
spring
boot

INDEX

Spring Boot Security ----------

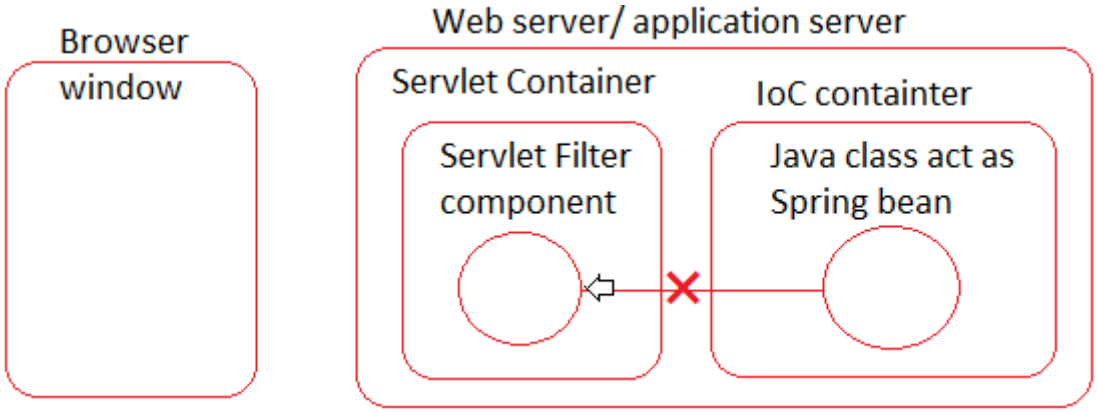
1. Introduction 04
2. Different way of implementing the Security 07
3. Spring Boot Security using InMemory DB 08
    a. AntMatchers in Spring Boot Security 17
4. Spring Boot Security using DB s/w 20
5. Spring Boot Security with Spring Boot Data JPA 24
6. CSRF problem & Solution 43
7. Spring Boot Security using LDAP Server 46
8. Spring Boot Security with OAuth2.x 59
    a. OAuth 2.x Implementation 62
    b. Facebook Developer account creation Process 64
    c. Developing single Sign in application using Facebook 68
    d. Developing single Sign in application using Google 81

Spring Boot Security

Introduction

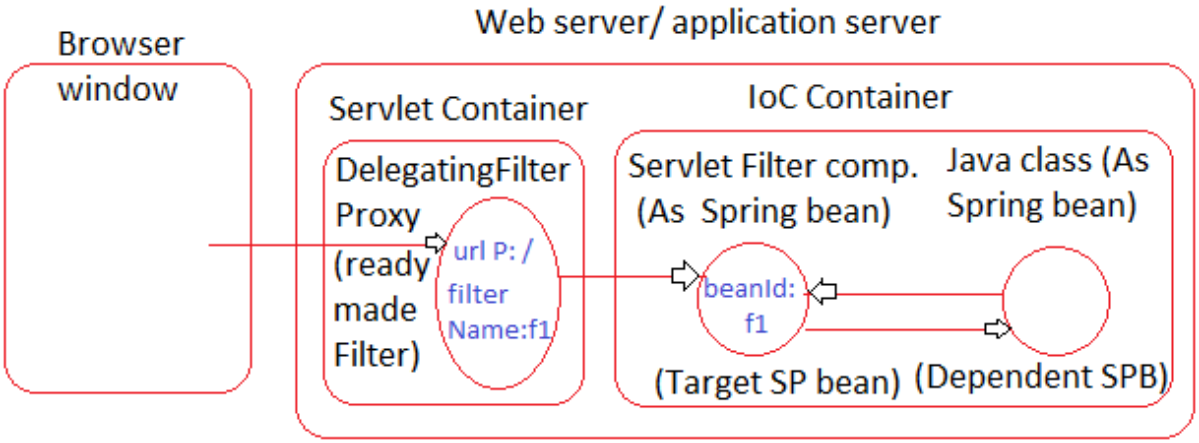
- It is Spring extension module that provides multiple readymade filters to enable security on Spring Boot MVC and Spring Boot Rest applications.

Problem:



- Here Spring bean injection to Servlet Filter component is not possible. Because Servlet Filter is not taken as Spring bean managed by IoC container.
- In order to inject dependent Spring bean to Servlet Filter component we need to take Servlet Filter component also as Spring bean in IoC container.

Solution:



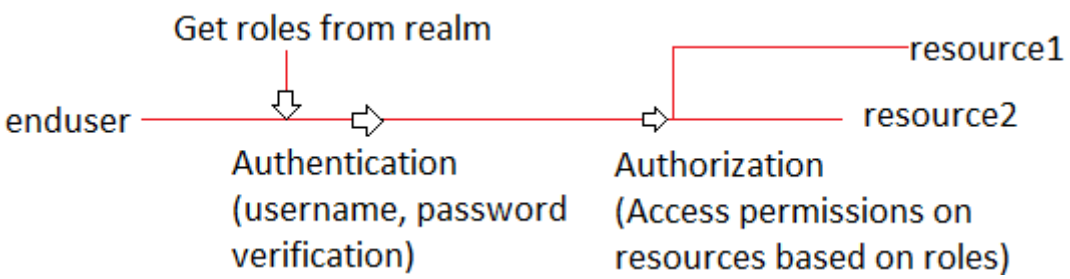
- This time dependency injection is possible.
- If we take Servlet Filter component as Spring bean in IoC container then any another dependent Spring bean object can be injected to that Servlet Filter Spring bean, but that Servlet Filter cannot trap and take the requests given browser window.

- For this we need to configure special readymade Servlet Filter component given by Spring Security module that is "DelegatingFilterProxy" having URL pattern "/" and logical name matching with Servlet Filter component's bean id.

With respect to Solution diagram:

1. Browser gives the request to web application.
2. DelegatingFilterProxy traps and takes the request.
3. DelegatingFilterProxy delegates/ passes the request to Servlet Filter component that is acting as Spring bean in IoC container.
4. The logics in Servlet Filter component executes.
5. Servlet Filter component uses the injected dependent Spring bean services through method calls.

- Security in web application is nothing but enabling Authentication + Authorization on the web application.
- Authentication: Checking the identity of user using usernames, passwords, thumb impressions, iris, digital signatures and etc.
- Authorization: Checking the access permissions of the authenticated users on different resources of the Project.
  (Here the roles of the user will be verified before allowing the user to access different resources of the Project)
- Roles are nothing but designations given to users based on roles of the users the access permissions on different resources will be decided.
- E.g.,
  - All customers, employees of the bank must be authenticated to use bank application.
  - The users having customer role will get less access permission on the resources where as the users having employee role will get more access permission on the resources.



Note: realm means small DB s/w or repository where the usernames, passwords and roles are managed.

- It is always recommended to enable authorization of accessing the resources not based on the username. It is recommended to perform on the roles of the users.
- During the authentication process gets the roles of the user and use those roles for authorization.

The components of Security implementation

a. Authentication provider/ Authentication Info provider
b. Authentication and Authorization manager

Authentication provider/ Authentication Info provider:

- It is the small realm where usernames, passwords, roles are managed and will be used during both authentication and authorization.
    a. Properties file
    b. XML file
    c. JSON file
    d. DB s/w
    e. LDAP server (best)
    f. lnMemory DB
    and etc.

Note:

- ✓ LDAP means Lightweight Directory Access Protocol.
- ✓ In all other Authentication providers, we can get the password of the users if they forgot the passwords. Whereas in LDAP server no provision to get back the password, only resetting of password is possible.

Authentication and Authorization manager:

- It is the component that verifies the given username and password to perform authentication and gives 401 error if authentication fails.
- The same component collects the roles of authenticated user and performs Authorization activities while accessing different resources and gives 403 error if authorization fails.
- The Authentication and Authorization manager can be arranged in two ways,
    a. Using Programmatic approach (bad)
        - Here we need to develop the logics of Authentication and Authorization manager explicitly by spending huge amounts of time and brain manually (Not recommended).

b. Using Declarative approach (good)
   - Max web server/ application server provide built-in security support by providing built-in Authentication and Authorization manager by adding entries web.xml file we can activate that Authentication and Authorization manager (refer security in web application in Servlet, JSP environment).

