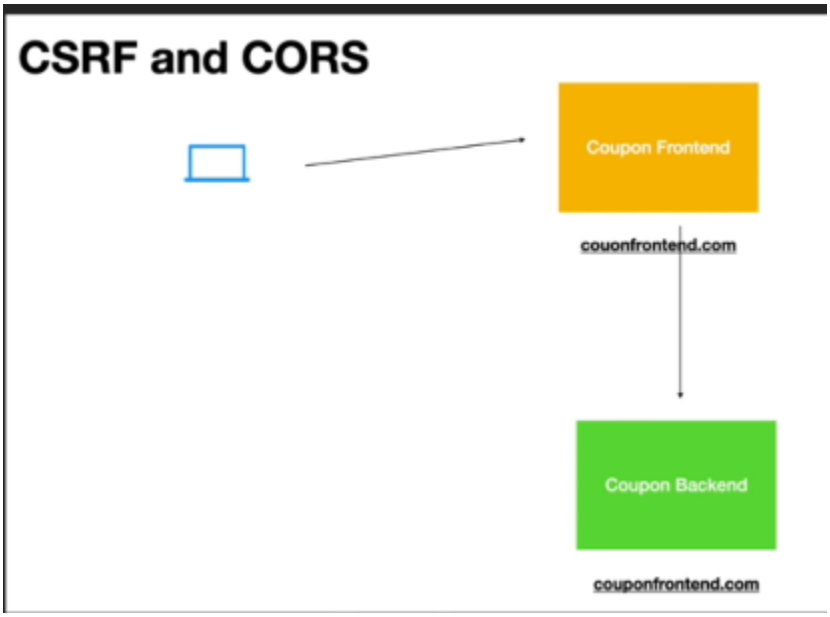
Resource server verify krta he that Token by the signature.

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.

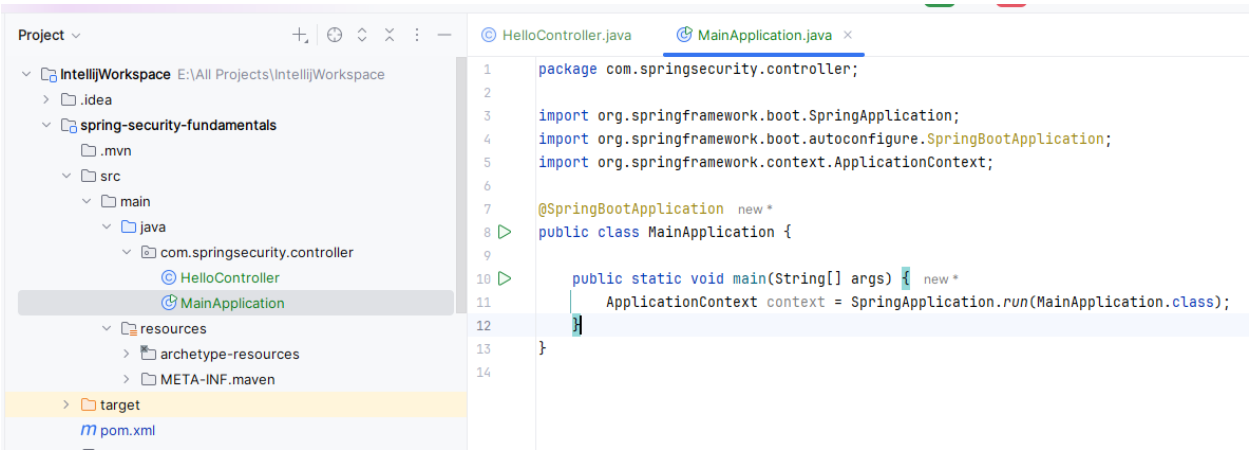
There are 6 components:

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.

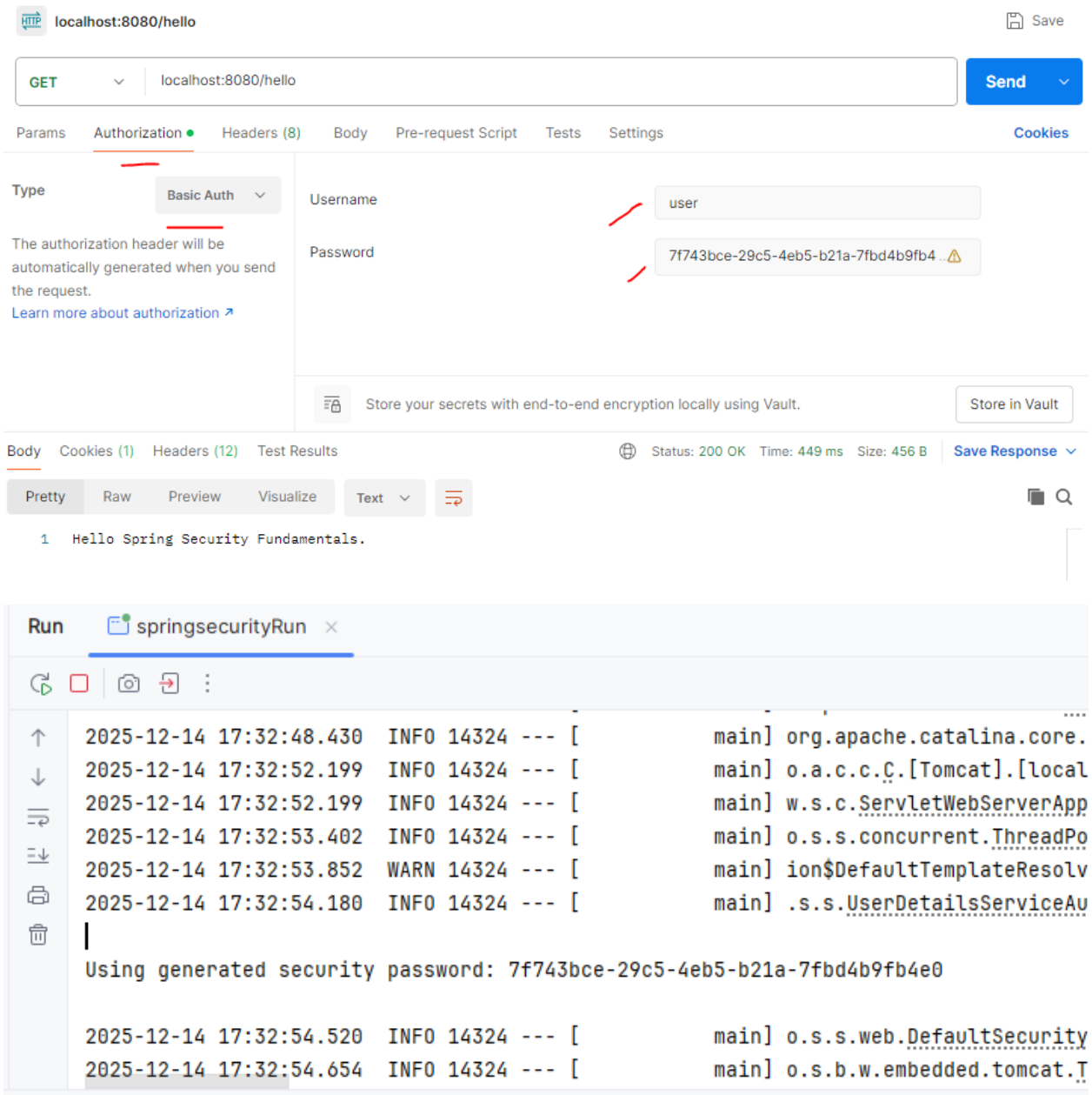


HelloController.java × MainApplication.java

```java
package com.springsecurity.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController  no usages  new *
public class HelloController{

    @GetMapping("/hello")  no usages  new *
    public String hello(){
        return "Hello Spring Security Fundamentals.";
    }

}
```

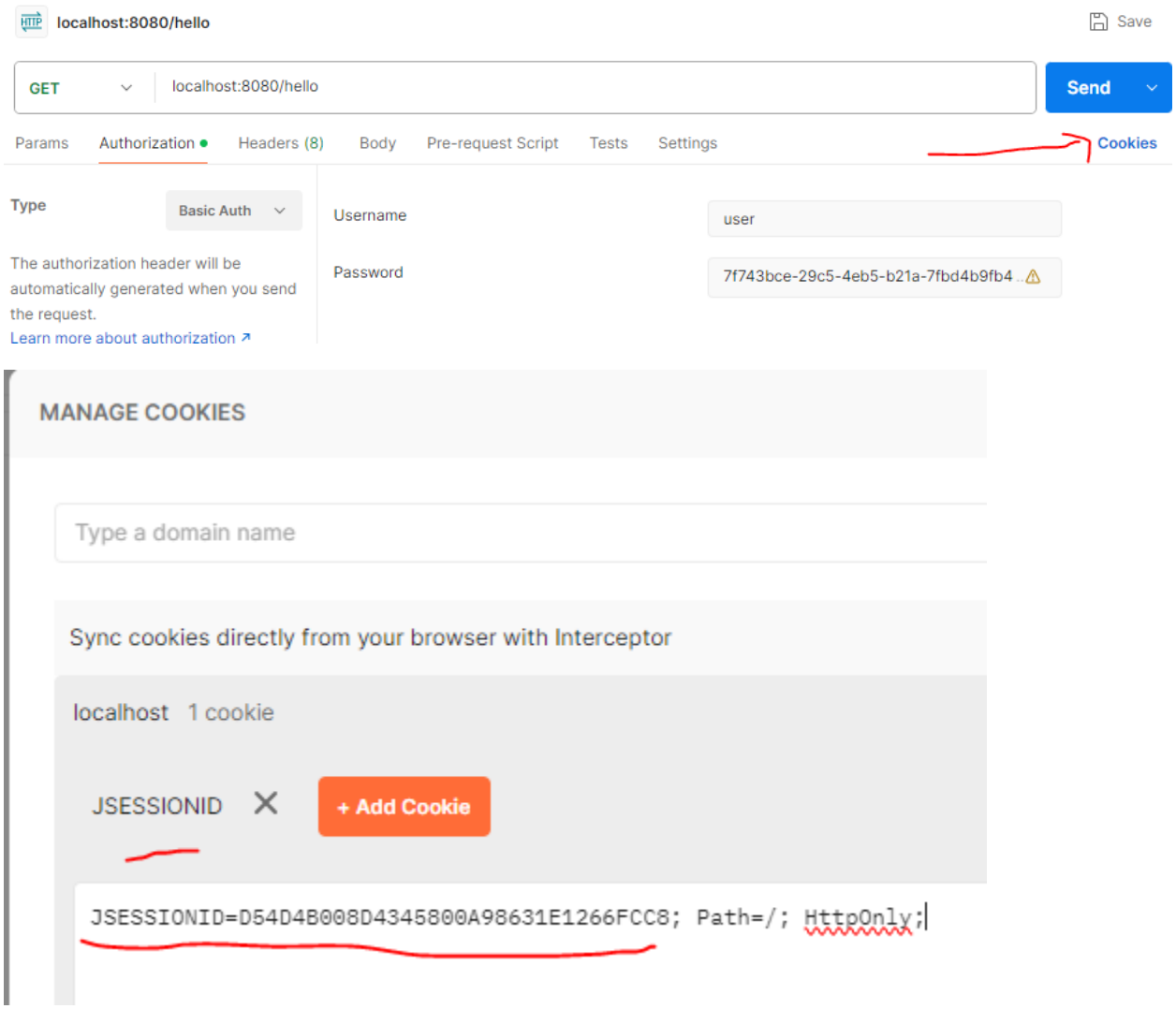
```xml
<dependencies>
    <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcprov-jdk15on</artifactId>
    <version>1.66</version>
</dependency>
</dependencies>
```

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:

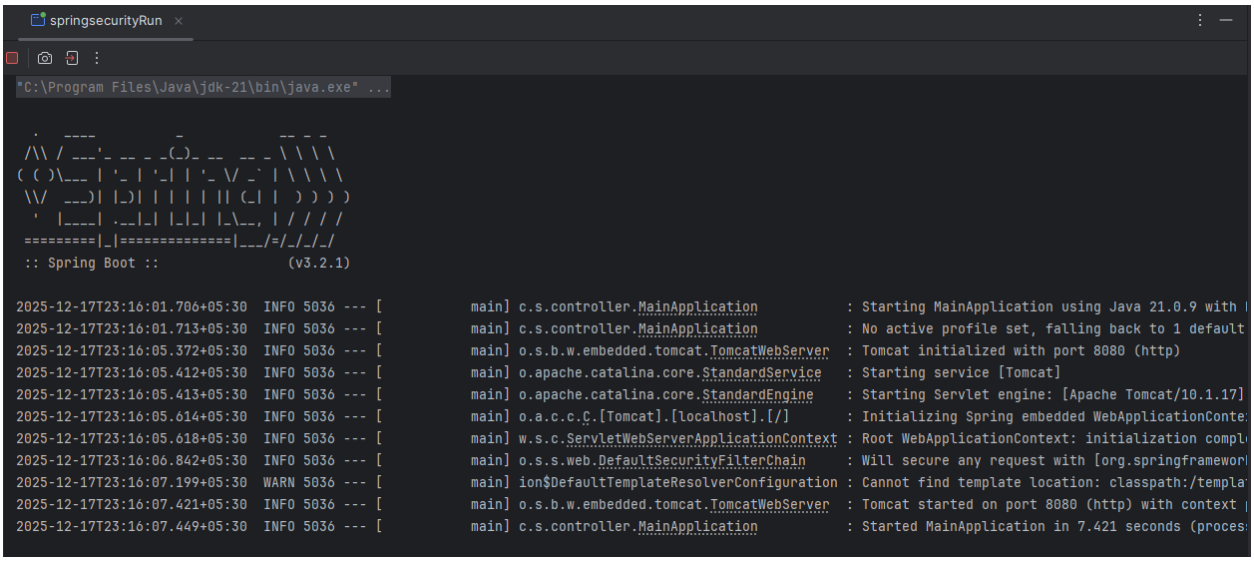
Create our custom MySecurityConfig.java in the same package where you have controller:

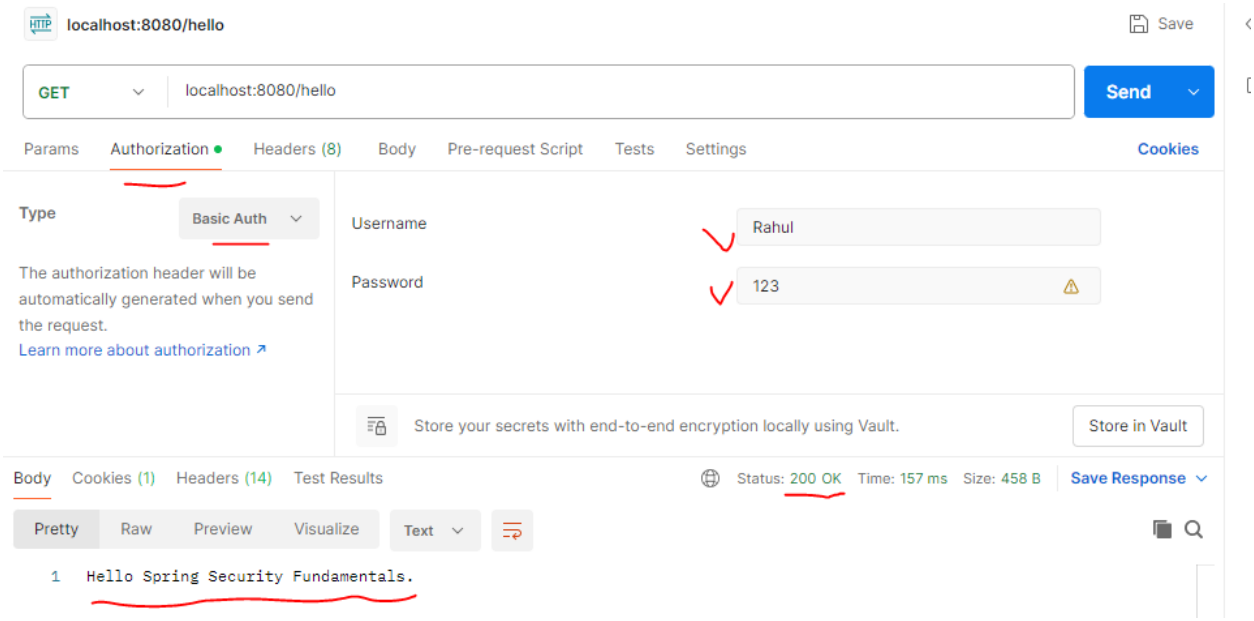
HelloController.java | MySecurityConfig.java | MainApplication.java

```java
package com.springsecurity.controller;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration  no usages  new *
public class MySecurityConfig {

    @Bean  no usages  new *
    UserDetailsService userDetailsService(){
        InMemoryUserDetailsManager userDetailsService = new InMemoryUserDetailsManager();
        UserDetails userDetails = User.withUsername("Rahul")
                .password(passwordEncoder().encode( rawPassword: "123")).authorities("read").build();
        userDetailsService.createUser(userDetails);
        return userDetailsService;
    }
```

```java
    @Bean  1 usage  new *
    PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Bean  no usages  new *
    SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception{
        httpSecurity.httpBasic(Customizer.withDefaults());
        httpSecurity.authorizeHttpRequests(
                AuthorizationManagerRequestMat... auth->auth.anyRequest().authenticated());
        return httpSecurity.build();

    }
}
```

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

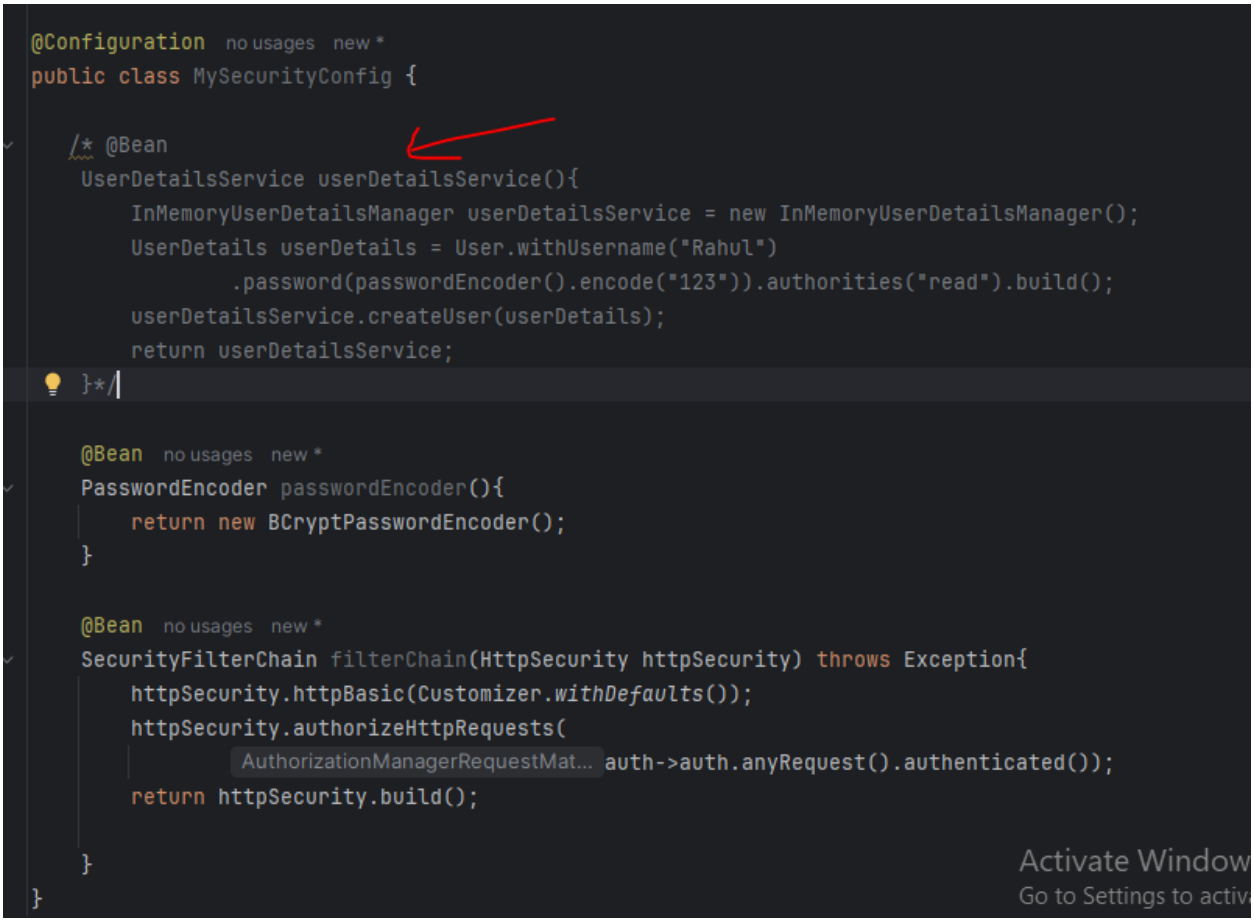




Create Custom AuthenticationProvider:

```java
import org.springframework.security.core.AuthenticationException;
import org.springframework.stereotype.Component;

import java.util.Arrays;

@Component  no usages  new *
public class MyAuthenticationProvider implements AuthenticationProvider {
    @Override  new *
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        String username = authentication.getName();
        String password = authentication.getCredentials().toString();

        if("Rahul".equals(username) && "123".equals(password)){
            return new UsernamePasswordAuthenticationToken(username,password, Arrays.asList());
        }else{
            throw new BadCredentialsException("Invalid username & password");
        }
    }

    @Override  new *
    public boolean supports(Class<?> authentication) {
        return authentication.equals(UsernamePasswordAuthenticationToken.class);
    }
}
```

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



```
@Configuration  no usages  new *
public class MySecurityConfig {

    /* @Bean
     UserDetailsService userDetailsService(){
         InMemoryUserDetailsManager userDetailsService = new InMemoryUserDetailsManager();
         UserDetails userDetails = User.withUsername("Rahul")
                 .password(passwordEncoder().encode("123")).authorities("read").build();
         userDetailsService.createUser(userDetails);
         return userDetailsService;
     }*/

    @Bean  no usages  new *
    PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Bean  no usages  new *
    SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception{
        httpSecurity.httpBasic(Customizer.withDefaults());
        httpSecurity.authorizeHttpRequests(
                AuthorizationManagerRequestMat... auth->auth.anyRequest().authenticated());
        return httpSecurity.build();

    }
}
```

localhost:8080/hello

GET localhost:8080/hello

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Type Basic Auth

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

Username Rahul

Password 123

Store your secrets with end-to-end encryption locally using Vault.

Body Cookies (1) Headers (14) Test Results Status: 200 OK Time: 34 ms

Pretty Raw Preview Visualize Text

```
1  Hello Spring Security Fundamentals.
```

localhost:8080/hello

GET localhost:8080/hello

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Type Basic Auth

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

Username Rahul

Password 123345

Store your secrets with end-to-end encryption locally using Vault.

Body Cookies (1) Headers (14) Test Results Status: 401 Unauthorized Time: 12 ms Size:

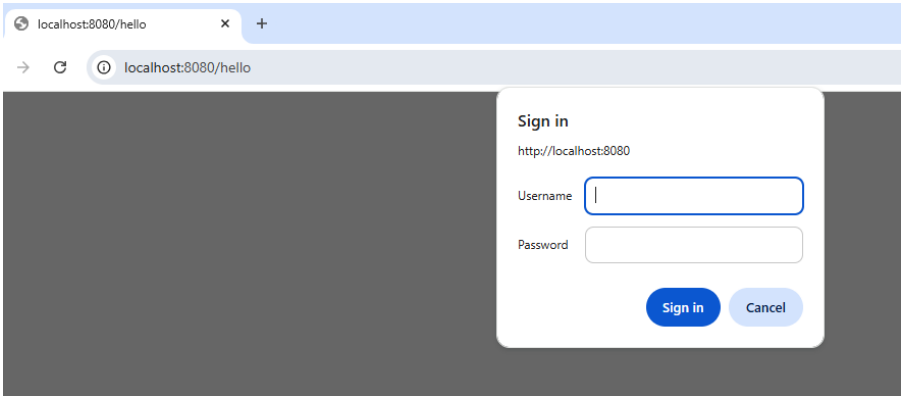
Pretty Raw Preview Visualize Text

```
1
```

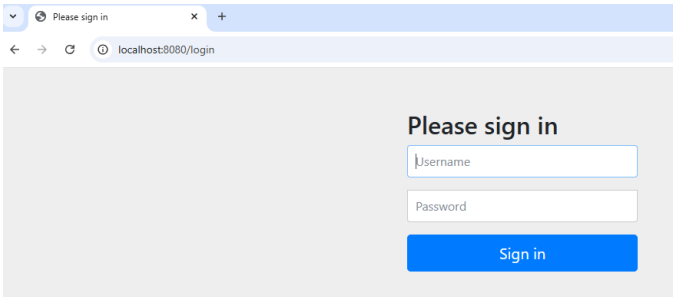
Form Based Login :

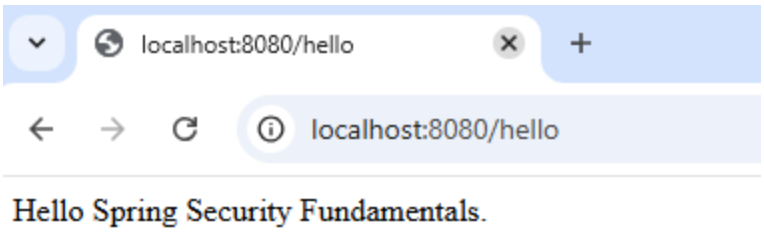
```
public class MySecurityConfig {
        userDetailsService.createUser(userDetails);
        return userDetailsService;
    }*/

    @Bean  no usages  new *
    PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Bean  no usages  new *
    SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception{
        httpSecurity.formLogin(Customizer.withDefaults());
        httpSecurity.authorizeHttpRequests(
                AuthorizationManagerRequestMat... auth->auth.anyRequest().authenticated());
        return httpSecurity.build();

    }
}
```

Let see login form in browser:



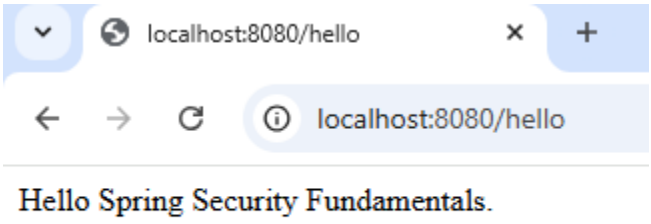
Or



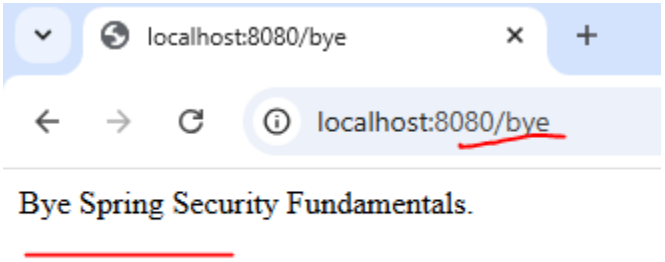


Few More Methods (permit):

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



Now hitting /bye api :



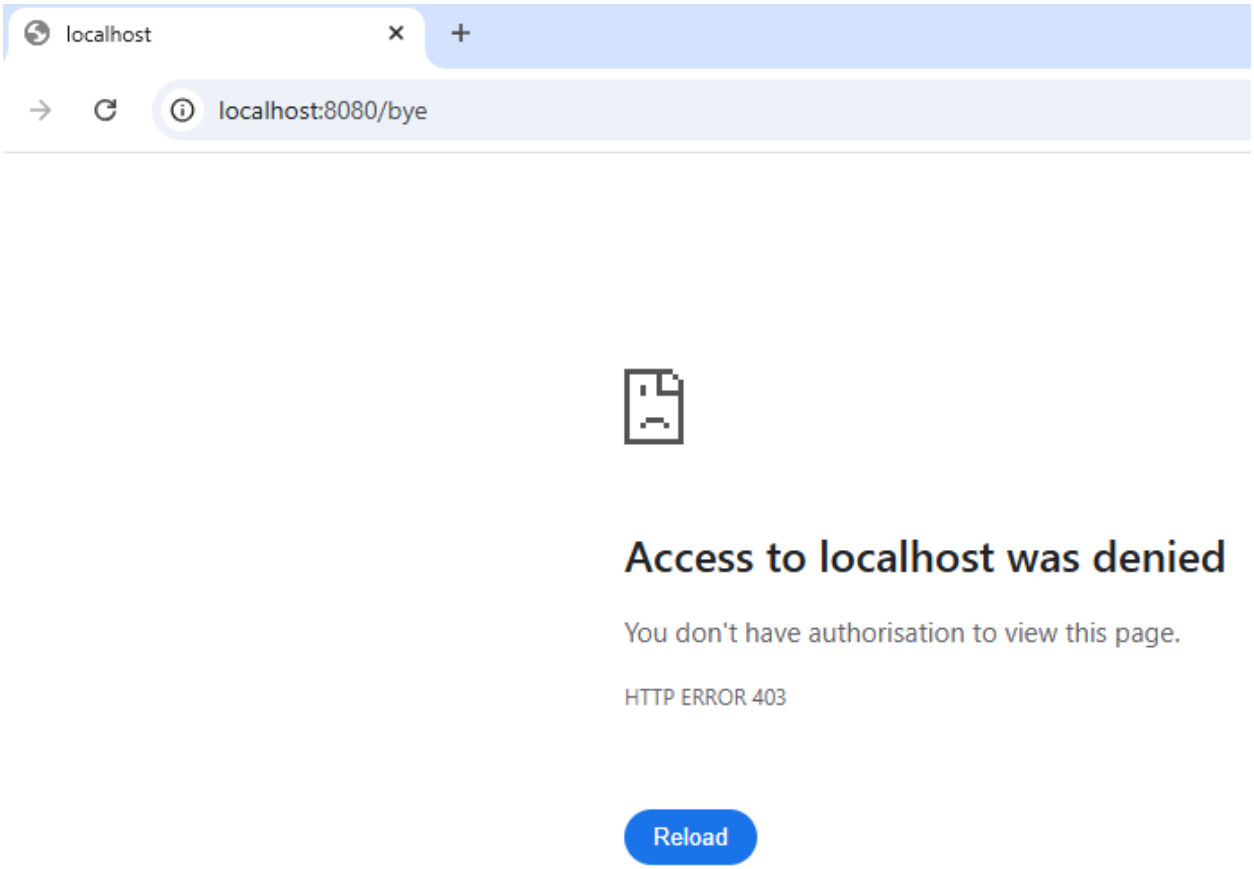
In this case its still fetching data - . means all request are accessible due to below code: anyRequest() method

```java
public class MySecurityConfig {

    @Bean  no usages  new *
    PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Bean  no usages  new *
    SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception{
        httpSecurity.formLogin(Customizer.withDefaults());
        httpSecurity.authorizeHttpRequests(
                AuthorizationManagerRequestMat... auth->auth.anyRequest().authenticated());
        return httpSecurity.build();

    }
}
```

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

```java
public class MySecurityConfig {                                                    7  1

    @Bean  no usages  new *
    PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Bean  no usages  new *
    SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception{
        httpSecurity.formLogin(Customizer.withDefaults());
        httpSecurity.authorizeHttpRequests(
                AuthorizationManagerRequestMat... auth->auth.requestMatchers( ...patterns: "/hello").authenticated());
        return httpSecurity.build();

    }
}
```

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:

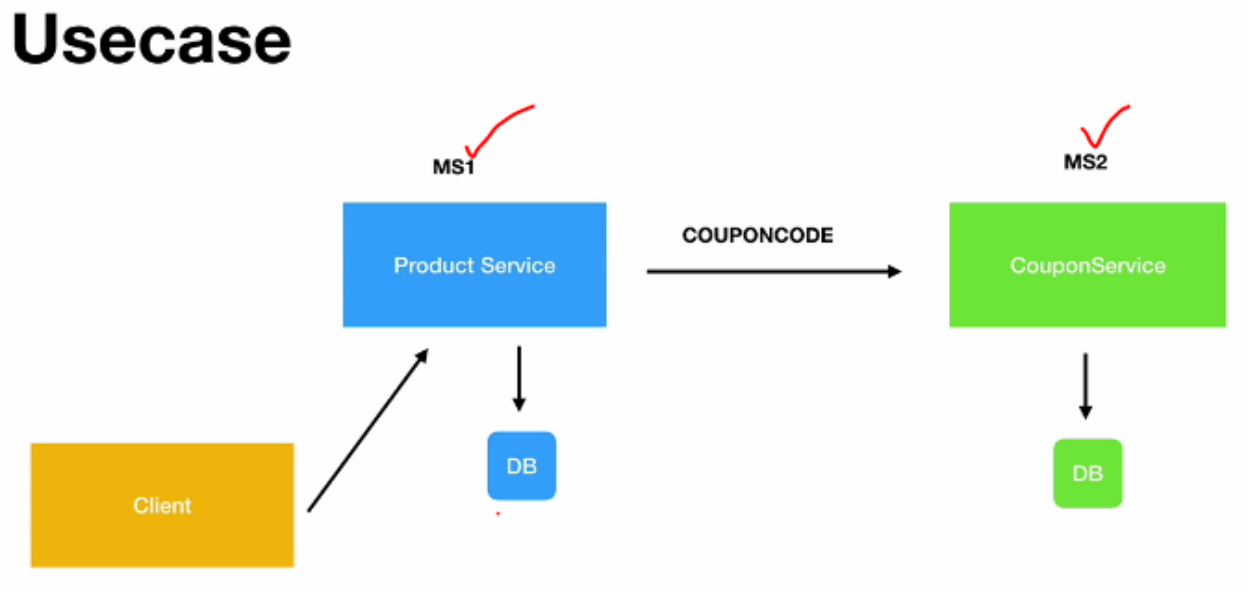
```
public class MySecurityFilter extends GenericFilterBean {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        System.out.println("Before");
        chain.doFilter(request, response);
        System.out.println("After");
    }

}
```

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

```java
public class MySecurityFilter extends OncePerRequestFilter {


    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        // TODO Auto-generated method stub

    }

}
```