Limitations of Declaration approach for securing web applications:

working with Servlet container supplied Authentication and Authorization manager.

a. Only selected servers support this feature sometimes we need to purchase License of costly web server or application server to use this feature.
b. As of now this facility possible by adding additional entries in web.xml file, i.e., not suitable in 100% driven configuration, Spring Boot apps.
c. When move from one server to another server these configurations in web.xml file may change (web.xml entries related to security are not portable across the multiple servers).
d. No support for LDAP server as authentication info provider.
   and etc.

Note: To overcome these problems, we use Spring Security or Spring Boot Security.

Advantages of Spring Security/ Spring Boot Security:

a. Can be used in Spring MVC/ Spring Rest/ Spring Microservices apps and also in non-Spring based web applications like Servlet, JSP web applications, JSF web applications and etc.
b. The security configurations code is portable across the multiple servers.
c. We can this security irrespective of whether the underlying server supports the Servlet Container level Declarative security service or not (Spring security is no way related to Servlet container's security).
d. Supports different Authentication info providers including LDAP.
e. We need not to arrange costly servers only for security.

Different way of implementing the Security

- In Spring environment or Spring Boot environment we can apply security on MVC apps or Spring Rest apps or Microservices apps in 3 approaches,
  a. Using Spring Security/ Spring Boot Security

- Basic Authentication (browser generates dialog box asking username, password)
- Form based Authentication (readymade or user-defined form page will be there asking username, password)
    i. Using lnMemory DB as authentication info provider (RAM Level DB)
    ii. Using Properties file as authentication info provider
    iii. Using DB s/w as authentication info provider with the support of Spring JDBC/ Spring ORM/ Spring Data JPA/ User Details service
    iv. Using LDAP server as authentication info provider.

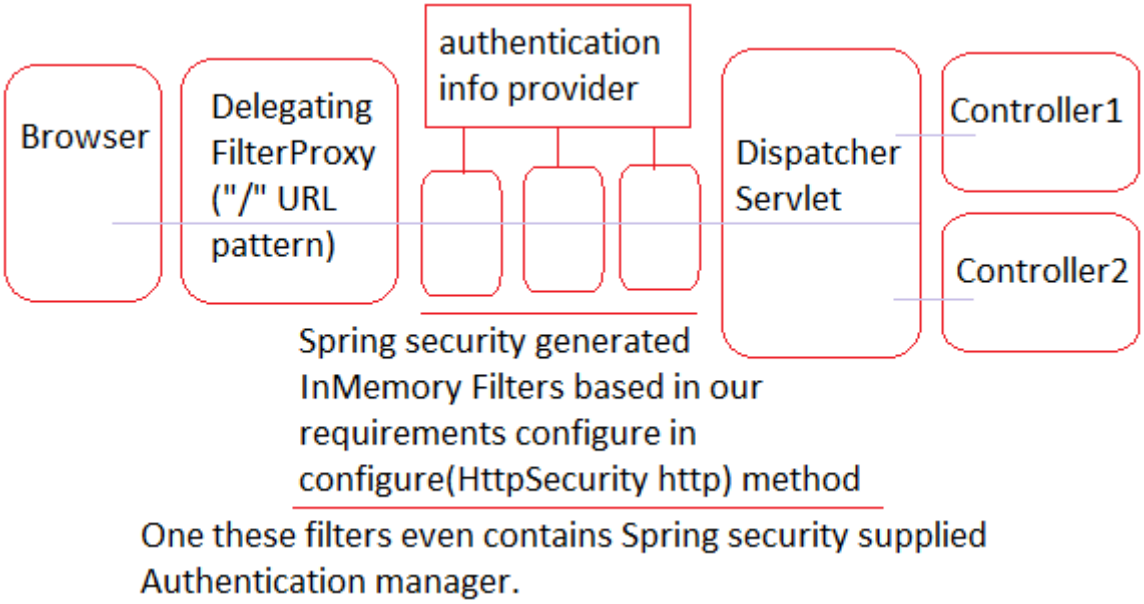
b. Using JWT (JSON web tokens)
c. Using OAuth 2.x (Open Authorization)

Spring Boot Security using InMemory DB

- Once we add spring-boot-stater-security to Spring MVC/ Spring Rest/ Microservices project one readymade filter called "DelegatingFilterProxy" will be registered with "/" URL pattern having logical name "springSecurityFilterChain".
- For this One class will be generated extending from AbstractSecurityWebApplicationlnitializer (AC) (Internal abstract class).
- We need to develop ConfigurationAdapter class as @Configuration class extending from WebSecurityConfigurerAdapter and overriding two configure (-) methods having authentication info provider, authentication, authorization details.
- Every detail we add in authentication and authorization one separate InMemory Filters will be generated as Spring bean and they will be linked to DelegatingFilterProxy filter i.e., DelegatingFilterProxy traps the request and links to these request with dynamically generate filters then passes to controller classes through DispatcherServlet.

```
@Configuration
@EnableWebSecurity
public class SpringSecurityConfig extends
WebSecurityConfigurerAdapter {

        @Override
        public void configure(AuthenticationManagerBuilder auth) throws
                                                        Exception {
```

```
        //Provide logic for configuration Authentication info
provider like InMemoryDB, DB s/w etc.
}

@Override
public void configure(HttpSecurity http) throws Exception {
        //Provide logic for   Authentication and authorization and
etc.
}

}
```



One these filters even contains Spring security supplied Authentication manager.

Authorization levels in Spring Security

a. permitAll()
   - No authentication + No authorization (no role checking).
   - E.g., home page, about us page, contact us page, terms and conditions page
b. authenticate()
   - Only Authentication on the given request URL resource(controller) and no authorization (no role checking).
   - E.g., main menu page, inbox page, send/ composite mail page
c. hasARole()
   - Authentication + authorization (role checking will be there).
   - E.g., checking balance page, transfer money, withdraw/ deposited money page, changing password deleting mails and etc.
d. hasAnyRole()

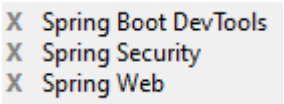
- Authentication + authorization (any one role should be there for user in the list of given roles)
- checking balance page, deposit money page, etc.

Controller classes request paths:

- /home permitAll
- /contactUs permitAll
- /aboutUs permitAll
- /inbox authenticated
- /checkbalance hasAnyRole “USER”, “MANAGER”
- /transferMoney hasARole “USER”
- /depositeMoney hasAnyRole “USER”, “MANAGER”, “VISITOR”

Procedure to develop Spring Boot Security app that is lnMemory DB (RAM Level) as the authentication provider

Step 1: Create Spring stater project adding the following starters



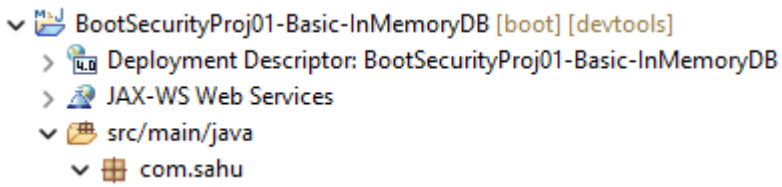
Step 2: Develop the regular controller class having different handler methods with different request paths.