Create Microservices:

Usercase:



DataBase:

use mydb;

create table product(

id int AUTO_INCREMENT PRIMARY KEY,

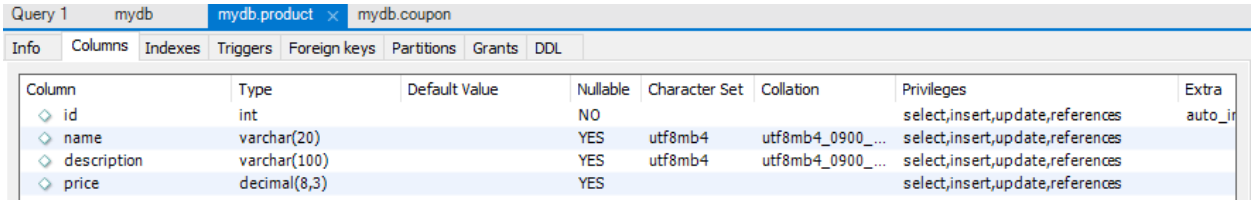
name varchar(20),

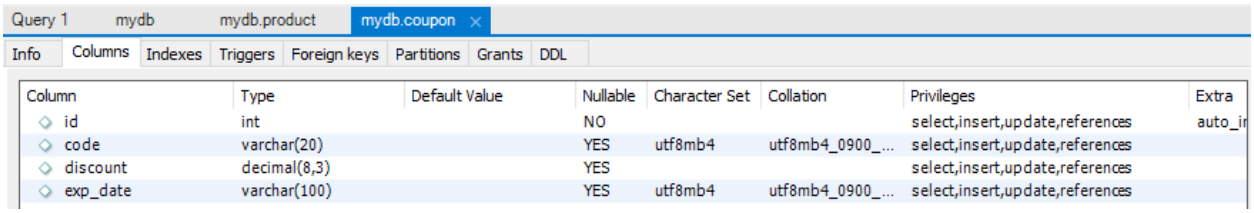
description varchar(100),

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/

```java
package com.couponservice.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

import java.math.BigDecimal;

@Entity  7 usages  new *
public class Coupon {
    @Id  3 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String code;  3 usages
    private String expDate;  3 usages
    private BigDecimal discount;  3 usages

    public Long getId() {  no usages  new *
```

```java
package com.couponservice.repo;

import com.couponservice.model.Coupon;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CouponRepo extends JpaRepository<Coupon, Long> {
    Coupon findByCode(String code);  1 usage  new *
}
```

```java
package com.couponservice.controller;

import com.couponservice.model.Coupon;
import com.couponservice.repo.CouponRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController  no usages  new *
@RequestMapping("/couponapi")
public class CouponController {

    @Autowired  2 usages
    private CouponRepo repo;

    @RequestMapping(value = "/coupon", method = RequestMethod.POST)  no usage
    public Coupon create(@RequestBody Coupon coupon){
        return repo.save(coupon);
    }


    @RequestMapping(value = "/coupons/{code}", method = RequestMethod.GET)
    public Coupon get(@PathVariable("code") String code){
        return repo.findByCode(code);
    }
}
```

Coupon.java | CouponRepo.java | CouponController.java | application.properties

```
1  spring.datasource.url=jdbc:mysql://localhost:3306/mydb
2  spring.datasource.username=root
3  spring.datasource.password=root
```

upon.java × CouponRepo.java CouponController.java application.properties pom.xml (couponservice) ×

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-autoconfigure</artifactId>
        </dependency>
    </dependencies>
```

localhost:8080/couponapi/coupon

POST localhost:8080/couponapi/coupon

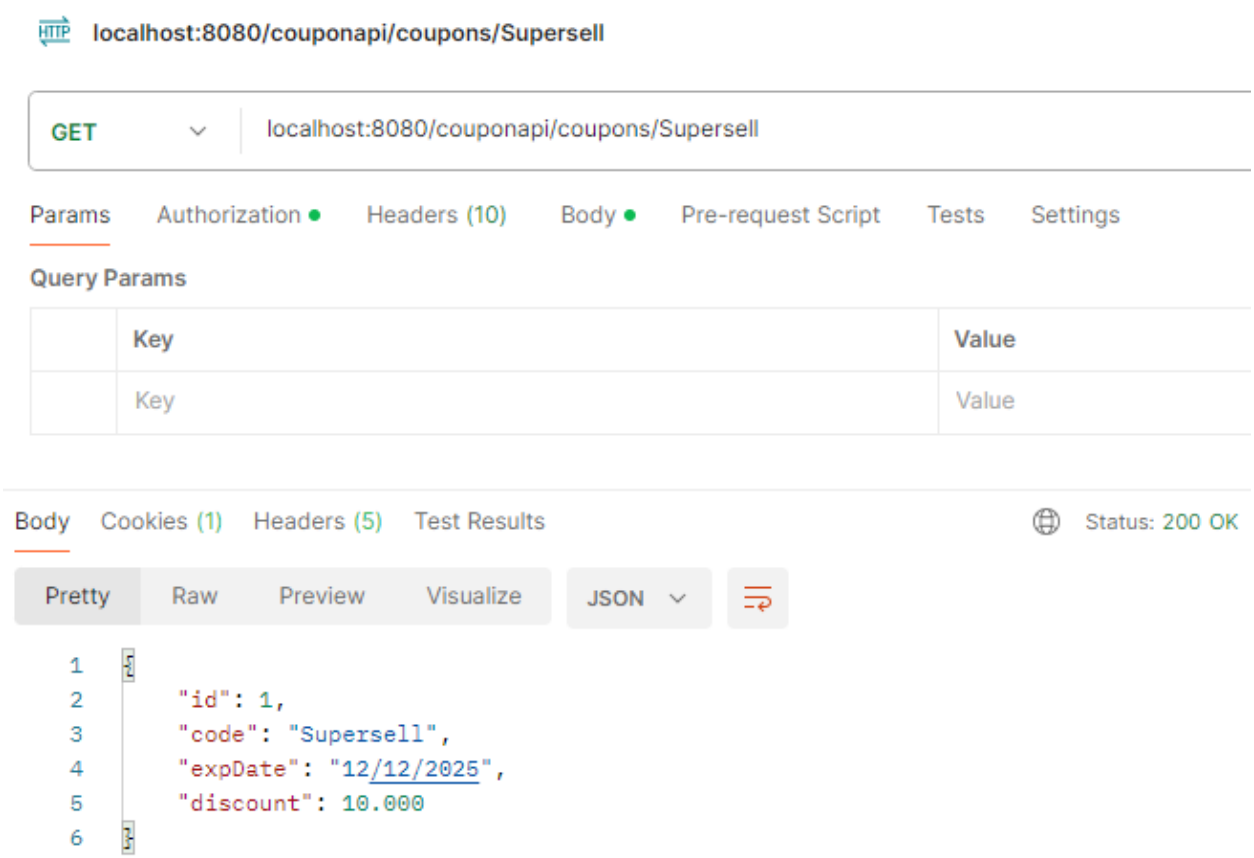
Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```json
{
    "code":"Supersell",
    "discount":10,
    "expDate":"12/12/2025"
}
```

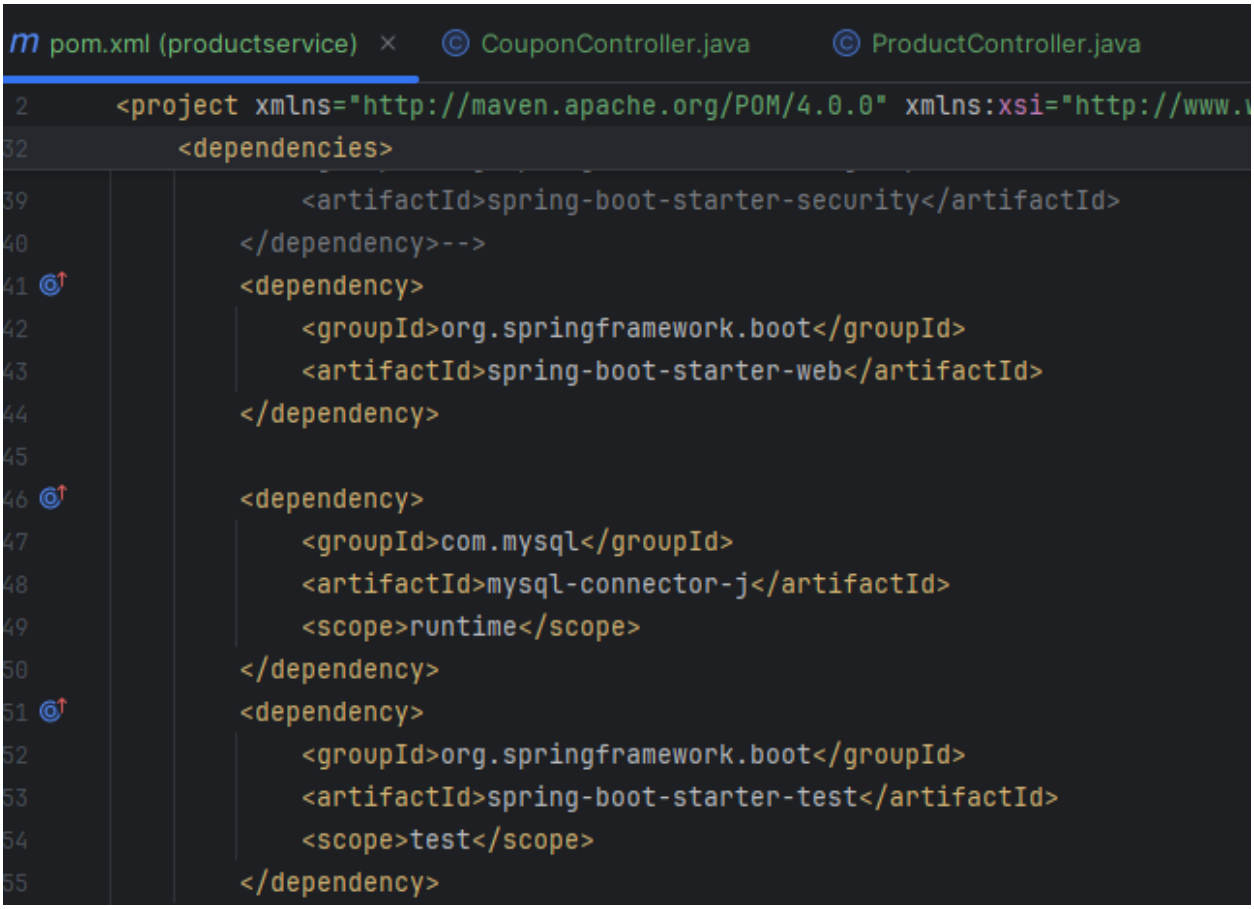
Body Cookies Headers Test Results Status: 200 OK Time: 1945 ms S

Pretty Raw Preview Visualize Text



Create product service module:

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

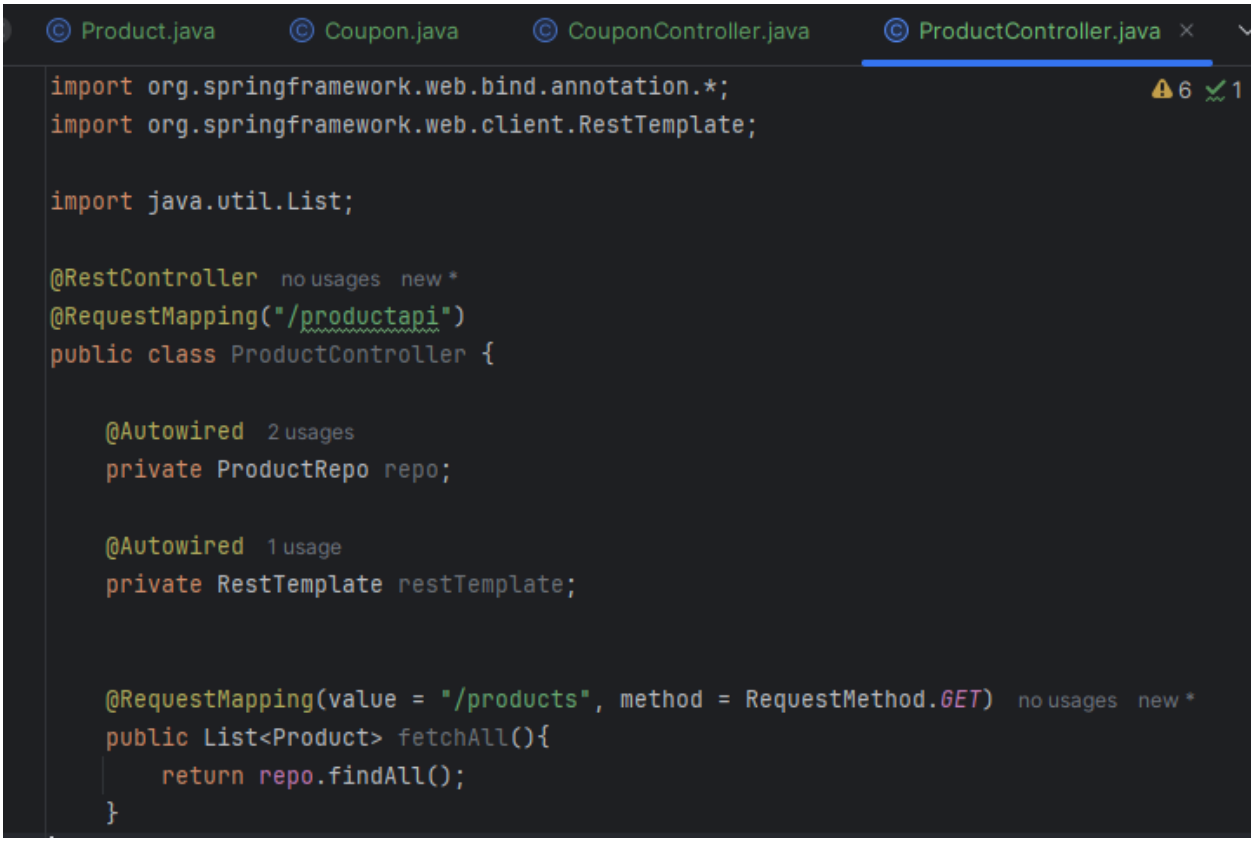

pom.xml (productservice)
CouponController.java
ProductController.java
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
<dependencies>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>-->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>

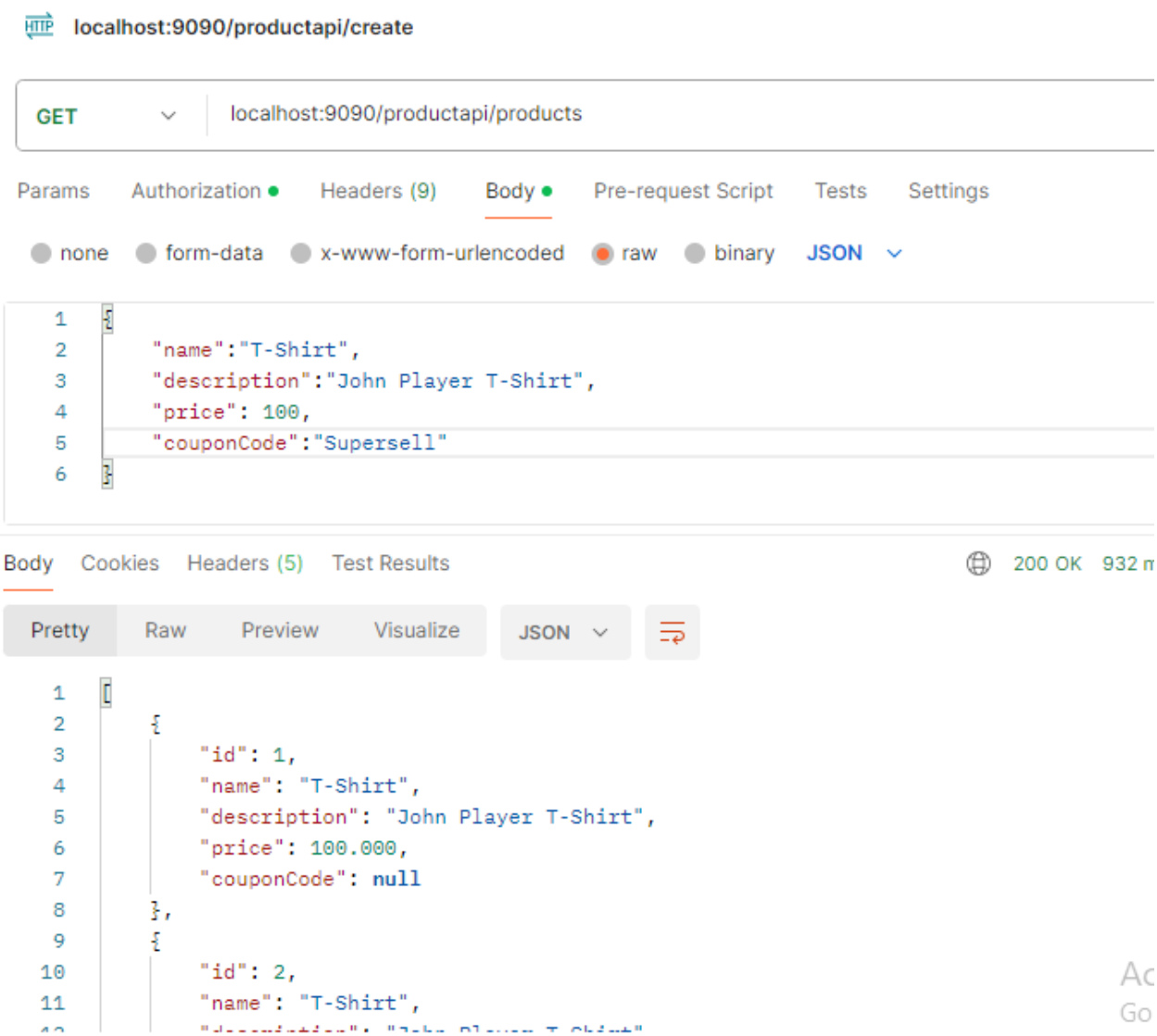
```java
package com.productservice;

import ...

@SpringBootApplication  new *
public class ProductserviceApplication {


    @Bean  no usages  new *
    public RestTemplate restTemplate() { return new RestTemplate(); }

    public static void main(String[] args) {  new *
        SpringApplication.run(ProductserviceApplication.class, args);
    }

}
```

```java
package com.productservice.repository;

import com.productservice.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepo extends JpaRepository<Product,Long> {  2 usages  new *

}
```

```java
package com.productservice.model;

import jakarta.persistence.*;
import org.springframework.web.bind.annotation.RequestMapping;

import java.math.BigDecimal;

@Entity  5 usages  new *
public class Product {

    @Id  2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;  2 usages
    private String description;  2 usages
    private BigDecimal price;  2 usages

    @Transient  2 usages
    private String couponCode;
```

```java
package com.productservice.model;

import ...

public class Coupon {  3 usages  new *

    private Long id;  3 usages
    private String code;  3 usages
    private String expDate;  3 usages
    private BigDecimal discount;  3 usages
```