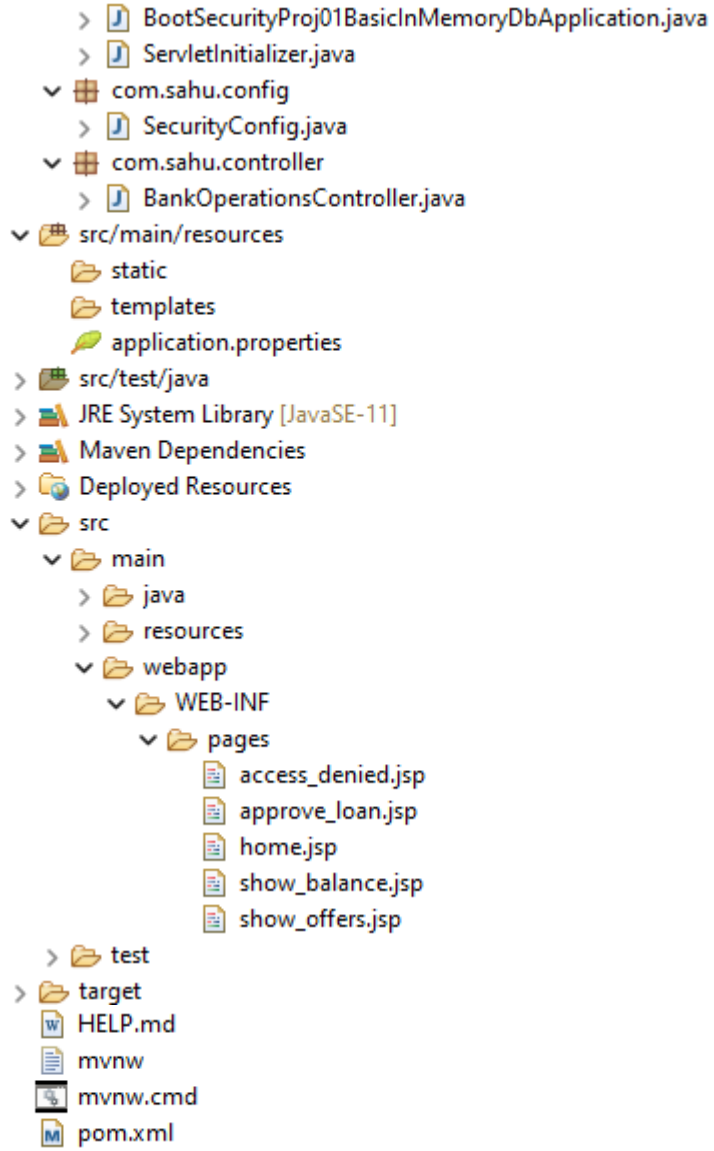
Step 3: Decide authentication and authorization level for different request URLs.

- / -- permitAll
- /offers – authenticated
- /balance – authenticated + authorization hasAnyRole("CUSTOMER","MANAGER")
- /loanApprove -- authenticated + authorization hasRole("MANAGER")

Step 4: Develop SecurityConfig class extending WebSecurityConfigurerAdapter and having annotations @Configuration + @EnableWebSecurity and also overriding two configure (-) methods.

Directory Structure of BootSecurityProj01-Basic-InMemoryDB:





- Develop the above directory structure using Spring Starter Project option and create the package, classes, folders and JSP files also.
- Use the following starters during project creation.

  X Spring Boot DevTools
  X Spring Security
  X Spring Web

- Then place the following code with in their respective files.

application.properties

```
#View Resolver
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
```

Note: Spring Boot DevTools is used for take the changes automatically.

BankOperationsController.java

```
package com.sahu.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class BankOperationsController {

	@GetMapping("/")
	public String showHome() {
		return "home";
	}

	@GetMapping("/offers")
	public String showOffers() {
		return "show_offers";
	}

	@GetMapping("/balance")
	public String checkBalance() {
		return "show_balance";
	}

	@GetMapping("/loanApprove")
	public String approveLoan() {
		return "approve_loan";
	}

	@GetMapping("/denied")
	public String accessDenied() {
		return "access_denied";
	}

}
```

SecurityConfig.java

```
package com.sahu.config;

import org.springframework.context.annotation.Configuration;
```

```
import
org.springframework.security.config.annotation.authentication.builders.Aut
henticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableW
ebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSec
urityConfigurerAdapter;

@Configuration
@EnableWebSecurity //Makes the normal @Configuration class to Spring
Security configuration class
public class SecurityConfig extends WebSecurityConfigurerAdapter {

		@Override
		public void configure(AuthenticationManagerBuilder auth)
throws Exception {
			//Provide logic for configuration Authentication info
provider like InMemoryDB, DB s/w etc.

	auth.inMemoryAuthentication().withUser("raja").password("{noop}ra
ni").authorities("CUSTOMER");

	auth.inMemoryAuthentication().withUser("ramesh").password("{noo
p}ramesh123").authorities("MANAGER");
		}

		@Override
		public void configure(HttpSecurity http) throws Exception {
			//Provide logic for   Authentication and authorization and
etc.
			http.authorizeRequests().antMatchers("/").permitAll()
//No authentication and no authorization
				.antMatchers("/offers").authenticated()
//Only authentication

	.antMatchers("/balance").hasAnyAuthority("CUSTOMER",
```

```
"MANAGER") //authentication + authorization for "CUSTOMER",
"MANAGER" role users

        .antMatchers("/loanApprove").hasAnyAuthority("MANAGER")
//authentication + authorization for "MANAGER" role users
                                .anyRequest().authenticated() //Remaining
all requests URL must be authenticated
                                .and().httpBasic() //Specify authentication
mode

        .and().exceptionHandling().accessDeniedPage("/denied");
//Exception/ error handling
                }

}
```

home.jsp

```
<%@ page isELIgnored="false" %>

<h1 style="color: red; text-align: center;">Welcome to XYZ Bank -- Home
page</h1>

<div style="text-align: center;">
        <a href="balance">Check Balance</a>   
        <a href="offers">Show Offers</a>  
        <a href=loanApprove>Approve Loan</a>
</div>
```

show_balance.jsp

```
<%@page import="java.util.Random"%>
<%@ page isELIgnored="false" %>

<h1 style="color: blue; text-align: center;">Show Balance page</h1>
<div style="text-align: center;">
        <b>Balance : <%=new Random().nextInt(10000000)%></b>
</div>

<div style="text-align: center;">
        <a href="./">Home</a>  
```

```
        <a href="offers">Show Offers</a>  
        <a href="loanApprove">Approve Loan</a>
</div>
```

show_offers.jsp

```
<%@ page isELIgnored="false" %>

<h1 style="color: blue; text-align: center;">Show offers page</h1>
<div style="text-align: center;">
        Home Loan ROI: 7% <br>
        Four wheeler Loan ROI: 8% <br>
        Personal Loan ROI: 12%
</div>

<div style="text-align: center;">
        <a href="./">Home</a>  
        <a href="balance">Check Balance</a>   
        <a href="loanApprove">Approve Loan</a>
</div>
```

approve_loan.jsp

```
<%@page import="java.util.Random"%>
<%@ page isELIgnored="false" %>

<h1 style="color: blue; text-align: center;">Loan Approval page</h1>
<div style="text-align: center;">
        <b>Your approved for Loan amount : <%=new
Random().nextInt(10000000)%></b>
</div>

<div style="text-align: center;">
        <a href="./">Home</a>  
        <a href="balance">Check Balance</a>   
        <a href="offers">Show Offers</a>
</div>
```

access_denied.jsp

```
<%@ page isELIgnored="false" %>
```

```
<h1 style="color: red; text-align: center;">Authorization Failed</h1>

<br>
<div style="text-align: center;">
        <a href="./">home</a>
</div>
```

Limitations of BASIC mode Authentication:

a. The dialog box asking username, password is browser specific dialog and it cannot be customized.
b. Does not allow to add the following features
    - Logout
    - Remember Me
    - Session Max Active Count Limit
      and etc.

- To overcome the above problems, take the support of form login
    - .and().httpBasic() gives BASIC mode of authentication
    - .and().formLogin() gives FORM mode of mode of authentication.
    - .and().rememberMe() adds another Filter supporting remember me authentication. Internally uses persistent cookies to remember given username, password having 48-hour expiry time. In this 48 hour, after successfully sing in, if you close the browser by taking the URL from browser address bar, then we can use the same URL to get back to the page without any sign in activity.
    - .and().sessionManagement().maximumSessions(2).maxSessionsPreventsLogin(true) adds another Filter to controller max sessions for each user to operate the application.
    - .and().logout() add another Filter providing sign out activity having the URL or request path "/logout" by default this can be with additional code.
      .and().logout().logoutRequestMatcher(new AntPathRequestMatcher(" /signout"))
      Code in UI page
      Logout

Internal of Session Management

- If Login is successful, it creates new Session:
  HttpSession ses=req.getSession(); (or)
  HttpSession ses=req.getSession(true);

- To access the exist Session:
  HttpSession ses=req.getSession(false);

- To stop/ invalidate the Session:
  ses.invalidate();

- To specify max inactive interval period for a Session
  ses.setMaxInactiveInterval(20); //default is 30 secs

AntMatchers in Spring Boot Security

- Spring Boot security app every URL (nothing but request path of handler method) must be configured with security using permitAll() (no authentication and no authorization), authenticated() (only authentication), hasRole() & hasAnyRole() (Authentication + Authorization), for this we need to use AntMatchers concept in SecurityConfig class.

Case 1: AntMatcher for Multilevel path

```
@Controller
@RequestMapping("/customer")
public class CustomerController {

        @GetMapping("/register")
        public String registerCustomer(){
                ......
        }

        @GetMapping("/delete")
        public String deleteCustomer(){
                ......
        }

        @GetMapping("/update")
        public String updateCustomer(){
                ......
        }

}
```

- To match with multilevel path, we can give </path>** like /customer** in AntMatcher.

- .antMatchers("/customer**).hasRole("MANAGER"); - Only "MANAGER" role authenticated users can access web pages whose URLs starts with /customer and contains multi path.

Note: Multiple level path is like /customer/register, /customer/delete, /customer/update, /customer/register/abc, /customer/update/type

Case 2: AntMatcher for single level path

```
@Controller
public class CustomerController {

        @PostMapping("/registerCustomer")
        public String registerCustomer() {
                ............
        }

        @PostMapping("/registerProduct")
        public String registerProduct() {
                ..............
        }

        @PostMapping("/registerFaculty")
        public String registerFaculty() {
                ................
        }

}
```

- We can give AntMatcher using </path>* pattern.
- .antMatchers("/register*").hasAnyRole("MANAGER", "CUSTOMER")
- Matches with /registerCustomer, /registerProduct, /registerFaculty URLs.

Case 3: Multiple URLs can be given in single AntMatcher expression

Version 1:

- .antMatchers("/save").hasRole("MANAGER")
- .antMatchers("/update").hasRole("MANAGER")
- .antMatchers("/delete").hasRole("MANAGER")

Version 2: (Improved code of Version 1)

- .antherMatchers("/save", "/update", "/delete").hasRole("MANAGER")

Case 4: Left over request URLs can be identified and mapped using, .anyRequest() expression.

- Let assume we are having multiple request URLs/ paths as shown below "/save", "/update", "/delete", "/report", "/upload", "/download", "/paging", "/info", "/aboutUs"
- .antMatchers("/save", "/update").hasRole("CUSTOMER") .antMatchers("/report","/upload","/download").hasRole("MANAGER") .anyRequest().authenticated(); //represents the left over URLs like "/paging", "/aboutUs", "/info".

- {noop}<password> indicates password is not encoded indirectly it says "NoopEncoder" is to encode the password.
- Initially Spring security used allows not encoded passwords later it stopped allowing them. So, to pass non-encoded passwords we need use {noop}.
- We can use different Encoders like "BCryptEndcoder", "Base64Encoder", "SHA512Encoder", "MD5Encoder" and etc. to encode the passwords.
- If you do not to use {noop} expression based "NoopPasswordEncoder" then we need to pass encoded passwords as shown below.

Step 1: Take separate class to get Encoded passwords.

PasswordEncoder.java

```
package com.sahu.encrypt;

import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public class PasswordEncoder {

	public static void main(String[] args) {
		BCryptPasswordEncoder encoder = new
BCryptPasswordEncoder();
		String password1 = encoder.encode("rani");
		String password2 = encoder.encode("ramesh123");
		System.out.println(password1);
		System.out.println(password2);
	}

}
```

Step 2: Run the above class as normal java class and get the Encoded passwords and use them in 1st configure (-) method of SecurityConfig class.

SecurityConfig.java

```
@Override
                public void configure(AuthenticationManagerBuilder auth)
throws Exception {

                        auth.inMemoryAuthentication().passwordEncoder(new
BCryptPasswordEncoder()).withUser("raja").password("$2a$10$gocQj0TlqP
Mp6kqPzNIkguTh5zJg9HY9NA2ZihCKJ0H6GZW.MmWFa").roles("CUSTOMER
");
                        auth.inMemoryAuthentication().passwordEncoder(new
BCryptPasswordEncoder()).withUser("ramesh").password("$2a$10$R.tCZrQ
Szee50STxfR2b.OYirfDxhwDgh7jEd7uNgLzfINkcGkir2").roles("MANAGER");
                }
```

Note: always recommended to work with encoded password to manage the passwords with strong encryption.

Q. Do we need to encode the passwords manually as shown above in the real projects?

Ans.

Definitely not, as part of user registration logic we include our choice encoder to get encoded password for the given password they will be saved DB table.

Spring Boot Security using DB s/w

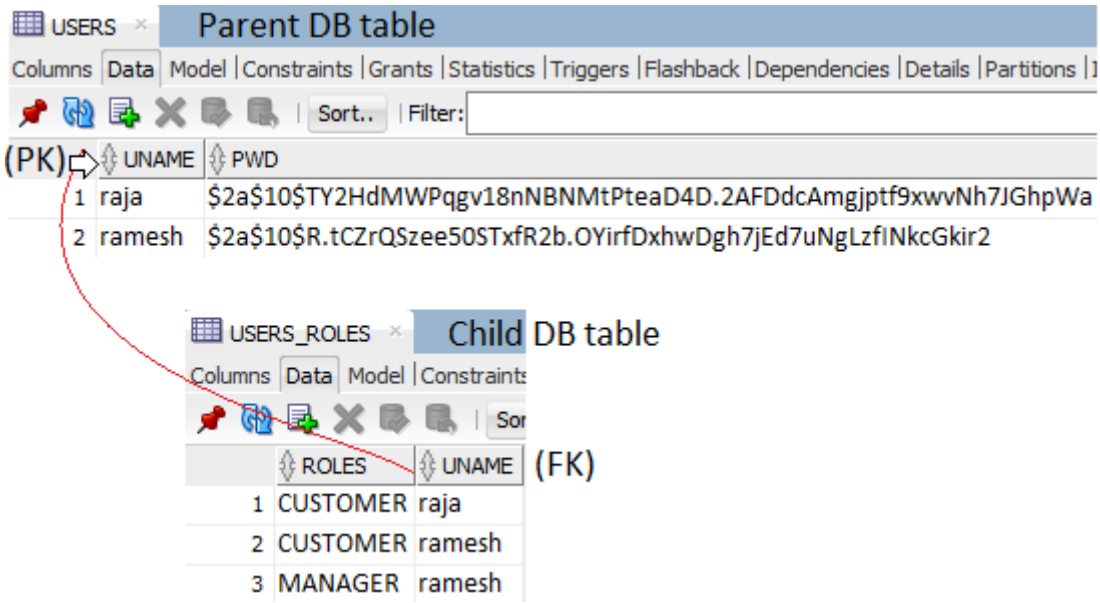
- Working with JDBC authentication that uses Spring JDBC based DB s/w as Authentication Info Provider.

Step 1: Makes that following DB tables are available in any DB s/w like oracle

```
Parent table
--------------------------------
 CREATE TABLE "SYSTEM"."USERS"
 (        "UNAME" VARCHAR2(20 BYTE) NOT NULL ENABLE,
          "PWD" VARCHAR2(100 BYTE),
          "STATUS" NUMBER (1,0),
           CONSTRAINT "USERS_PK" PRIMARY KEY ("UNAME"));
```

```
Child table
--------------------------------
   CREATE TABLE "SYSTEM"."USERS_ROLES"
  (        "ROLES " VARCHAR2(20 BYTE),
           "UNAME" VARCHAR2(20 BYTE),
            CONSTRAINT "FK1" FOREIGN KEY ("UNAME")
             REFERENCES "SYSTEM"."USERS" ("UNAME") ENABLE));
```

Note: Taking user details and roles details in single DB table is bad practice. This approach does not support one user having multiple roles, so always prefer taking two DB tables having FK relationship.