Product.java
Coupon.java
CouponController.java
ProductController.java
6 1
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;
import java.util.List;
@RestController no usages new *
@RequestMapping("/productapi")
public class ProductController {
@Autowired 2 usages
private ProductRepo repo;
@Autowired 1 usage
private RestTemplate restTemplate;
@RequestMapping(value = "/products", method = RequestMethod.GET) no usages new *
public List<Product> fetchAll(){
return repo.findAll();
}



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

ProductserviceApplication.java | ProductRepo.java | Product.java | Coupon.java | CouponController.java | ProductController.java

```java
public class ProductController {
    private ProductRepo repo;

    @Autowired  1 usage
    private RestTemplate restTemplate;


    @RequestMapping(value = "/products", method = RequestMethod.GET)  no usages  new *
    public List<Product> fetchAll(){
        return repo.findAll();
    }

    @PostMapping("/create")  no usages  new *
    public Product create(@RequestBody Product request){

        Coupon coupon = restTemplate.getForObject( url: "http://localhost:9091/couponapi/coupons/"+request.getCouponCode(), Coupon.class);

        request.setPrice(request.getPrice().subtract(coupon.getDiscount()));
        return repo.save(request);
    }

}
```

localhost:9090/productapi/create

POST localhost:9090/productapi/create

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | JSON

```
1  {
2      "name":"Pent",
3      "description":"John Player Pent",
4      "price": 200,
5      "couponCode":"Supersell"
6  }
```

Body | Cookies | Headers (5) | Test Results | Status: 200 OK | Tim

Pretty | Raw | Preview | Visualize | JSON

```
1  {
2      "id": 3,
3      "name": "Pent",
4      "description": "John Player Pent",
5      "price": 190.000,
6      "couponCode": "Supersell"
7  }
```

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

(

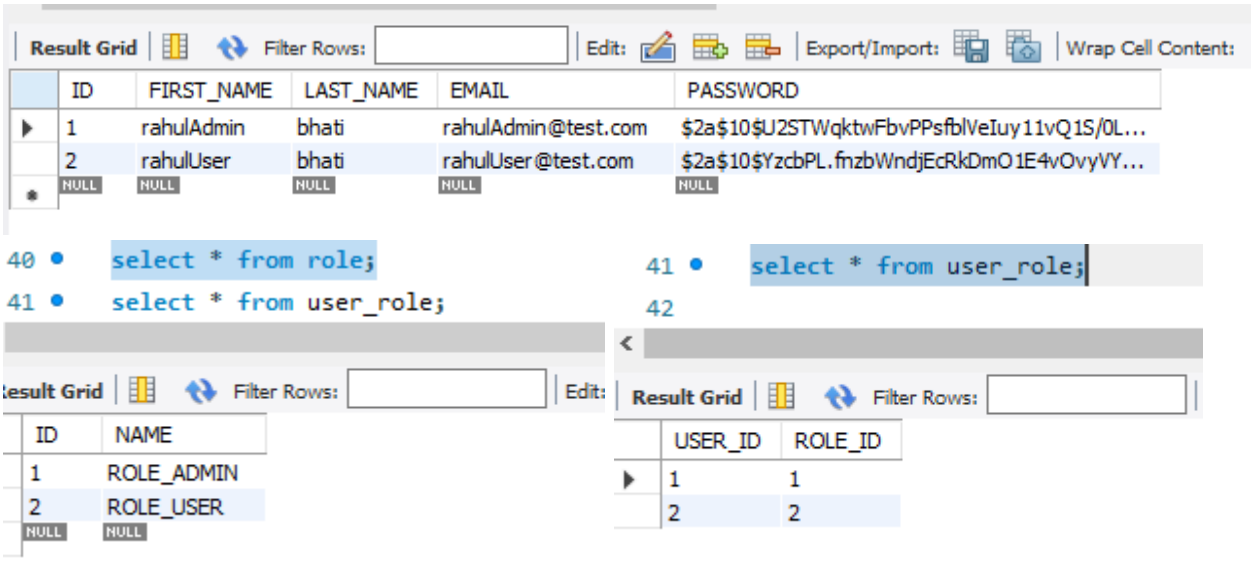
ID INT NOT NULL AUTO_INCREMENT,

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/0LYLeXQf1ZL0c
MXc9HuTEA2');

insert into user(first_name,last_name,email,password) values
('john','ferguson','john@ferguson.com','$2a$10$YzcbPL.fnzbWndjEcRkDmO1E4vOvyVYP5kLsJvtZ
nR1f8nlXjvq/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:

```java
package com.couponservice.model;

import jakarta.persistence.*;

import java.util.Set;

@Entity  6 usages  new *
public class User {

    @Id  2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstName;  2 usages
    private String lastName;  2 usages
    private String email;  2 usages
    private String password;  2 usages

    @ManyToMany(fetch = FetchType.EAGER)  2 usages
    @JoinTable(name = "user_role", joinColumns = @JoinColumn(name="user_id")
            ,inverseJoinColumns = @JoinColumn(name = "role_id") )
    private Set<Role> roles;
```

```java
import jakarta.persistence.*;
import org.springframework.security.core.GrantedAuthority;

import java.util.Set;

@Entity  5 usages  new *
public class Role implements GrantedAuthority {

    @Id  2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;  3 usages

    @ManyToMany(mappedBy = "roles")  no usages
    private Set<User> users;

    @Override  new *
    public String getAuthority() {
        return name;
    }

    public Long getId() {  no usages  new *
        return id;
    }
```

Custom UserDetailsService

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

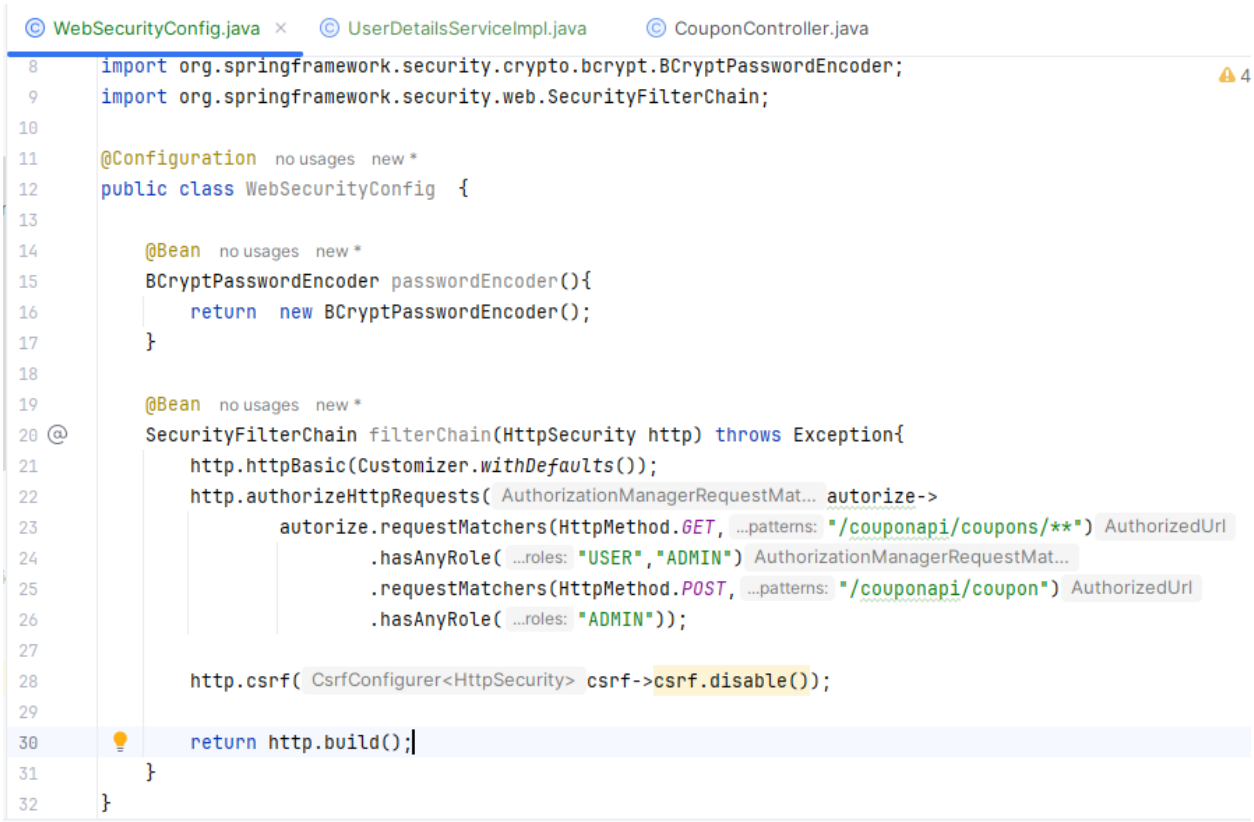
Kyoki spring boot ko springboot ka User class chahiye.

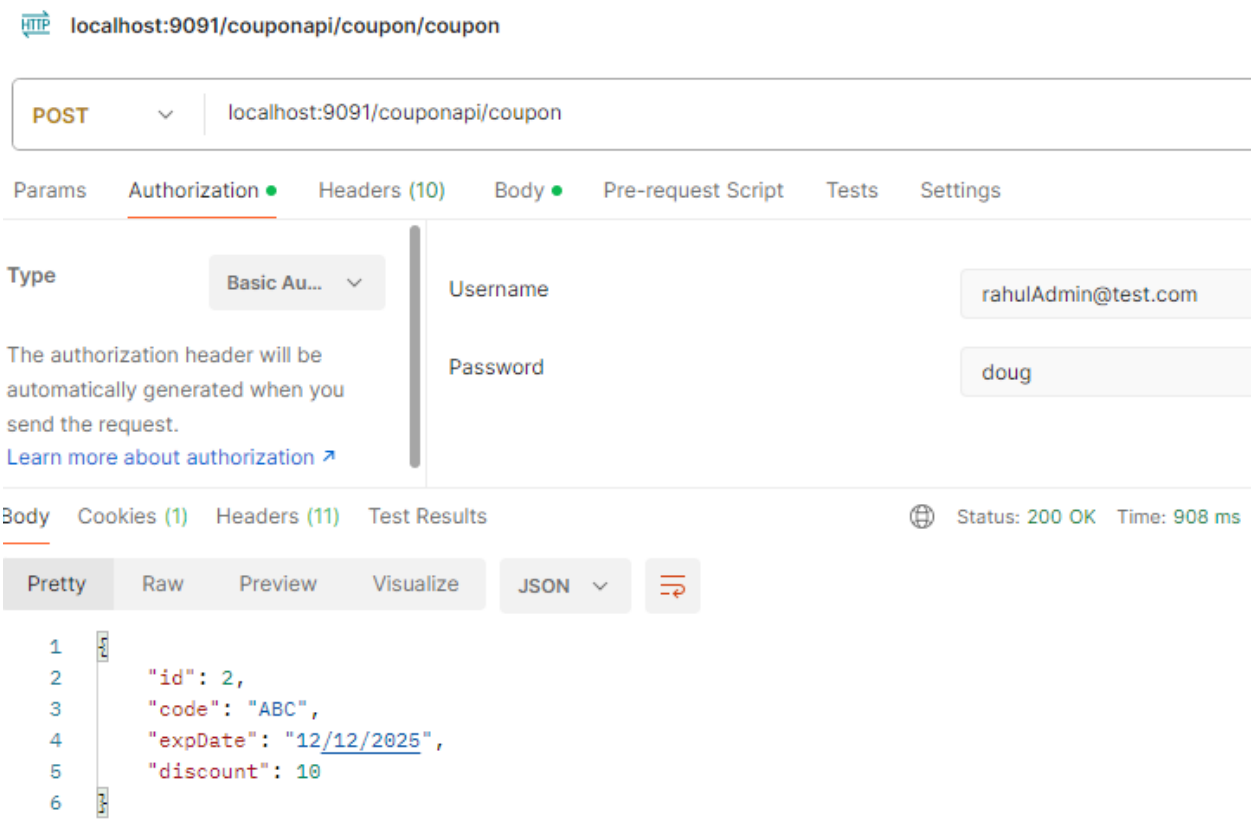
Isliye hm email(username) pass kr ke data get kr rhe he database me se or pass work rhe he spring boot User class ko.

```java
import org.springframework.stereotype.Service;

@Service  no usages  new *
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired  1 usage
    private UserRepo repo;

    @Override  no usages  new *
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        User user = repo.findByEmail(username);
        if(user==null){
            throw new UsernameNotFoundException("Username not found!!!");
        }
        // Ye krne se hme spring security ko UserDetails ka object bna kr return kiya.
        return new org.springframework.security.core.userdetails.User(
                user.getEmail(),user.getPassword(),user.getRoles());
    }
}
```

Code change

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



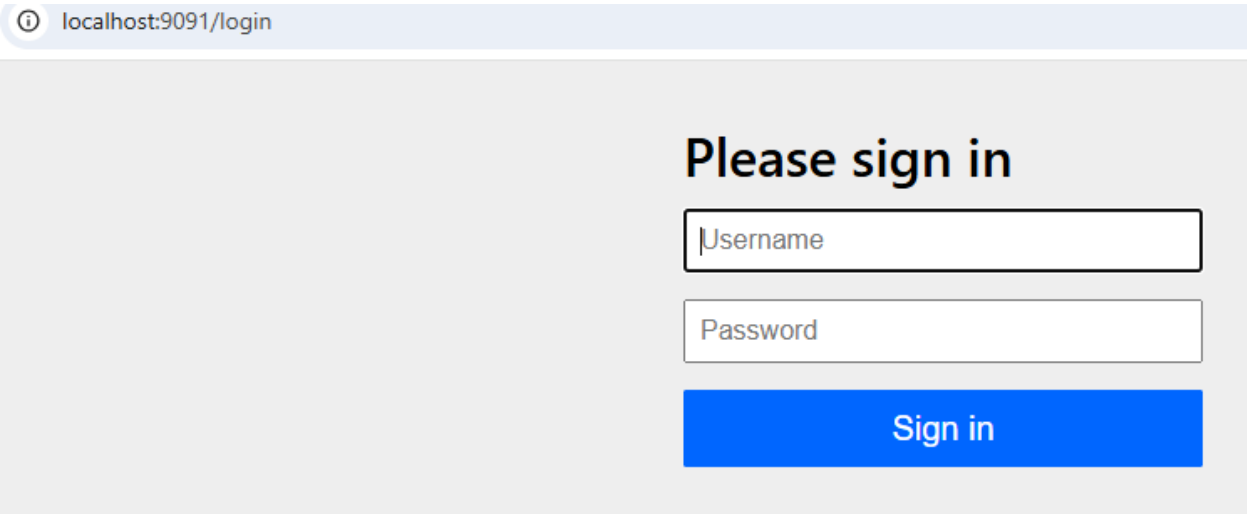


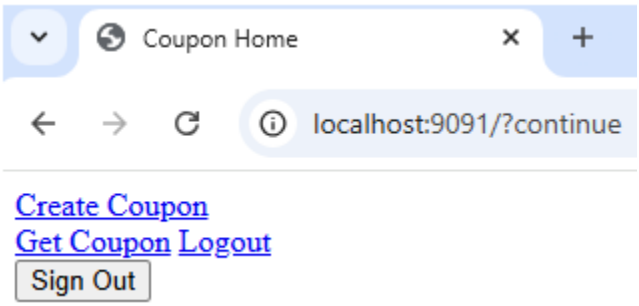
Secure a WebApp:

Thymeleaf Dependency:

```
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
```

```
public class WebSecurityConfig  {
    return  new BCryptPasswordEncoder();
}

@Bean  no usages  Rahul Bhati *
SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
    http.formLogin(Customizer.withDefaults());
    http.authorizeHttpRequests( AuthorizationManagerRequestMat... autorize->
            autorize.requestMatchers(HttpMethod.GET, ...patterns: "/couponapi/coupons/**","/") AuthorizedUrl
                    .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
                    .requestMatchers(HttpMethod.POST, ...patterns: "/couponapi/coupon") AuthorizedUrl
                    .hasAnyRole( ...roles: "ADMIN"));

    http.csrf( CsrfConfigurer<HttpSecurity> csrf->csrf.disable());

    return http.build();
```





Till now we have implemented login and index page.