Step 2: Create the project having following starters.

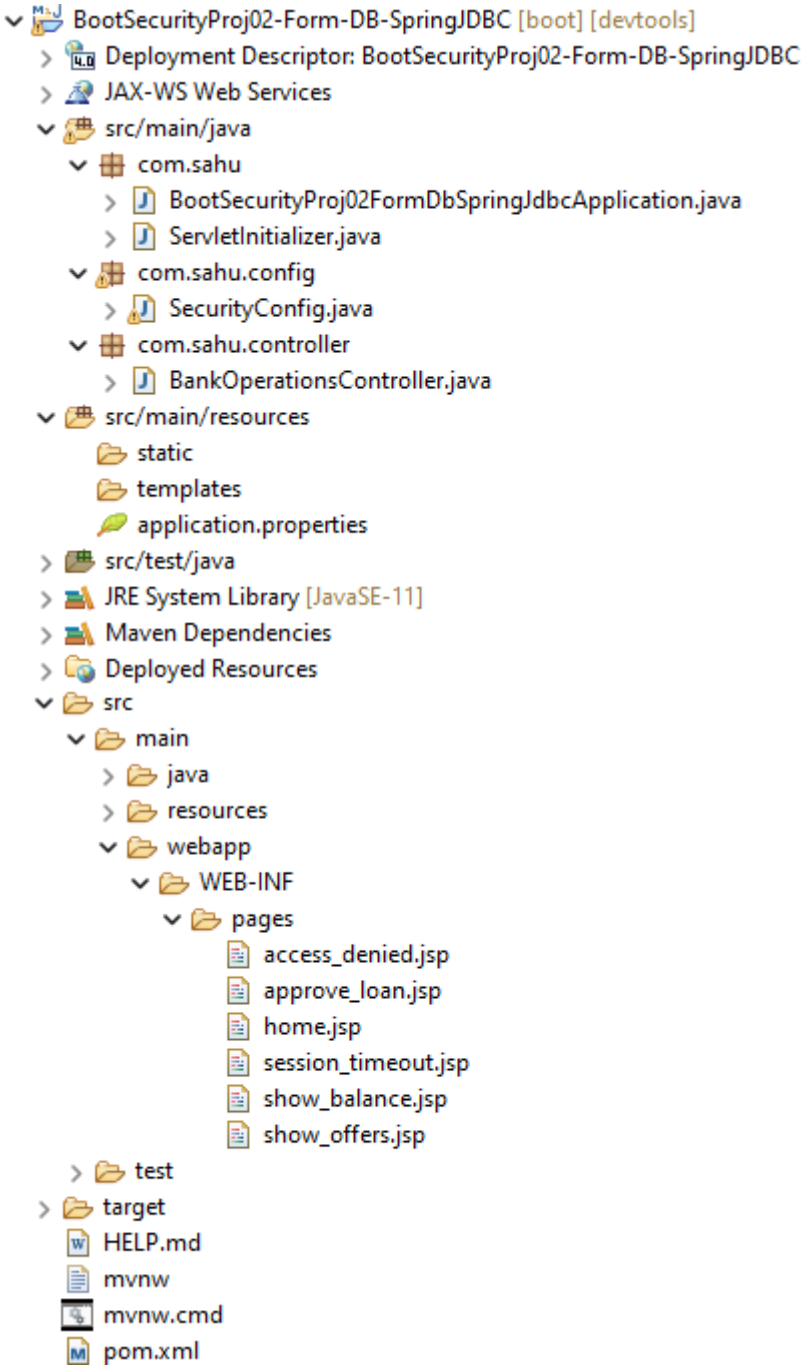
- X Spring Boot DevTools
- X JDBC API
- X Oracle Driver
- X Spring Security
- X Spring Web

Step 3: Add JDBC properties in application.properties file

Step 4: Copy past the required package and class from previous project and inject DataSource object to SecurityConfig class.

Step 5: Write the code in 1st configure(-) method to enable JDBC authentication.

Directory Structure of BootSecurityProj02-Form-DB-SpringJDBC:



- Develop the above directory structure using Spring Starter Project option and create the package, classes, folders and JSP files also.
- Use the following starters during project creation.

  X Spring Boot DevTools
  X JDBC API
  X Oracle Driver
  X Spring Security
  X Spring Web

- Copy and paste the required package, class from previous project.
- Then place the following code with in their respective files.

application.properties

```
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
#JDBC properties for data source
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
```

SecurityConfig.properties

```
@Configuration
@EnableWebSecurity //Makes the normal @Configuration class to Spring
Security configuration class
public class SecurityConfig extends WebSecurityConfigurerAdapter {

	@Autowired
	private DataSource dataSource;

	@Override
	public void configure(AuthenticationManagerBuilder auth) throws
Exception {

	auth.jdbcAuthentication().dataSource(dataSource).passwordEncoder(
new BCryptPasswordEncoder())
		.usersByUsernameQuery("SELECT UNAME, PWD, STATUS FROM
USERS WHERE UNAME=?") //For Authentication
		.authoritiesByUsernameQuery("SELECT UNAME, ROLES FROM
USER_ROLES WHERE UNAME=?"); //For Authorization
	}

	@Override
	public void configure(HttpSecurity http) throws Exception {
		//Provide logic for   Authentication and authorization and etc.
		http.authorizeRequests().antMatchers("/").permitAll() //No
authentication and no authorization
				.antMatchers("/offers").authenticated() //Only
authentication
```

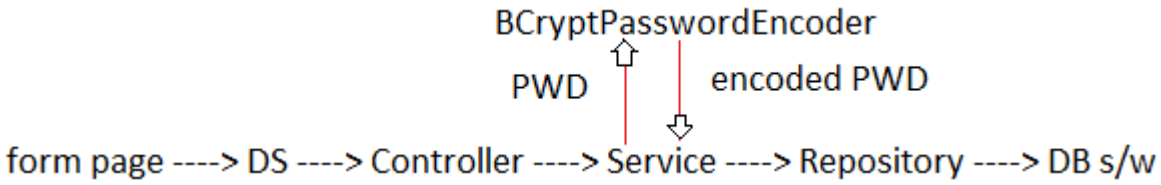
```
        .antMatchers("/balance").hasAnyAuthority("CUSTOMER",
"MANAGER") //authentication + authorization for "CUSTOMER",
"MANAGER" role users

        .antMatchers("/loanApprove").hasAnyAuthority("MANAGER")
//authentication + authorization for "MANAGER" role users
                                .anyRequest().authenticated() //Remaining all
requests URL must be authenticated
                                //.and().httpBasic() //Specify authentication mode
                                .and().formLogin().and().rememberMe() //enable
remember me option
                                .and().logout() //enable logout

        .and().exceptionHandling().accessDeniedPage("/denied")
//Exception/ error handling

        .and().sessionManagement().maximumSessions(2).maxSessionsPreve
ntsLogin(true).expiredUrl("/timeout");
        }

}
```

Note: This hasAnyAuthority() and hasAnyRole() methods are changing according to the first configure method's authorities() or roles() method.

Spring Boot Security with Spring Boot Data JPA

- There is no direct provision to work with Spring Data JPA or Spring ORM based authentication info provider i.e., we need to implement most of the logics manually as we do in other Spring Boot layered apps by taking separate repository interfaces, service classes, model classes and etc.
- auth.inMomeoryAuthentication(), auth.jdbcAuthentication() like this there is no direct template to work with Spring Data JPA (ORM) based Authentication provider.