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

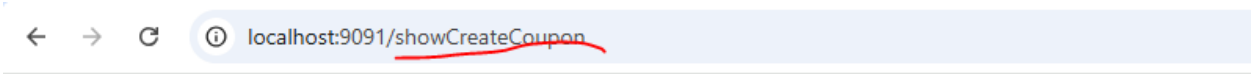
Admin only can create coupon and save it.

User role only able to get the coupon details.

```java
@RestController  no usages  Rahul Bhati *
//@RequestMapping("/couponapi")
public class CouponController {

    @Autowired  2 usages
    private CouponRepo repo;

    @GetMapping("/showCreateCoupon")  no usages  new *
    public ModelAndView showCreateCoupon() {
        ModelAndView mav = new ModelAndView( viewName: "createCoupon");
        return mav;
    }

    @PostMapping("/saveCoupon")  no usages  new *
    public ModelAndView save(Coupon coupon) {
        repo.save(coupon);
        ModelAndView mav = new ModelAndView( viewName: "createResponse");
        return mav;
    }

    @GetMapping("/showGetCoupon")  no usages  new *
    public ModelAndView showGetCoupon() {
        ModelAndView mav = new ModelAndView( viewName: "getCoupon");
        return mav;
    }
    @PostMapping("/getCoupon")  no usages  new *
    public ModelAndView getCoupon(String code) {
        ModelAndView mav = new ModelAndView( viewName: "couponDetails");
        System.out.println(code);
        mav.addObject(repo.findByCode(code));
        return mav;
    }
}
```

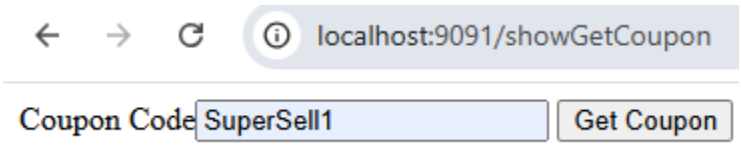
```java
@Configuration  no usages  Rahul Bhati *
public class WebSecurityConfig  {

    @Bean  no usages  Rahul Bhati
    BCryptPasswordEncoder passwordEncoder(){
        return  new BCryptPasswordEncoder();
    }

    @Bean  no usages  Rahul Bhati *
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
        http.formLogin(Customizer.withDefaults());
        http.authorizeHttpRequests( AuthorizationManagerRequestMat... autorize->
                autorize.requestMatchers(HttpMethod.GET, ...patterns: "/couponapi/coupons/**","/") AuthorizedUrl
                        .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.GET, ...patterns: "/showCreateCoupon","/createCoupon","/createResponse")
                        .hasAnyRole( ...roles: "ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.POST, ...patterns: "/couponapi/coupon","/saveCoupon") AuthorizedUrl
                        .hasAnyRole( ...roles: "ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.POST, ...patterns: "/getCoupon") AuthorizedUrl
                        .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.GET, ...patterns: "/showGetCoupon","/couponDetails") AuthorizedUrl
                        .hasAnyRole( ...roles: "USER","ADMIN")) ;

        http.csrf( CsrfConfigurer<HttpSecurity> csrf->csrf.disable());

        return http.build();
    }
}
```

Admin Role can create role:

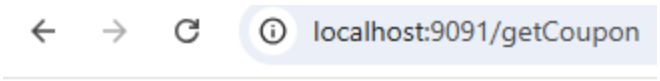


Create Coupon

Code: Discount: Expiry Date: Save



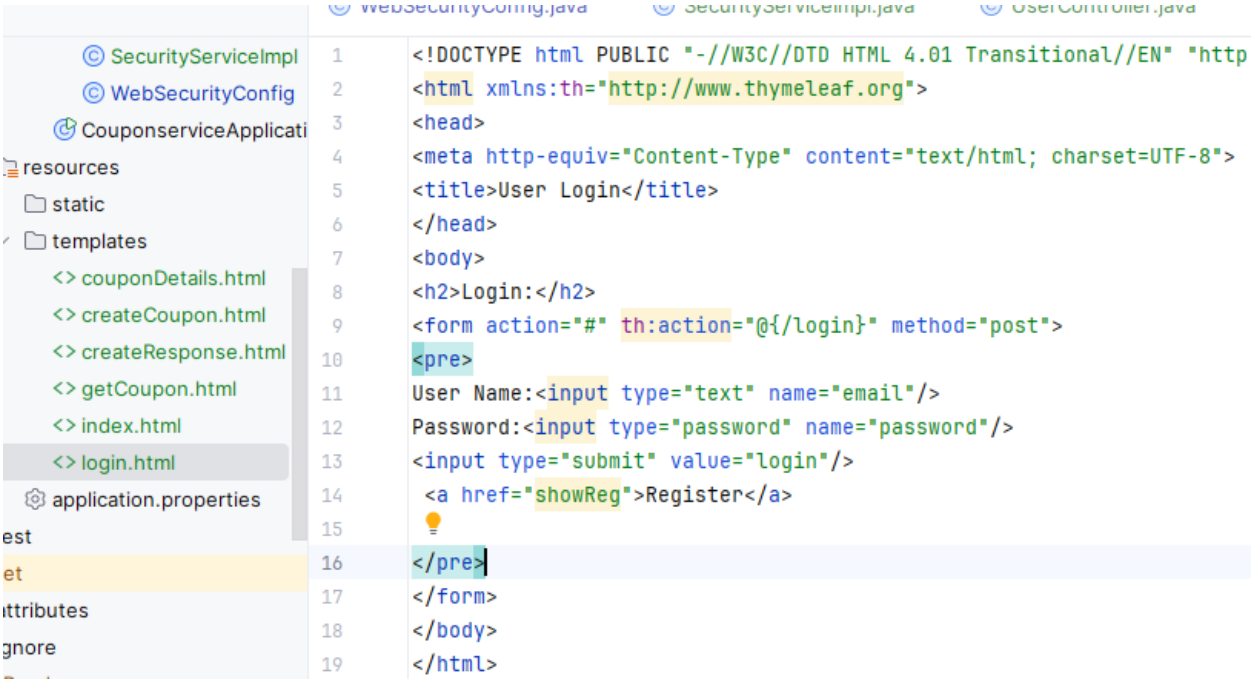
Coupon Code SuperSell1 Get Coupon



Coupon Details

Code:**SuperSell1**
Discount:**20.000**
Expiry Date:**12/12/2026**

Implement Custom Login:



UserController.java

WebSecurityConfig.java | SecurityServiceImpl.java | UserController.java | SecurityServ

```java
public class UserController {

    @Autowired  1 usage
    private SecurityService securityService;

    @GetMapping("/")  no usages  new *
    public ModelAndView showLoginpage(){
        ModelAndView mav = new ModelAndView( viewName: "login");
        return mav;
    }

    @PostMapping("/login")  no usages  new *
    public ModelAndView login(String email, String password
    , HttpServletRequest request, HttpServletResponse response){
        boolean isLogin = securityService.login(email,password,request,response);
        if(isLogin){
            ModelAndView mav = new ModelAndView( viewName: "index");
            return mav;
        }
        ModelAndView mav = new ModelAndView( viewName: "login");
        return mav;
    }

}
```

SecurityService Interface:

```java
package com.couponservice.security;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public interface SecurityService {  3 usages  1 implementation  new *

    boolean login(String username, String password, HttpServletRequest request, HttpServletResponse response);
}
```

SecurityServiceImpl.java:

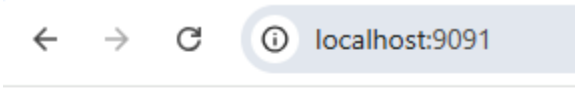
```java
@Service  no usages  new *
public class SecurityServiceImpl implements SecurityService{

    @Autowired  1 usage
    private UserDetailsService userDetailsService;

    @Autowired  1 usage
    private AuthenticationManager authenticationManager;

    @Autowired  1 usage
    private SecurityContextRepository securityContextRepository;

    @Override  1 usage  new *
    public boolean login(String username, String password
    , HttpServletRequest request, HttpServletResponse response) {

        UserDetails userDetails = userDetailsService.loadUserByUsername(username);

        UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(
                userDetails,password,userDetails.getAuthorities());
        authenticationManager.authenticate(token);
        boolean result = token.isAuthenticated();
        if(result){
            SecurityContext securityContext = SecurityContextHolder.getContext();
            securityContext.setAuthentication(token);
            securityContextRepository.saveContext(securityContext,request,response);
            }
        return result;
```


WebSecurityConfig.java:

```java
public class WebSecurityConfig  {                                                    8

    @Autowired  1 usage                                          Code changed:
    UserDetailsService userDetailsService;

    @Bean  1 usage  Rahul Bhati
    BCryptPasswordEncoder passwordEncoder(){
        return  new BCryptPasswordEncoder();
    }

    @Bean  no usages  new *
    SecurityContextRepository securityContextRepository(){
        return new DelegatingSecurityContextRepository(new RequestAttributeSecurityContextRepository(),
                new HttpSessionSecurityContextRepository());
    }

    @Bean  no usages  new *
    AuthenticationManager authenticationManager(){
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return new ProviderManager(provider);
    }

    @Bean  no usages  Rahul Bhati *
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
        //http.formLogin(Customizer.withDefaults());
        http.authorizeHttpRequests( AuthorizationManagerRequestMat... autorize->
                autorize.requestMatchers(HttpMethod.GET, ...patterns: "/couponapi/coupons/**") AuthorizedUrl
                        .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.GET, ...patterns: "/showCreateCoupon","/createCoupon","/createResponse")
                        .hasAnyRole( ...roles: "ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.POST, ...patterns: "/couponapi/coupon","/saveCoupon") AuthorizedUrl
                        .hasAnyRole( ...roles: "ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.POST, ...patterns: "/getCoupon") AuthorizedUrl
                        .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers(HttpMethod.GET, ...patterns: "/showGetCoupon","/couponDetails") AuthorizedUrl
                        .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
                        .requestMatchers( ...patterns: "/","/login","/index").permitAll())
                .logout( LogoutConfigurer<HttpSecurity> logout->logout.logoutSuccessUrl("/")) ;

        http.csrf( CsrfConfigurer<HttpSecurity> csrf->csrf.disable());
        http.securityContext( SecurityContextConfigurer<HttpSecurity> context->context.requireExplicitSave(true));
        return http.build();
    }
}
```

Custom UserDetailsServiceImpl.java:

```java
package com.couponservice.impl;

import com.couponservice.model.User;
import com.couponservice.repo.UserRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service  no usages  Rahul Bhati
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired  1 usage
    private UserRepo repo;

    @Override  1 usage  Rahul Bhati
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        User user = repo.findByEmail(username);
        if(user==null){
            throw new UsernameNotFoundException("Username not found!!!");
        }
        // Ye krne se hme spring security ko UserDetails ka object bna kr return kiya.
        return new org.springframework.security.core.userdetails.User(
                user.getEmail(),user.getPassword(),user.getRoles());
    }
}
```



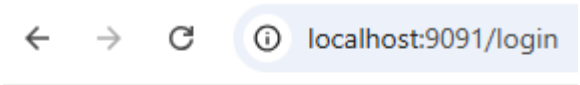
Login:

User Name:

Password:

login

Register



Create Coupon

Get Coupon Logout

Sign Out

Implement Registration:

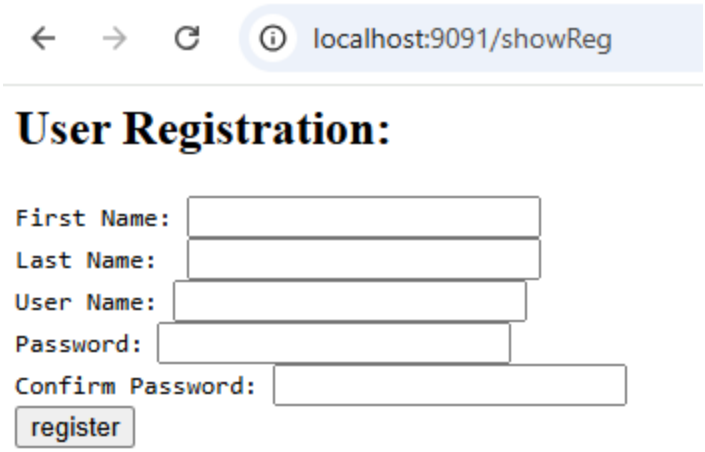
WebSecurityConfig:

WebSecurityConfig.java

```
21  public class WebSecurityConfig  {
46  @   SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
47          //http.formLogin(Customizer.withDefaults());
48          http.authorizeHttpRequests( AuthorizationManagerRequestMat... autorize->
49                  autorize.requestMatchers(HttpMethod.GET, ...patterns: "/couponapi/coupons/**") AuthorizedUrl
50                          .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
51                          .requestMatchers(HttpMethod.GET, ...patterns: "/showCreateCoupon","/createCoupon","/createResponse")
52                          .hasAnyRole( ...roles: "ADMIN") AuthorizationManagerRequestMat...
53                          .requestMatchers(HttpMethod.POST, ...patterns: "/couponapi/coupon","/saveCoupon") AuthorizedUrl
54                          .hasAnyRole( ...roles: "ADMIN") AuthorizationManagerRequestMat...
55                          .requestMatchers(HttpMethod.POST, ...patterns: "/getCoupon") AuthorizedUrl
56                          .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
57                          .requestMatchers(HttpMethod.GET, ...patterns: "/showGetCoupon","/couponDetails") AuthorizedUrl
58                          .hasAnyRole( ...roles: "USER","ADMIN") AuthorizationManagerRequestMat...
59                          .requestMatchers( ...patterns: "/","/login","/showReg","/registerUser","/index").permitAll())
60                  .logout( LogoutConfigurer<HttpSecurity> logout->logout.logoutSuccessUrl("/")) ;
61
```

UserController:

```
public class UserController {

    @GetMapping("/showReg")  no usages  Rahul Bhati
    public ModelAndView showRegistrationPage(){
        ModelAndView mav = new ModelAndView( viewName: "registerUser");
        return mav;
    }

    @PostMapping("/registerUser")  no usages  Rahul Bhati
    public ModelAndView register(User user){
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        HashSet<Role> roles=new HashSet<>();
        Role adminRole = new Role();
        adminRole.setId(1L);
        roles.add(adminRole);
        user.setRoles(roles);
        userRepo.save(user);
        ModelAndView mav = new ModelAndView( viewName: "login");
        return mav;
    }
```



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.

1. Ek Real-world Example

Maaniye aapne apne **Bank Account** ki website par login kiya hua hai.

1. Aap login hain, isliye aapke browser mein bank ki **Session Cookie** saved hai.
2. Ab aap galti se kisi "Hacker ki Website" par chale jaate hain.
3. Wo website background mein ek hidden form chala deti hai jo aapke Bank ko request bhejta hai: *"Mera saara paisa Hacker ke account mein bhej do"*.
4. Bank ko lagta hai ki ye request **aapne** hi bheji hai, kyunki browser ne request ke saath aapki purani "Session Cookie" bhi bhej di. Bank transaction complete kar deta hai.

2. Spring Security ise kaise rokti hai? (CSRF Token)

Spring Security isse bachne ke liye **CSRF Token** ka use karti hai.

- Jab bhi aap koi form load karte hain, Spring ek unique, random token generate karta hai (jaise: `asdf897asdf`).
- Jab aap form submit karte hain, to browser ko wo token wapas bhejana padta hai.
- Hacker ke paas wo token nahi hota, isliye uski request fail ho jati hai.

3. Spring Boot mein CSRF Settings

Kyunki aap Spring Boot 3 use kar rahe hain, to default taur par CSRF **enabled** hota hai.