User Registration



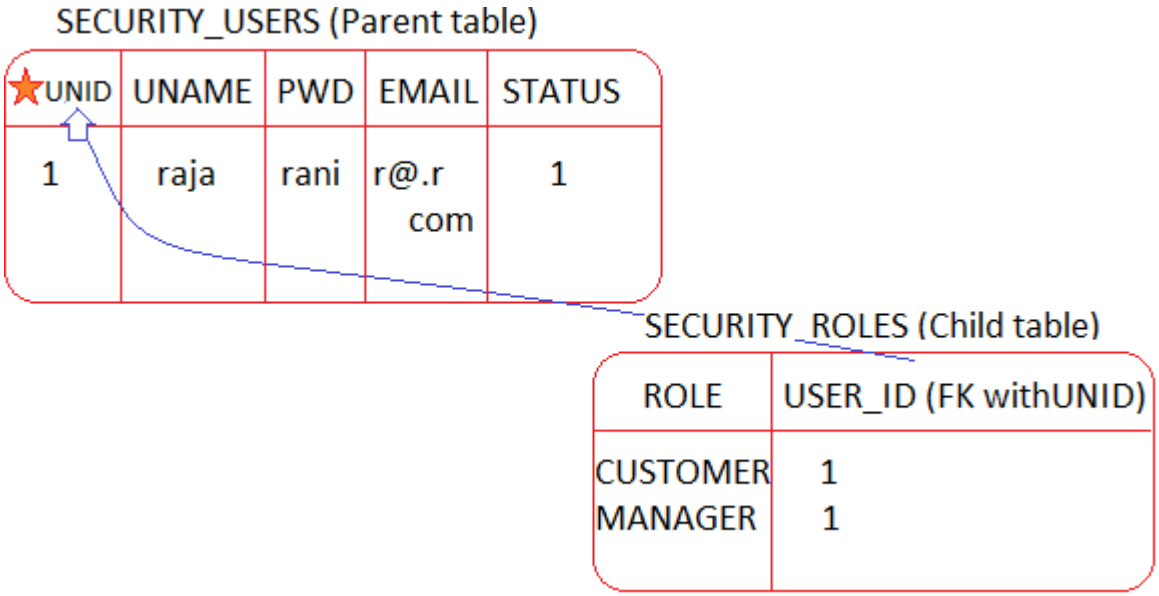
Note: To get currently logged in username from Spring Boot Security environment use the following code in JSP page,
<%=SecurityContextHolder.getContext().getAuthentication().getName()%>

Step 1: Create a Spring starter project by adding the following starters and

- X Spring Boot DevTools
- X Lombok
- X Spring Data JPA
- X Oracle Driver
- X Spring Security
- X Thymeleaf
- X Spring Web

Step 2: Create Thymeleaf HTML pages in src/main/resources/template folder.

Step 3: Create the Model class.



Note:

```
@ElementCollection(fetch = FetchType.EAGER)
@CollectionTable(name = "SECURITY_ROLES", joinColumns =
        @JoinColumn(name="USER_ID", referencedColumnName = "uid"))
@Column(name = "role")
private Set<String> roles;
```

- Having collection data for all objects of Entity class separate child table will created.
- Here collection mapping concept to maintain set collection values in child table.
- If needed you can go for One-to-many Associations mapping.

Step 4: Design form page in thymeleaf having user registration details.

Step 5: Develop Repository interface for UserDetails Model class.

Step 6: Develop Service interface and implementation class using the above Repository and a BCryptPasswordEncoder to encode the password.

Step 7: Develop separate Controller class for User registration, login activities having global path "/user".

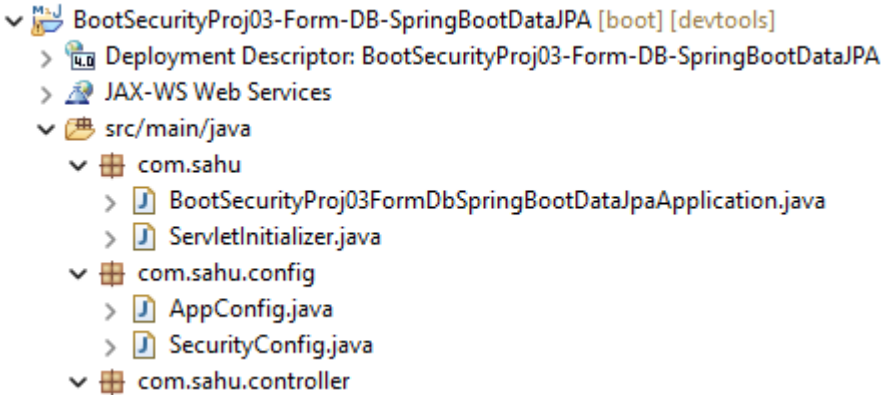
Step 8: Specify current service class as the Authentication info provider along with Password Encoder in SecurityConfig class.

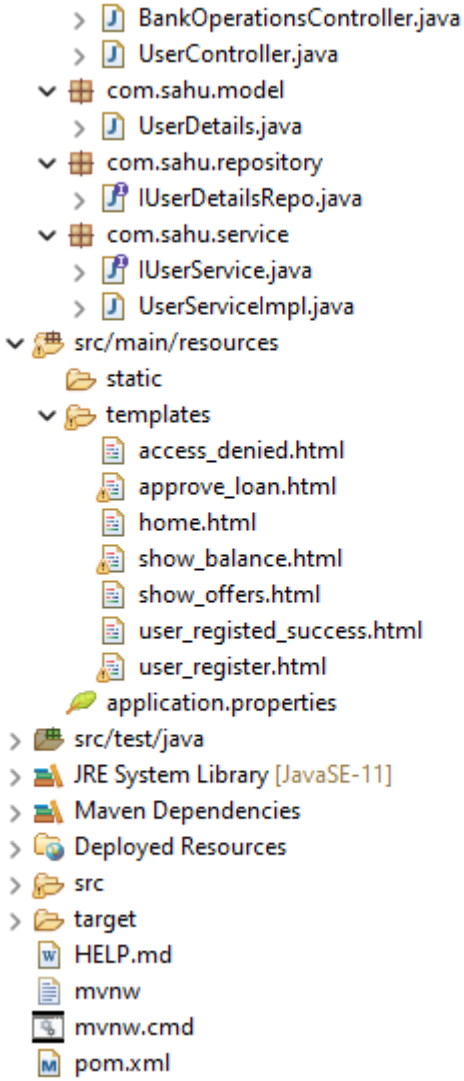
Step 9: Provide permitAll() access to "/user/register" URL.

Step 10: Add hyperlink in the home page for user registration.

Step 11: Specify DataSource, ORM properties in application.properties.

Directory Structure of BootSecurityProj03-Form-DB-SpringBootDataJPA:





- Develop the above directory structure using Spring Starter Project option packaging as war and create the package, classes, and HTML files.
- Use the following starters during project creation.

- Then place the following code with in their respective files.

application.properties

```
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp
#JDBC properties for data source
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
```

```
#Spring Data JPA Hibernate properties
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

home.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: red; text-align: center;">Welcome to XYZ Bank -- Home
page</h1>

<div style="text-align: center;">
        <a th:href="@{/bank/balance}">Check Balance</a>   
        <a th:href="@{/bank/offers}">Show Offers</a>  
        <a th:href="@{/bank/loanApprove}">Approve Loan</a>
        <br><br>
        <a th:href="@{/user/register}">Register User</a>
</div>
```

show_balance.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: blue; text-align: center;">Show Balance page</h1>
<div style="text-align: center;">
        <b>Balance : <span th:text="${balance}"/></b>
</div>

<div style="text-align: center;">
        <a th:href="@{/bank/}">Home</a>  
        <a th:href="@{/bank/offers}">Show Offers</a>   
        <a th:href="@{/bank/loanApprove}">Approve Loan</a> 
 
        <a th:href="@{/logout}">Logout</a>
</div>
```

show_offers.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: blue; text-align: center;">Show offers page</h1>
```

```
<div style="text-align: center;">
        Home Loan ROI: 7% <br>
        Four wheeler Loan ROI: 8% <br>
        Personal Loan ROI: 12%
</div>

<div style="text-align: center;">
        <a th:href="@{/bank/}">Home</a>  
        <a th:href="@{/bank/balance}">Check Balance</a>   
        <a th:href="@{/bank/loanApprove}">Approve Loan</a>
        <a th:href="@{/logout}">Logout</a>
</div>
```

user_register_success.html

```
<html xmlns:th="http://www.thymeleaf.org">

<div style=" text-align: center;">
        <h1 style="color: green;" th:text="${result}"></h1>
        <br>
        <a th:href="@{/user/login}">Login</a>
        <br><br>
        <a th:href="@{/user/register}">Add another User</a>
        <br><br>
        <a th:href="@{/bank/}">Home</a>
</div>
```

user_register.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: red; text-align: center;">User Registration page</h1>

<form th:action="@{/user/register}" method="POST"
th:object="${userInfo}">
        <table border="0" bgcolor="cyan" align="center">
                <tr>
                        <td>User Name:</td>
                        <td><input type="text" th:field="*{uname}"/></td>
                <tr>
                <tr>
```

```
                        <td>Password:</td>
                        <td><input type="password" th:field="*{pwd}"/></td>
                <tr>
                <tr>
                        <td>Email:</td>
                        <td><input type="email" th:field="*{email}"/></td>
                <tr>
                <tr>
                        <td>Roles:</td>
                        <td>
                                <input type="checkbox" th:field="*{roles}"
value="CUSTOMER" checked="checked"/>CUSTOMER
                                   
                                <input type="checkbox" th:field="*{roles}"
value="MANAGER" />MANAGER
                        </td>
                <tr>
                <tr>
                        <td></td>
                        <td colspan="2">
                                <input type="reset"  value="Cancel"/>
                                   
                                <input type="submit" value="Register"/>
                        </td>
                <tr>
        </table>
</form>
<br>
<div style=" text-align: center;">
        <a th:href="@{/bank/}">Home</a>
</div>
```

approve_loan.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: blue; text-align: center;">Loan Approval page</h1>
<div style="text-align: center;">
        <b>Your approved for Loan amount : <span
th:text="${amount}"/></b>
</div>
```

```
<div style="text-align: center;">
        <a th:href="@{/bank/}">Home</a>  
        <a th:href="@{/bank/balance}">Check Balance</a>   
        <a th:href="@{/bank/offers}">Show Offers</a>
        <a th:href="@{/logout}">Logout</a>
</div>
```

access_denied.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: red; text-align: center;">Authorization Failed</h1>

<br>
<div style="text-align: center;">
        <a th:href="@{/bank/}">Home</a>
</div>
```

UserDetails.java

```
package com.sahu.model;

import java.util.Set;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Data
@Table(name="SECURITY_USERS")
public class UserDetails {
```

```
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Integer unid;
	@Column(length = 20, nullable = false, unique = true)
	private String uname;
	@Column(length = 150, nullable = false)
	private String pwd;
	@Column(length = 30, nullable = false)
	private String email;
	private Boolean status = true;

	@ElementCollection(fetch = FetchType.EAGER)
	@CollectionTable(name = "SECURITY_ROLES", joinColumns =
@JoinColumn(name="USER_ID", referencedColumnName = "unid"))
	@Column(name = "roles")
	private Set<String> roles;
}
```

IUserDetailsRepo.java

```
package com.sahu.repository;

import org.springframework.data.repository.PagingAndSortingRepository;

import com.sahu.model.UserDetails;

public interface IUserDetailsRepo extends
PagingAndSortingRepository<UserDetails, Integer> {

}
```

AppConfig.java

```
package com.sahu.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
```

```
public class AppConfig {

	@Bean
	public BCryptPasswordEncoder createBCPEncoder() {
		return new BCryptPasswordEncoder();
	}

}
```

SecurityConfig.java

```
package com.sahu.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Aut
henticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableW
ebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSec
urityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
@EnableWebSecurity //Makes the normal @Configuration class to Spring
Security configuration class
public class SecurityConfig extends WebSecurityConfigurerAdapter {

	@Autowired
	private UserDetailsService service;

	@Autowired
	private BCryptPasswordEncoder encoder;
```

```
	@Override
	public void configure(AuthenticationManagerBuilder auth) throws
Exception {
		auth.userDetailsService(service).passwordEncoder(encoder);
	}

	@Override
	public void configure(HttpSecurity http) throws Exception {
		//Provide logic for   Authentication and authorization and etc.
		http.authorizeRequests().antMatchers("/bank/").permitAll()
//No authentication and no authorization
				.antMatchers("/user/register/").permitAll()
				.antMatchers("/bank/offers").authenticated()
//Only authentication

	.antMatchers("/bank/balance").hasAnyAuthority("CUSTOMER",
"MANAGER") //authentication + authorization for "CUSTOMER",
"MANAGER" role users

	.antMatchers("/bank/loanApprove").hasAnyAuthority("MANAGER")
//authentication + authorization for "MANAGER" role users
				.anyRequest().authenticated() //Remaining all
requests URL must be authenticated
				//.and().httpBasic() //Specify authentication mode
				.and().formLogin().defaultSuccessUrl("/bank/",
true)
				.and().rememberMe() //enable remember me
option
				.and().logout() //enable logout

	.and().exceptionHandling().accessDeniedPage("/bank/denied")
//Exception/ error handling

	.and().sessionManagement().maximumSessions(2).maxSessionsPreve
ntsLogin(true).expiredUrl("/timeout");
	}

}
```

IUserService.java

```
package com.sahu.service;

import org.springframework.security.core.userdetails.UserDetailsService;

import com.sahu.model.UserDetails;

public interface IUserService extends UserDetailsService {
	public String registerUser(UserDetails userDetails);
}
```

UserServiceImpl.java

```
package com.sahu.service;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.userdetails.UsernameNotFoundExceptio
n;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import com.sahu.model.UserDetails;
import com.sahu.repository.IUserDetailsRepo;

@Service("userServie")
public class UserServiceImpl implements IUserService {

	@Autowired
	private IUserDetailsRepo userDetailsRepo;

	@Autowired
	private BCryptPasswordEncoder passwordEncoder;

	@Override
	public String registerUser(UserDetails userDetails) {

	userDetails.setPwd(passwordEncoder.encode(userDetails.getPwd()));
		return userDetailsRepo.save(userDetails).getUname()+" details
has registered";
```

```
has registered";
	}

	@Override
	public org.springframework.security.core.userdetails.UserDetails
loadUserByUsername(String username)
			throws UsernameNotFoundException {
		return null;
	}

}
```

UserController.java

```
package com.sahu.controller;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import com.sahu.model.UserDetails;
import com.sahu.service.IUserService;

@Controller
@RequestMapping("/user")
public class UserController {

	@Autowired
	private IUserService userService;

	@GetMapping("/register")
	public String showUserRegisterForm(@ModelAttribute("userInfo")
UserDetails details) {
		return "user_register";
	}
```

```
	@PostMapping("/register")
	public String registerUserDetails(Map<String, Object> map,
@ModelAttribute("userInfo") UserDetails details) {
		//Use service
		String resultMsg = userService.registerUser(details);
		map.put("result", resultMsg);
		return "user_registed_success";
	}

	@GetMapping("/login")
	public String showLogin() {
		//return LVN
		return "user_login";
	}

}
```

BankOperationsController.java

```
package com.sahu.controller;

import java.util.Map;
import java.util.Random;

import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/bank")
public class BankOperationsController {