Agar aapko disable karna ho (Sirf Testing ya Stateless API/JWT ke liye):

Java

```
http.csrf(csrf -> csrf.disable());
```

Agar aap JSP ya Thymeleaf use kar rahe hain: Spring automatically aapke forms mein ek hidden field add kar dega:

HTML

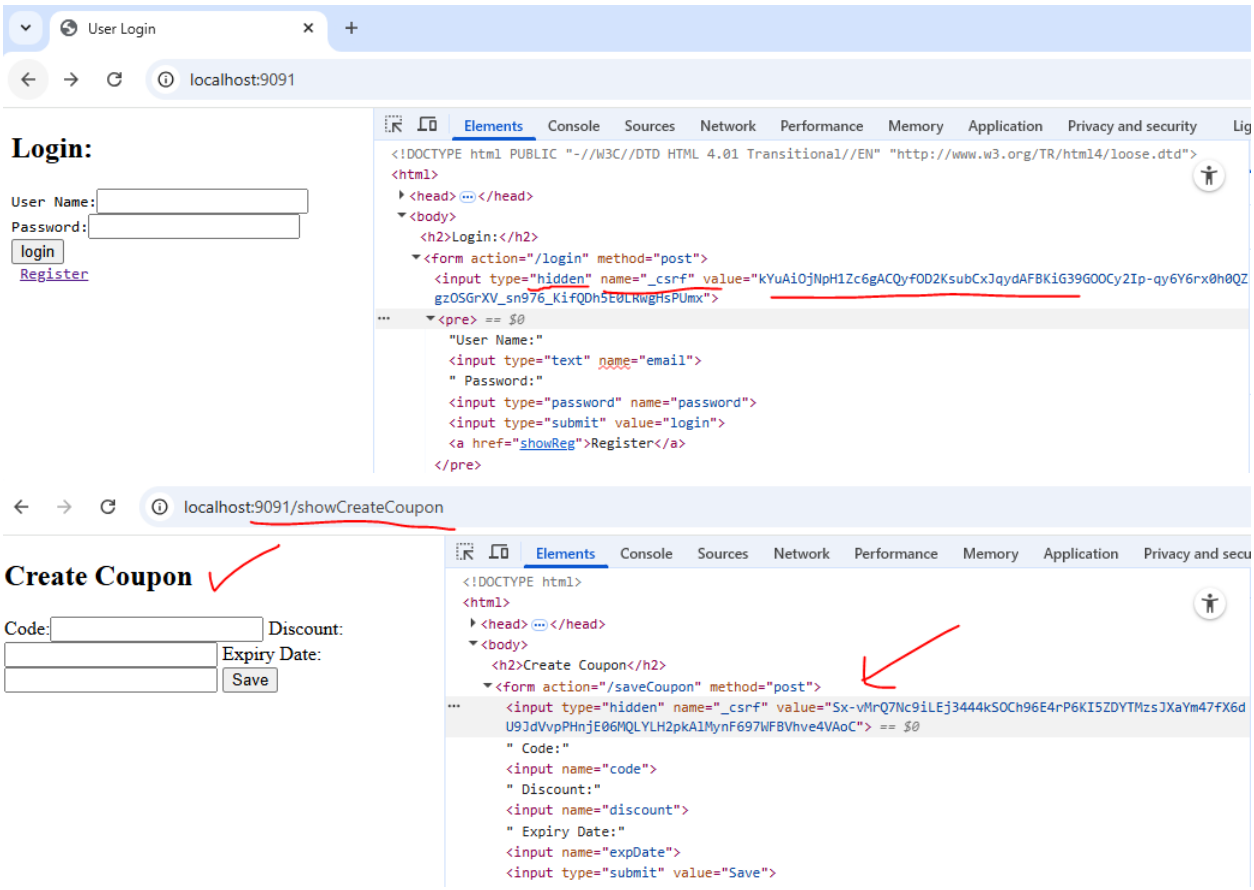
```
<input type="hidden" name="_csrf" value="asdf897asdf..." />
```

CSRF Kab Disable Karna Chahiye?

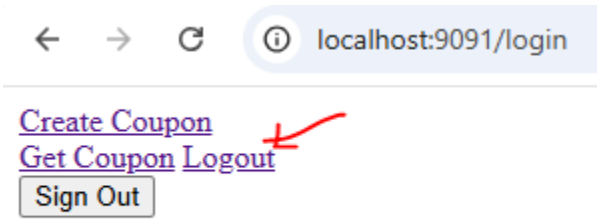
Aapko CSRF tabhi disable karna chahiye jab:

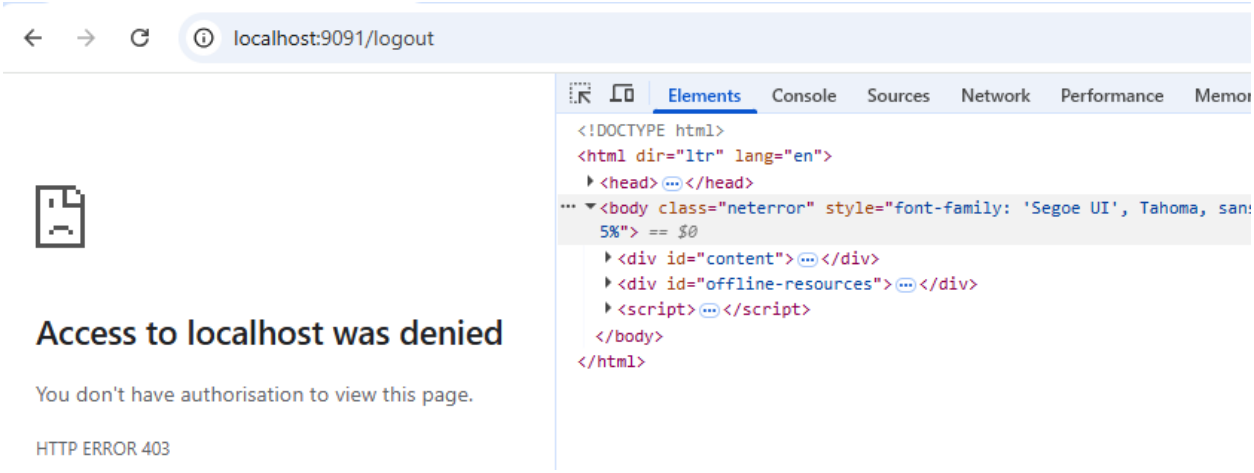
1. Aap ek **REST API** bana rahe hain jo **Stateless** hai (yaani aap Cookies use nahi kar rahe, balki **JWT/Token** use kar rahe hain).
2. Aapki API browser ke bahar se (jaise Mobile App) access hone wali hai.

Every Incomming request has this hidden csrf value token.

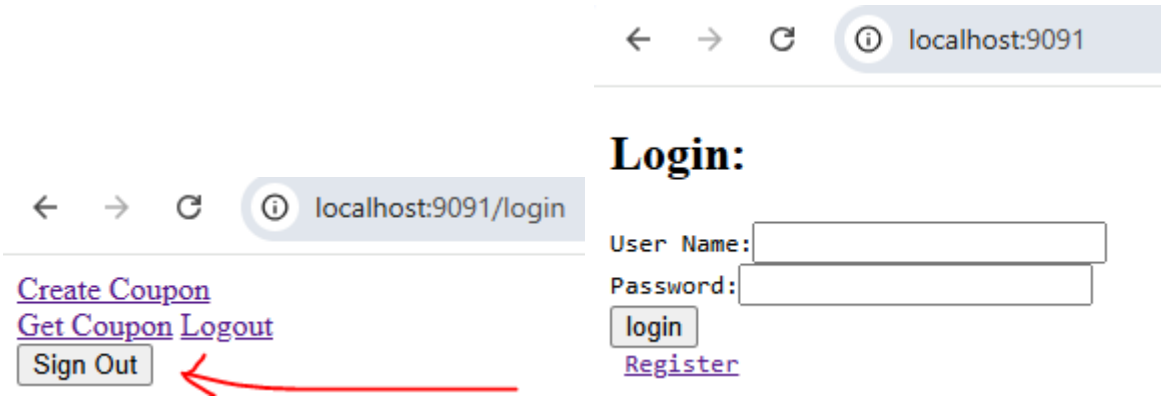


Jab hm logout kisi href link pr click krte he to will get error





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.

```
//http.csrf(csrf->csrf.disable());
http.csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf
        .ignoringRequestMatchers( ...patterns: "/couponapi/coupons/**", "/getCoupon")
);
```

Latest code me kuchh is tarah ka changes he:

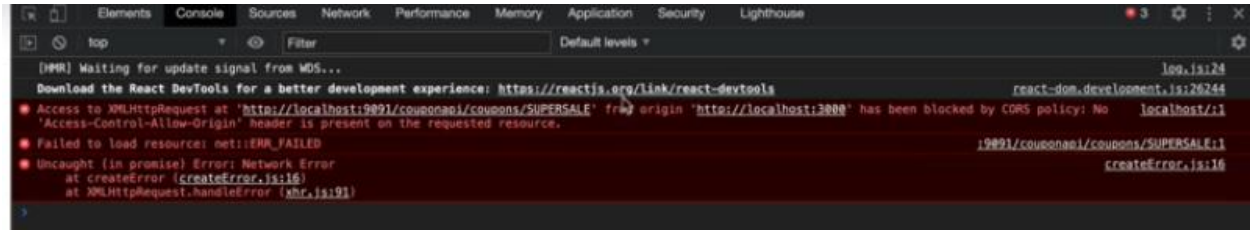
```
http.csrf(csrfCustomizer -> {
    csrfCustomizer.ignoringAntMatchers("/couponapi/coupons/**");
    RequestMatcher requestMatchers = new RegexRequestMatcher("/couponapi/coupons/{code:^[A-Z]*$}", "POST");
    requestMatchers = new MvcRequestMatcher(new HandlerMappingIntrospector(), "/getCoupon");
    csrfCustomizer.ignoringRequestMatchers(requestMatchers);
});
}
```

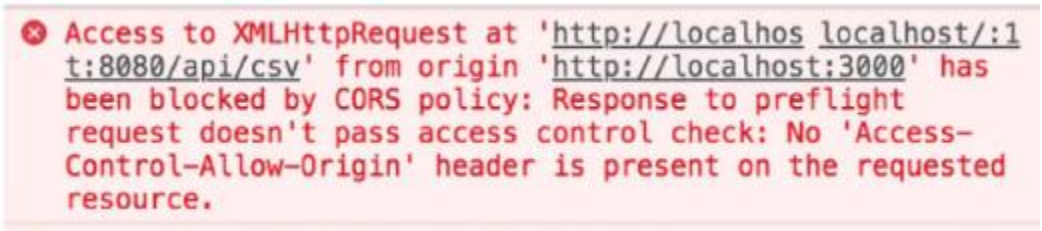
Spring Security CORS: (Cross-Origin Resource Sharing):

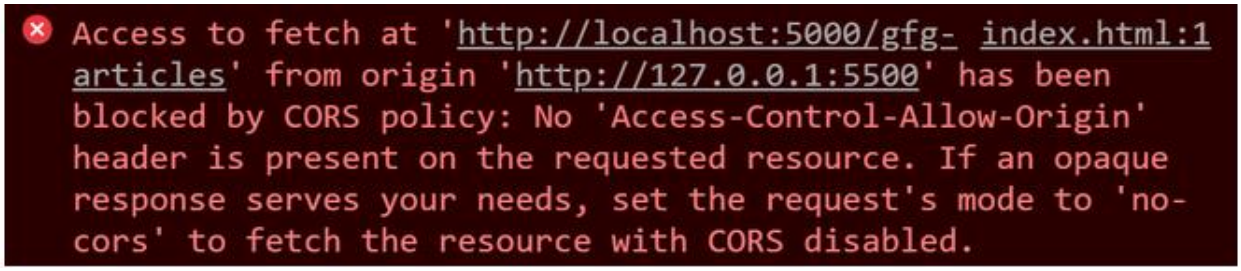
CORS ka matlab hai **Cross-Origin Resource Sharing**. Yeh ek security feature hai jo browsers mein hota hai.

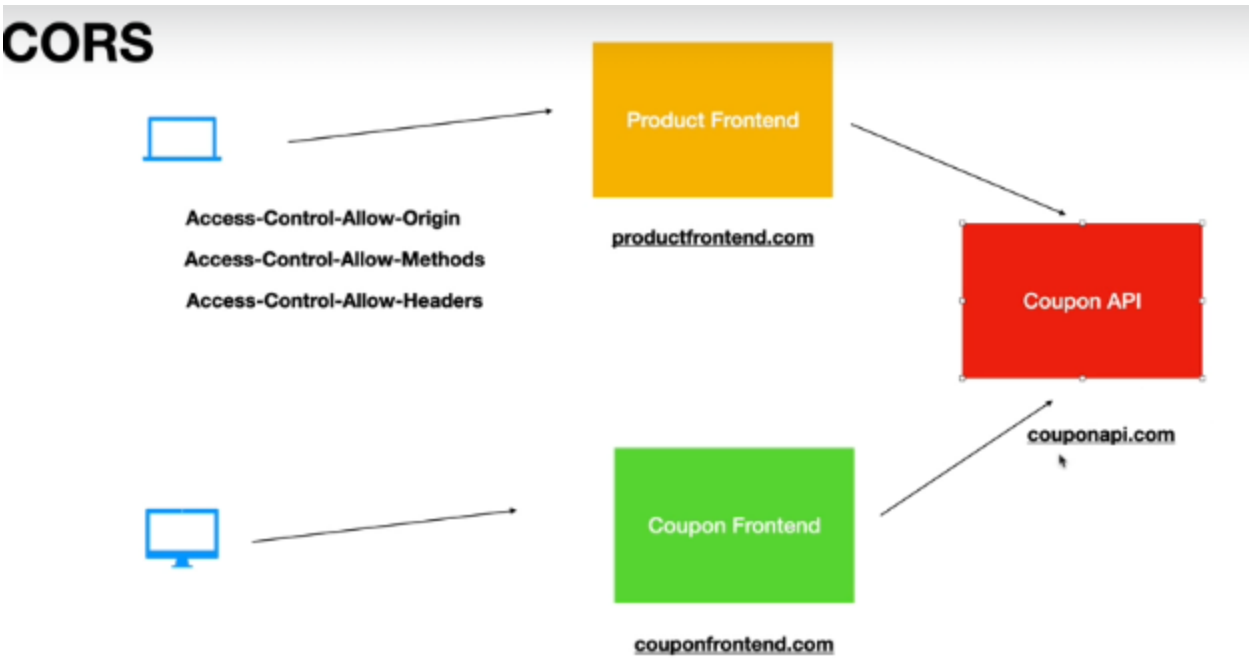
Asaan bhasha mein: Agar aapka Frontend (jaise React/Angular) `localhost:3000` par chal raha hai aur aapka Backend (Spring Boot) `localhost:8080` par, to browser backend ko request block kar dega jab tak backend permission na de. Is permission ko hi CORS configuration kehte hain.

Issue are below:









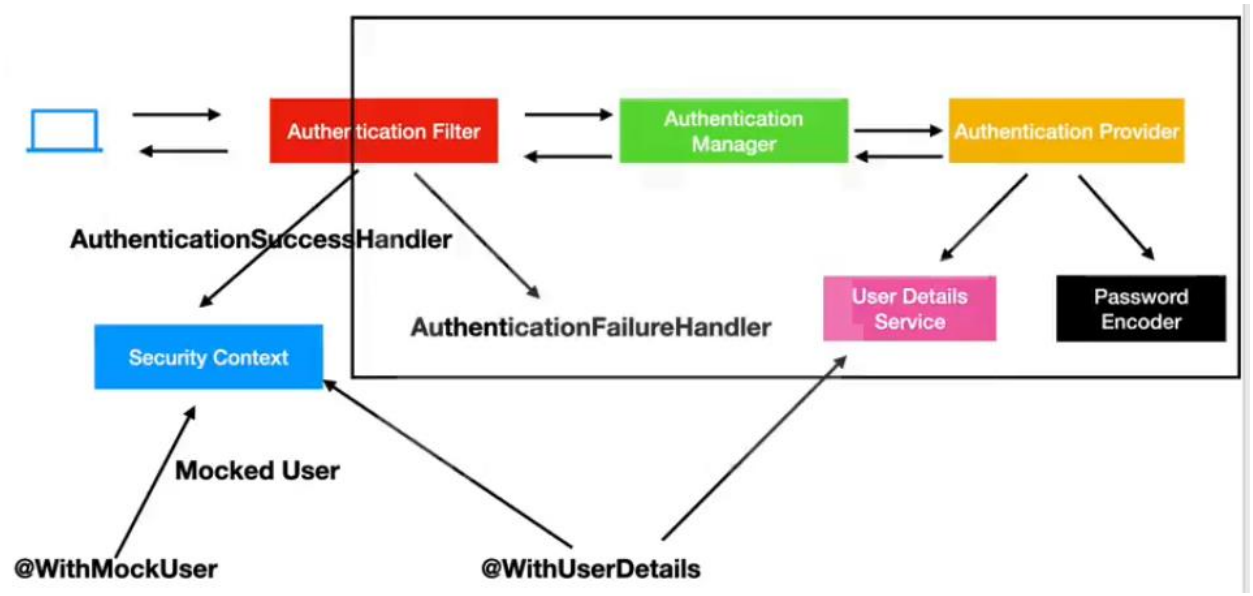
Configuration in your controller only need to add annotation - **@CrossOrigin**

```
@RestController
@RequestMapping("/couponapi")
@CrossOrigin
public class CouponRestController {

    @Autowired
    CouponRepo repo;

    @PostMapping("/coupons")
    public Coupon create(@RequestBody Coupon coupon) {

        return repo.save(coupon);

    }

    @GetMapping("/coupons/{code}")
    public Coupon getCoupon(@PathVariable("code") String code) {
        return repo.findByCode(code);

    }
}
```

You can also customize CORS like below:

```
        http.cors(corsCustomizer -> {
            CorsConfigurationSource configurationSource = request -> {
                CorsConfiguration corsConfiguration = new CorsConfiguration();
                corsConfiguration.setAllowedOrigins(List.of("localhost:3000"));
                corsConfiguration.setAllowedMethods(List.of("GET"));
                return corsConfiguration;
            };
            corsCustomizer.configurationSource(configurationSource);
        });

    }
```

Spring Security Testing:



Need to add dependency:

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

```
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.web.servlet.MockMvc;

@SpringBootTest
@AutoConfigureMockMvc
public class CouponserviceApplicationTests {

    @Autowired
    MockMvc mvc;

    @Test
    public void testGetCouponWithoutAuth_Forbidden() throws Exception {
        mvc.perform(get("/couponapi/coupons/SUPERSALE")).andExpect(status().isForbidden());
    }

    @Test
    @WithMockUser(roles = { "USER" })
    public void testGetCouponWithAuth_Success() throws Exception {
        mvc.perform(get("/couponapi/coupons/SUPERSALE")).andExpect(status().isOk()).andExpect(
                content().string("{\"id\":1,\"code\":\"SUPERSALE\",\"discount\":80.000,\"expDate\":\"12/12/2020\"}"));
    }

}
```

Test CSRF Roles:

```
@SpringBootTest
@AutoConfigureMockMvc
public class CouponserviceApplicationTests {

    @Autowired
    MockMvc mvc;

    @Test
    public void testGetCouponWithoutAuth_Forbidden() throws Exception {
        mvc.perform(get("/couponapi/coupons/SUPERSALE")).andExpect(status().isForbidden());
    }

    @Test
    @WithMockUser(roles = { "USER" })
    public void testGetCouponWithAuth_Success() throws Exception {
        mvc.perform(get("/couponapi/coupons/SUPERSALE")).andExpect(status().isOk()).andExpect(
                content().string("{\"id\":1,\"code\":\"SUPERSALE\",\"discount\":80.000,\"expDate\":\"12/12/2020\"}"));
    }

    @Test
    @WithMockUser(roles = { "ADMIN" })
    public void testCreateCoupon_WithoutCSRF_Forbidden() throws Exception {
        mvc.perform(post("/couponapi/coupons")
                .content("{\"code\":\"SUPERSALECSRF\",\"discount\":80.000,\"expDate\":\"12/12/2020\"}")
                .contentType(MediaType.APPLICATION_JSON)).andExpect(status().isForbidden());
    }

    @Test
    @WithMockUser(roles = { "ADMIN" })
    public void testCreateCoupon_WithCSRF_Forbidden() throws Exception {
        mvc.perform(post("/couponapi/coupons")
                .content("{\"code\":\"SUPERSALECSRF\",\"discount\":80.000,\"expDate\":\"12/12/2020\"}")
                .contentType(MediaType.APPLICATION_JSON).with(csrf().asHeader())).andExpect(status().isOk());
    }

}
```

```
    @Test
    @WithMockUser(roles = { "USER" })
    public void testCreateCoupon_NonAdminUser_Forbidden() throws Exception {
        mvc.perform(post("/couponapi/coupons")
                .content("{\"code\":\"SUPERSALECSRF\",\"discount\":80.000,\"expDate\":\"12/12/2020\"}")
                .contentType(MediaType.APPLICATION_JSON).with(csrf().asHeader())).andExpect(status().isForbidden());
    }
```

Test for CORS support:

```
    @Test
    @WithMockUser(roles = { "USER" })
    public void testCORS() throws Exception {
        mvc.perform(options("/couponapi/coupons").header("Access-Control-Request-Method", "POST").header("Origin",
                "http://www.bharath.com")).andExpect(status().isOk())
                .andExpect(header().exists("Access-Control-Allow-Origin"))
                .andExpect(header().string("Access-Control-Allow-Origin","*"))
                .andExpect(header().exists("Access-Control-Allow-Methods"))
                .andExpect(header().string("Access-Control-Allow-Methods","POST"));

    }
```

@MockUserDetails:

```
@SpringBootTest
@AutoConfigureMockMvc
public class CouponserviceApplicationTests {

    @Autowired
    MockMvc mvc;

    @Test
    public void testGetCouponWithoutAuth_Forbidden() throws Exception {
        mvc.perform(get("/couponapi/coupons/SUPERSALE")).andExpect(status().isForbidden());
    }

    @Test
    //@WithMockUser(roles = { "USER" })
    @WithUserDetails("doug@bailey.com")
    public void testGetCouponWithAuth_Success() throws Exception {
        mvc.perform(get("/couponapi/coupons/SUPERSALE")).andExpect(status().isOk()).andExpect(
                content().string("{\"id\":1,\"code\":\"SUPERSALE\",\"discount\":80.000,\"expDate\":\"12/12/2020\"}"));
    }

    @Test
    @WithMockUser(roles = { "ADMIN" })
    public void testCreateCoupon_WithoutCSRF_Forbidden() throws Exception {
        mvc.perform(post("/couponapi/coupons")
                .content("{\"code\":\"SUPERSALECSRF\",\"discount\":80.000,\"expDate\":\"12/12/2020\"}")
                .contentType(MediaType.APPLICATION_JSON)).andExpect(status().isForbidden());
    }
```

Spring Method Level Security:

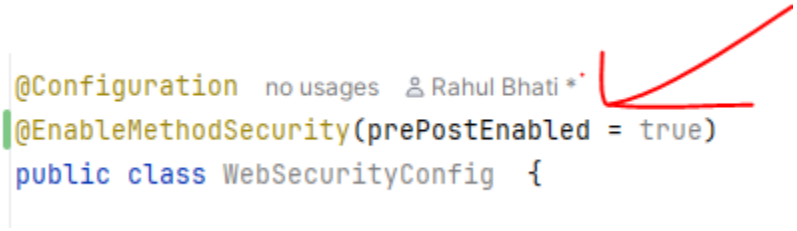
Global Method Security

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.

```
public class WebSecurityConfig  {

    @Bean  no usages  Rahul Bhati *
    SecurityFilterChain filterChain(HttpSecurity http,HandlerMappingIntrospector introspector) throws Exception{
        //http.formLogin(Customizer.withDefaults());
    /*   http.authorizeHttpRequests(autorize->
                autorize.requestMatchers(HttpMethod.GET,"/couponapi/coupons/**")
                        .hasAnyRole("USER","ADMIN")
                        .requestMatchers(HttpMethod.GET,"/showCreateCoupon","/createCoupon","/createResponse")
                        .hasAnyRole("ADMIN")
                        .requestMatchers(HttpMethod.POST,"/couponapi/coupon","/saveCoupon")
                        .hasAnyRole("ADMIN")
                        .requestMatchers(HttpMethod.POST,"/getCoupon")
                        .hasAnyRole("USER","ADMIN")
                        .requestMatchers(HttpMethod.GET,"/showGetCoupon","/couponDetails")
                        .hasAnyRole("USER","ADMIN")
                        .requestMatchers("/","/login","/showReg","/registerUser","/index").permitAll())
                .logout(logout->logout.logoutSuccessUrl("/")) ;
    */
```

Enable Method Security Annotation:



1. Enable Kaise Karein?

Apni configuration class (jahan `SecurityFilterChain` hai) ke upar ye annotation lagayein:

Java

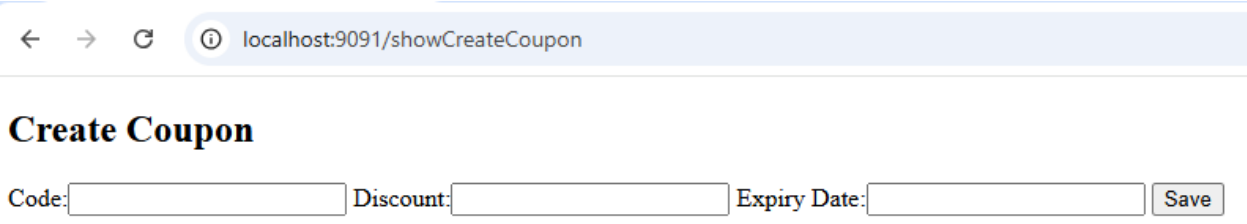
```
@Configuration
@EnableMethodSecurity // Default mein prePostEnabled = true hota hai
public class MethodSecurityConfig {
    // ...
}
```

@PreAuthorize:

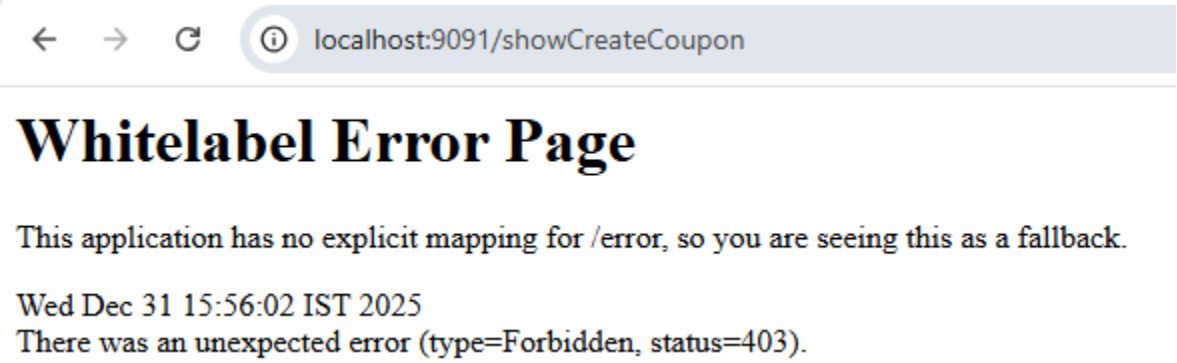
@PreAuthorize Spring Security ka sabse powerful annotation hai. Iska kaam method ke execute hone se **theek pehle** access check karna hota hai. Agar condition true hai, tabhi method chalega, warna 403 Forbidden error aa jayega.

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.

1. Simple Example (Owner Check)

Maaniye aap ek coupon system bana rahe hain. User koi bhi ID daal kar coupon dekhne ki koshish kar sakta hai, lekin aap chahte hain ki user sirf **apna hi coupon** dekh sake.

Java

```java
@PostAuthorize("returnObject.owner == authentication.name")
public Coupon getCouponById(Long id) {
    // Ye method pehle database se data layega
    return couponRepository.findById(id);
    // Agar returnObject (Coupon) ka owner login user nahi hai,
    // to user ko AccessDeniedException mil jayegi.
}
```

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

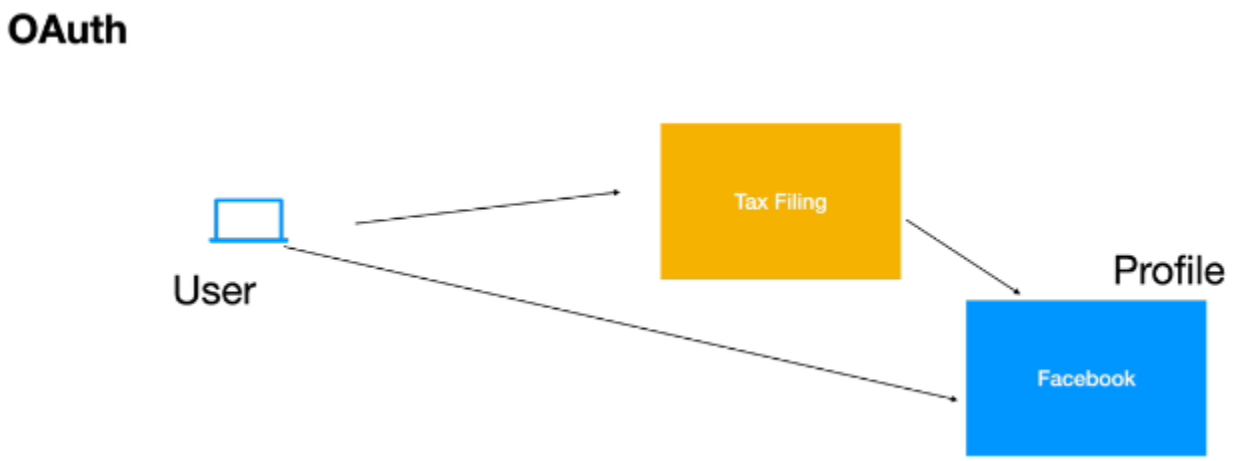
```java
    @PostMapping("/getCoupon")  no usages  Rahul Bhati
    @PostAuthorize("returnObject.discount<60")
    public ModelAndView getCoupon(String code) {
        ModelAndView mav = new ModelAndView( viewName: "couponDetails");
        System.out.println(code);
        mav.addObject(repo.findByCode(code));
        return mav;
    }
}
```

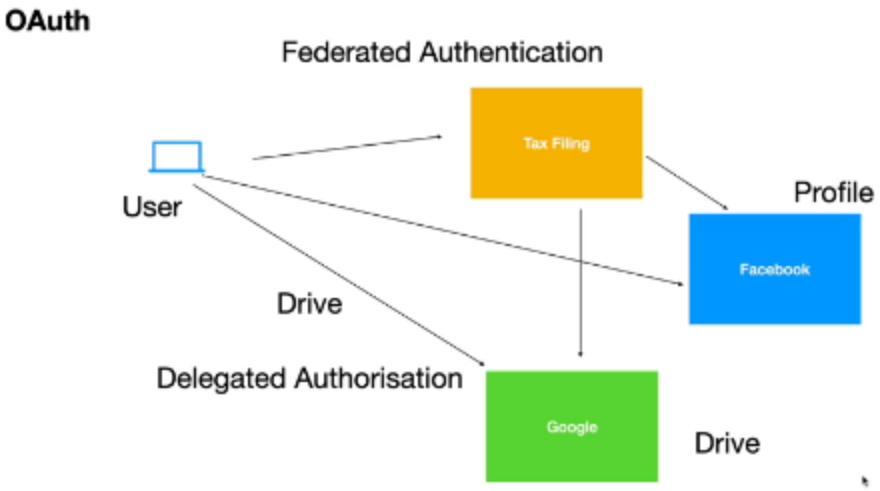
OAuth and JWT Concepts:

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.

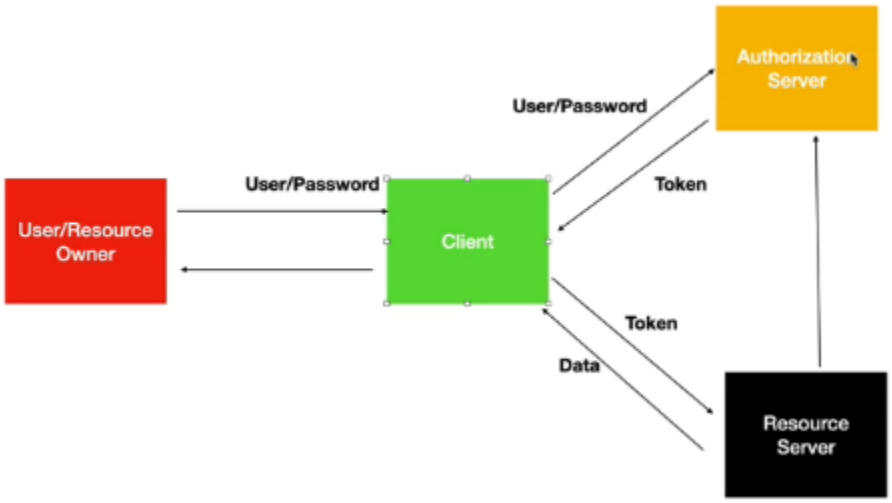




OAuth ke 4 Main Characters (Actors):

OAuth ko samajhne ke liye in 4 roles ko samajhna zaroori hai:

1. **Resource Owner:** Ye **Aap** hain. Aapka data (profile, photos) aapka resource hai.
2. **Client:** Wo **App ya Website** jo aapka data access karna chahti hai (e.g., Zomato, Canva).
3. **Authorization Server:** Wo server jo permission handle karta hai (**Google/Facebook** ka login page).
4. **Resource Server:** Jahan aapka asal data rakha hai (Google Drive, Gmail API, etc.).



OAuth Kaam Kaise Karta Hai? (The Flow)

Ise ek simple story se samajhte hain:

1. **Request:** Aap Zomato par "Login with Google" par click karte hain.
2. **Redirect:** Zomato aapko Google ke login page par bhej deta hai.
3. **Consent:** Google aapse puchta hai— *"Kya aap Zomato ko apni email aur name dekhne ki permission dete hain?"*
4. **Authorization Code:** Agar aap "Yes" kehte hain, to Google Zomato ko ek temporary **"Authorization Code"** bhejta hai.
5. **Token Exchange:** Zomato wo code lekar Google ke server ke paas jata hai aur kehta hai, *"Ye raha code, ab mujhe 'Access Token' do."*
6. **Access Token:** Google Zomato ko ek **Access Token** (ek lambi secret key) de deta hai.
7. **Data Access:** Ab Zomato us token ko dikha kar Google Resource Server se aapka naam aur email le leta hai.

Key Terms Jo Aapko Pata Honi Chahiyen:

- **Scope:** Ye permissions ki limit hoti hai (e.g., `read-only`, `profile`, `contacts`).
- **Access Token:** Wo "Chabi" (Key) jo limited time ke liye valid hoti hai.
- **Refresh Token:** Jab Access Token expire ho jaye, to bina login kiye naya token lene ke liye iska use hota hai.

Grant Type:

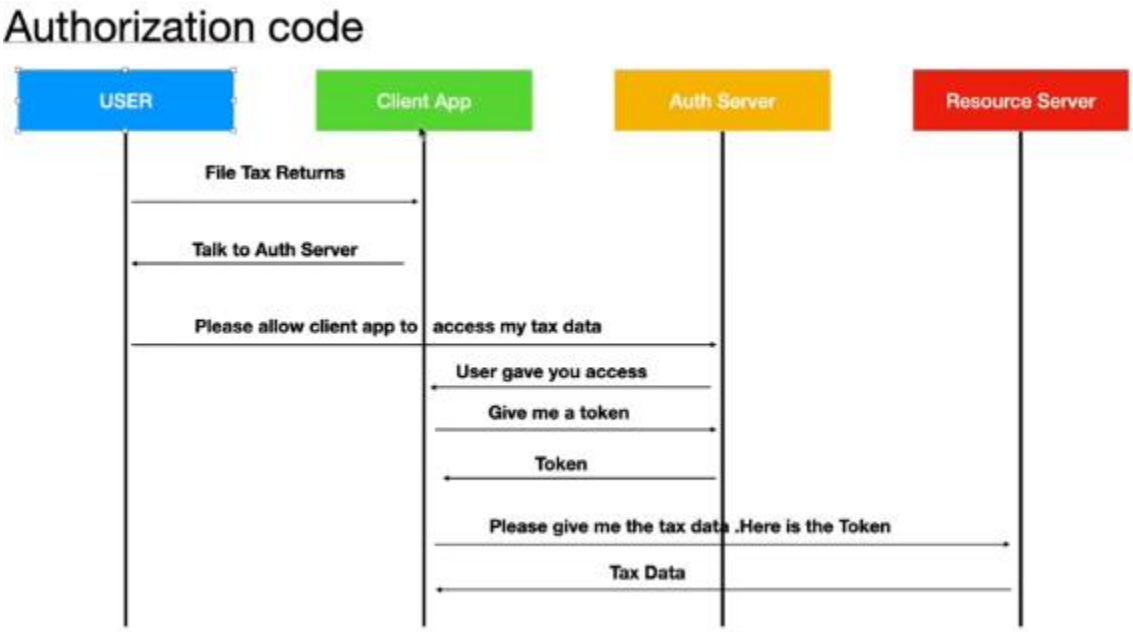
OAuth 2.0 mein **Grant Type** ka matlab hai wo "Rasta" ya "Tarika" (Method) jiske zariye ek Client App ek **Access Token** hasil karti hai.

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):

Ye sabse zyada use hone wala flow hai (e.g., Social Logins). Isme Client App ko kabhi user ka password nahi dikhta.

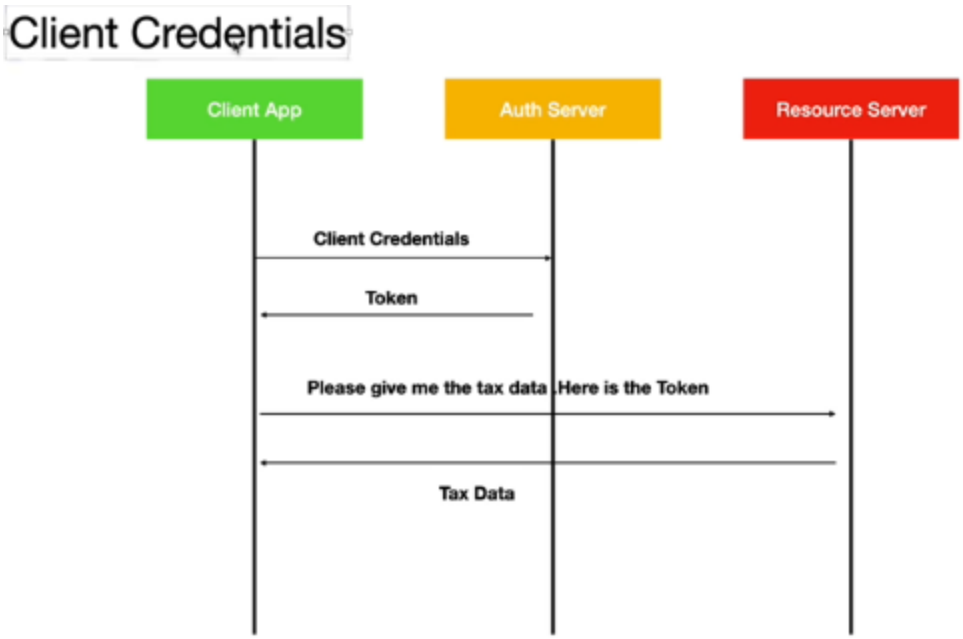
- **Kahan use hota hai:** Web Apps (Server-side) jahan secret key safe rakhi ja sake.
- **Kaise kaam karta hai:** Pehle ek temporary "Code" milta hai, fir us code ko "Access Token" se exchange kiya jata hai.



2. Client Credentials Grant:

Isme koi "User" involve nahi hota. Ye **Machine-to-Machine (M2M)** communication ke liye hai.

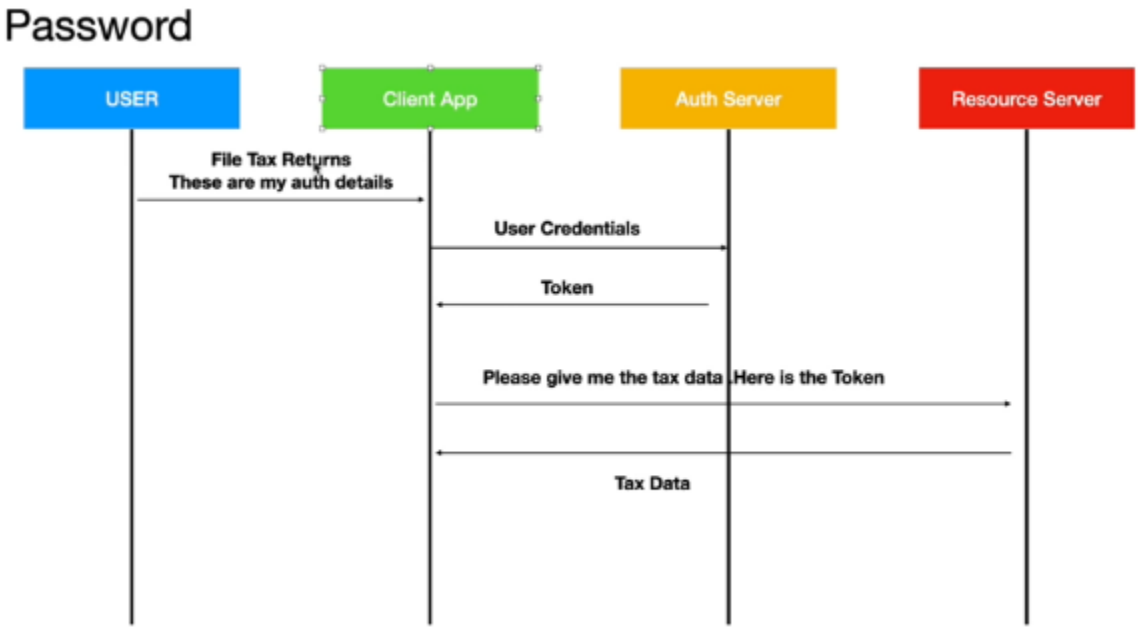
- **Kahan use hota hai:** Jab ek Microservice dusri Microservice se baat karti hai.
- **Kaise kaam karta hai:** App apni `client_id` aur `client_secret` bhejti hai aur token mil jata hai.



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

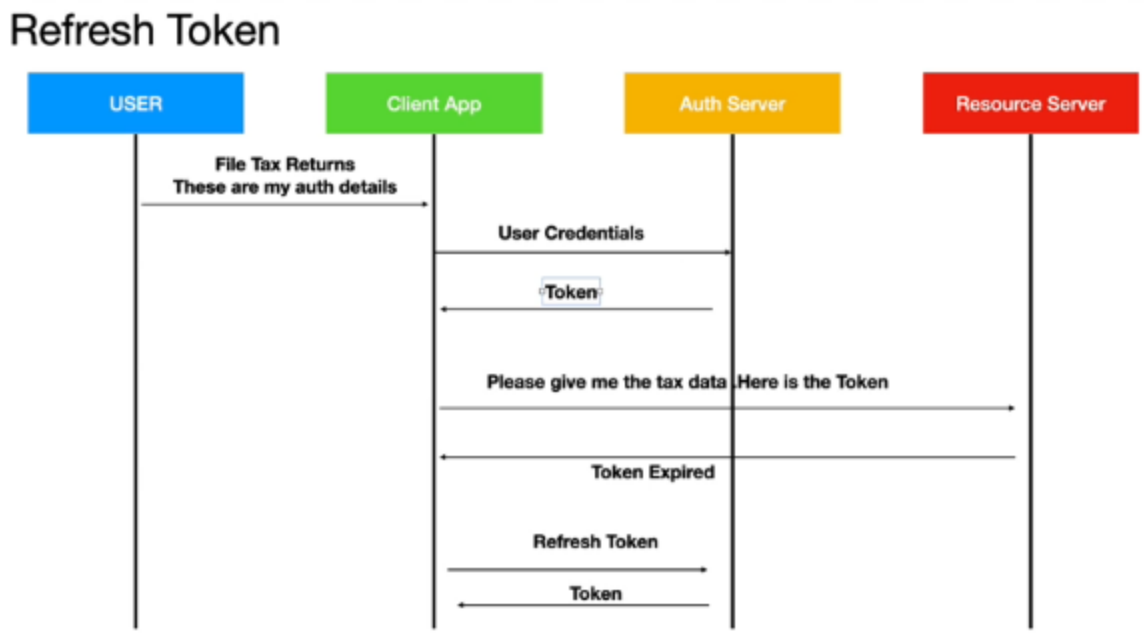
Isme user apna username aur password direct Third-party app ko de deta hai.

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



5. 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.



Grant Types Comparison Table

Grant Type	Kiske Liye Hai?	User Involvement?	Security Level
Authorization Code	Web Apps / Mobile Apps	Haan	High
Client Credentials	Backend Services (M2M)	Nahi	Medium
Refresh Token	Token renew karne ke liye	Nahi	High
Implicit	Single Page Apps (Old)	Haan	**Low (Deprecated)**

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.

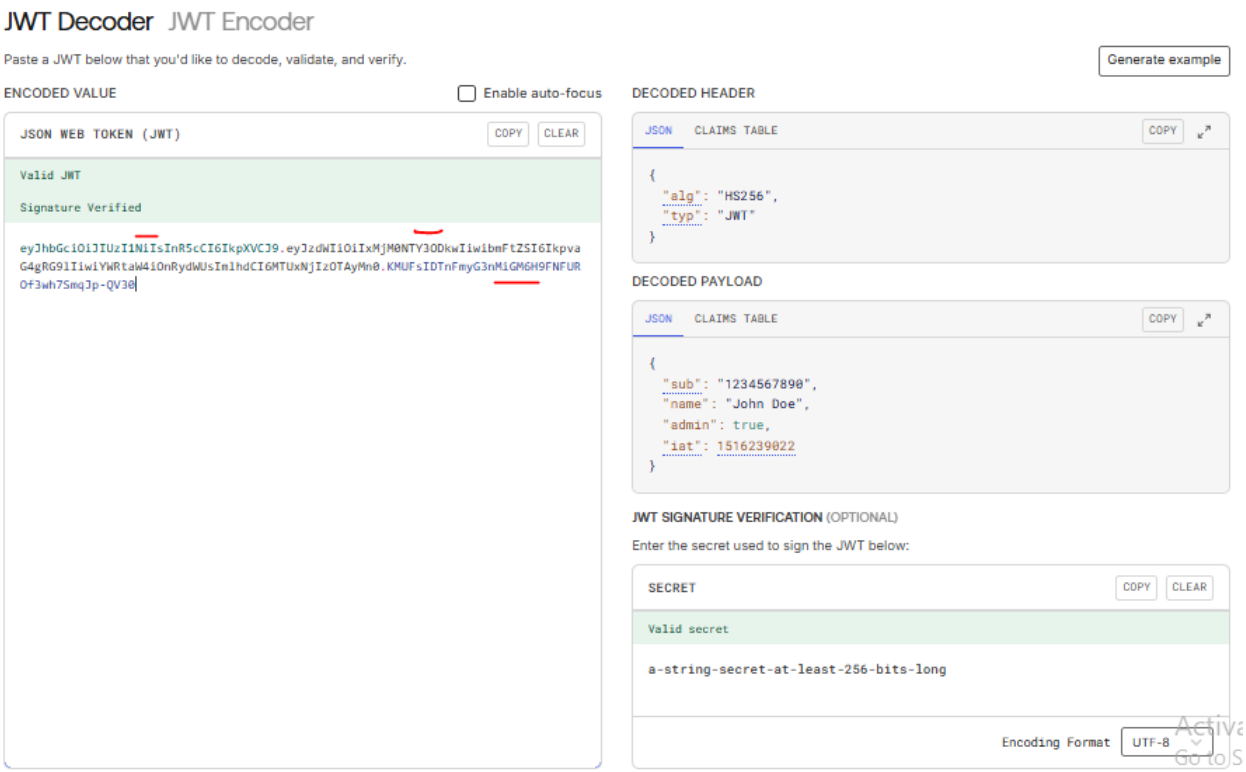
Spring Boot aur Modern Web Apps mein JWT ka sabse bada use **Stateless Authentication** ke liye hota hai.

JWT Kaise Dikhta Hai?

Ek JWT teen hisson (parts) mein divide hota hai jo dots (`.`) se separate hote hain:
`header.payload.signature`

1. **Header:** Isme token ka type (JWT) aur hashing algorithm (jaise HS256 ya RS256) ki jaankari hoti hai.
2. **Payload (Claims):** Isme actual data hota hai, jaise User ID, Name, aur Roles. (Dhyan rahe: Isme password jaisi sensitive info mat rakhein kyunki ise koi bhi decode kar sakta hai).
3. **Signature:** Yeh sabse zaroori hissa hai. Yeh Header + Payload + Ek **Secret Key** ko milakar banta hai. Isse ye verify hota hai ki token ke saath raste mein koi chhed-chhad (tampering) nahi hui hai.

https://www.jwt.io/



JWT Kaam Kaise Karta Hai? (The Workflow)

1. **Login:** User apna username/password bhejta hai.
2. **Verification:** Server credentials check karta hai aur ek **JWT** generate karke wapas bhej deta hai.
3. **Storage:** Browser is token ko `LocalStorage` ya `Cookie` mein save kar leta hai.
4. **Next Requests:** Agli har request ke saath browser ye token **Authorization Header** mein bhejta hai: `Authorization: Bearer <token>`
5. **Validation:** Server sirf signature verify karta hai. Use database baar-baar check karne ki zaroorat nahi padti (isliye ise **Stateless** kehte hain).

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.

KeyStore vs TrustStore

- **KeyStore:** Isme aapki apni **Private Key** aur certificate hota hai. (Jaise aapke ghar ki asli chabi).
- **TrustStore:** Isme dusre servers ke **Public Certificates** hote hain jinpar aap trust karte hain. (Jaise kisi guest ka ID card).

Spring Boot mein iska Istemal (JWT Asymmetric Signing)

Jab aap JWT use karte hain, to do options hote hain:

1. **Symmetric (HS256):** Ek hi secret key se sign aur verify hota hai.
2. **Asymmetric (RS256):** Private key se sign hota hai aur Public key se verify hota hai (Zyada secure).

Generate Asymmetric Key:

```
keytool -genkeypair -alias jwtiscool -keyalg RSA -keypass jwtiscool -keystore jwtiscool.jks
-storepass jwtiscool

keytool -list -rfc --keystore jwtiscool.jks | openssl x509 -inform pem -pubkey
```

It will generate jks key file in the folder :

```
bharaththippireddy@bharaths-MacBook-Pro ~ % pwd
/Users/bharaththippireddy
bharaththippireddy@bharaths-MacBook-Pro ~ % keytool -genkeypair -alias jwtiscool -keyalg
RSA -keypass jwtiscool -keystore jwtiscool.jks -storepass jwtiscool
What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a vali
dity of 90 days
        for: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
bharaththippireddy@bharaths-MacBook-Pro ~ %
```

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

Authorization Server ke Main Components

Ek Authorization Server banane ke liye aapko ye beans configure karni padti hain:

1. **Protocol Endpoints:** Jo `/oauth2/authorize`, `/oauth2/token`, aur `/.well-known/openid-configuration` jaise URLs handle karte hain.
2. **Client Repository:** Jahan aapki "Client Apps" (jaise React, Android app) ki details save hoti hain (`clientId`, `clientSecret`, `scopes`).
3. **User Details Service:** Jahan aapke asli users (username/password) save hote hain.
4. **JWK Source:** JSON Web Keys jo JWT tokens ko sign karne ke liye use hoti hain (Isi ke liye humne `keytool` ka discuss kiya tha).