	@GetMapping("/")
	public String showHome() {
		return "home";
	}

	@GetMapping("/offers")
	public String showOffers() {
		return "show  offers";
```

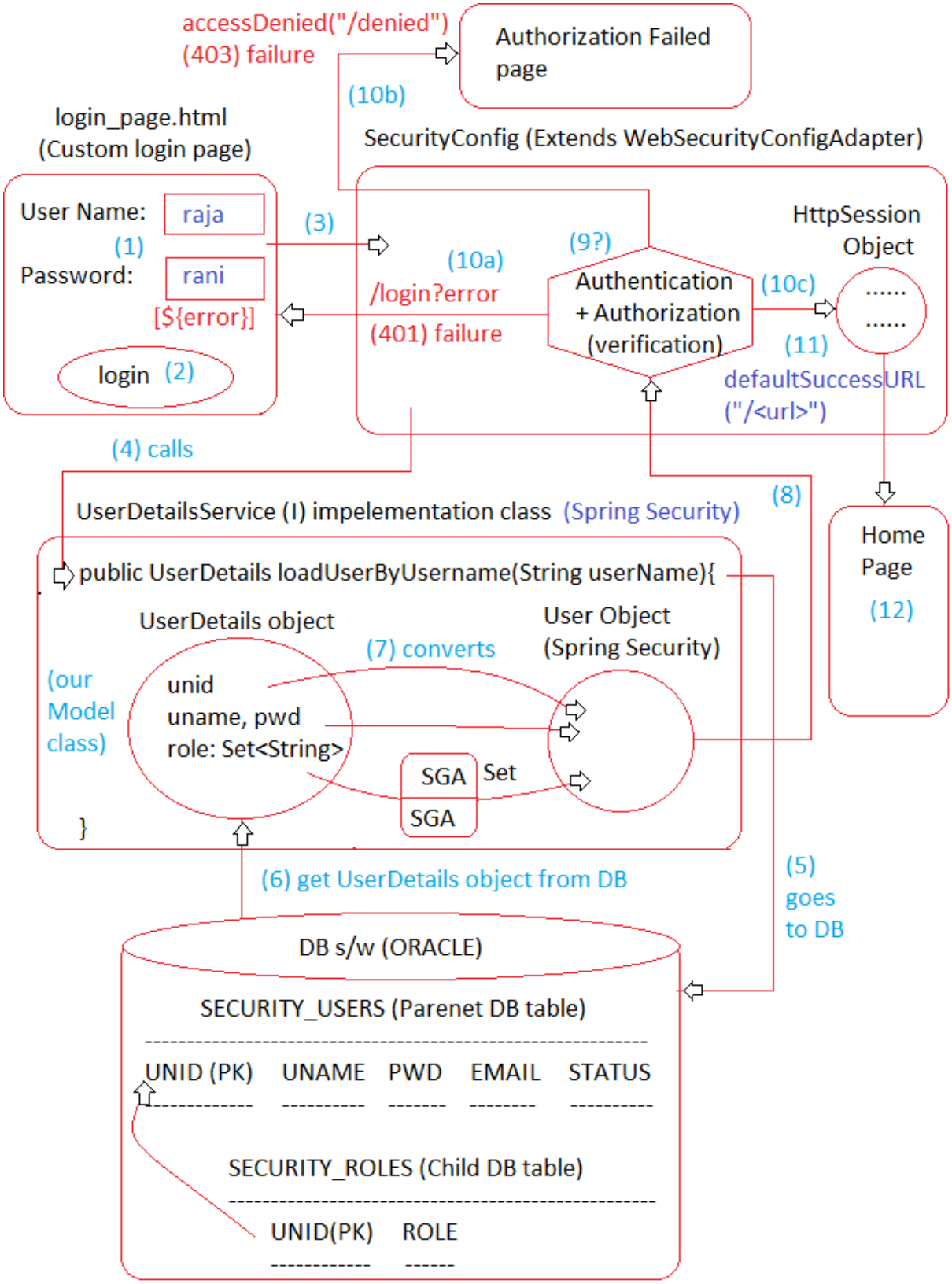
```
        }

        @GetMapping("/balance")
        public String checkBalance(Map<String, Object> map) {
                map.put("balance", new Random().nextInt(1000000000));
                return "show_balance";
        }

        @GetMapping("/loanApprove")
        public String approveLoan(Map<String, Object> map) {
                map.put("amount", new Random().nextInt(1000000000));

                return "approve_loan";
        }

        @GetMapping("/denied")
        public String accessDenied(Map<String, Object> map) {
                map.put("userName",
SecurityContextHolder.getContext().getAuthentication().getName());
                return "access_denied";
        }

}
```

- Here we need to configure service class that implements org.springframework.security.core.userdetails.UserDetailsService (I) directly or indirectly.
- When give POST + /login request to Spring Boot Security app, then the SecurityConfig class sends to request to loadUserByUsername(-) method of the above service class that method returns org.springframework.security.core.userdetails.User object that implements org.springframework.security.core.userdetails.UserDetails (I) having current logged user details like username, password, roles and etc. to SecurityConfig class for validation/ verification.
- If the verification success, then it keeps login details in HttpSession and show defaultSucessUrl page otherwise shows login error page.

org.springframework.security.core.userdetails.UserDetails(I)

⇧

org.springframework.security.core.userdetails.User



org.springframework.security.core.GantedAuthority (I) [GA]

⇧

org.springframework.security.core.authority.SimpleGrantedAuthority (c) [SGA]

Adding custom login page to the project and performing Authentication + Authorization using Spring Data JPA UserDetails Service

Step 1: First understand different default URLs

- /login + GET: To show default form-based authentication login page.
- /login + POST: To process default form-based authentication login page submission.
- /login?error + 401 status code: If authentication fails.
- <url> + 403 status code: If authorization fails.
- /login?logout: default logout success URL.

Step 2: Add Handler method in UserController class to show custom login page

Step 3: In SecurityConfig class

- add permitAll() for "/user/showLogin"
- add "/user/showLogin" as loginPage(-) URL
- Here /user is global path of controller class

Step 4: Develop custom_login.html as the customer login page in main/java/resources/templates folder.

- Textbox names should be username, password.
- Action URL should be "/login".
- These are fixed as long as we want the Spring Boot security's readymade Authentication Manager, Authorization Manager.

Step 5: Declare the finder method in repository interface to get UserDetails model class object based on the given username.

Step 6: Write the logic in loadUserByUsername(-) in UserDetailsService implementation to get Model class object (UserDetails object) and to convert into User class object (Spring security class object).

Step 7: Specify multiple URLs related form login authentication in Spring SecurityConfig class.

Step 8: In form page read and display “error”, “logout” request param values.

UserController.java

```
@GetMapping("/showLogin")
public String showLoginPage() {
        return "custom_login";
}
```

SecurityConfig.java

```
	@Override
	public void configure(HttpSecurity http) throws Exception {
		//Provide logic for   Authentication and authorization and etc.
		http.authorizeRequests().antMatchers("/bank/").permitAll()
//No authentication and no authorization
				.antMatchers("/user/register/",
"/user/showLogin").permitAll()
				.antMatchers("/bank/offers").authenticated()
//Only authentication

	.antMatchers("/bank/balance").hasAnyAuthority("CUSTOMER",
"MANAGER") //authentication + authorization for "CUSTOMER",
"MANAGER" role users

	.antMatchers("/bank/loanApprove").hasAnyAuthority("MANAGER")
//authentication + authorization for "MANAGER" role users
				.anyRequest().authenticated() //Remaining all
requests URL must be authenticated
				//.and().httpBasic() //Specify authentication mode
				.and().formLogin().defaultSuccessUrl("/bank/",
true)
				.loginPage("/user/showLogin") //For Get mode
request to launch form page
				.loginProcessingUrl("/login") //for POST mode
request  to submit and process the request
				.failureUrl("/user/showLogin?error")
//Authentication failedURL
				.and().rememberMe() //enable remember me
option
				.and().logout() //enable logout
				.logoutSuccessUrl("/user/showLogin?logout")
//After logout URL

	.and().exceptionHandling().accessDeniedPage("/bank/denied")
//Exception/ error handling

	.and().sessionManagement().maximumSessions(2).maxSessionsPreve
ntsLogin(true);
	}
```

custom_login.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: blue; text-align: center;">Login page</h1>

<form th:action="@{/login}" method="POST">
	<table border="0" bgcolor="cyan" align="center">
		<tr>
			<td>User Name:</td>
			<td><input type="text" name="username"/></td>
		<tr>
		<tr>
			<td>Password:</td>
			<td><input type="password" name="password"/></td>
		<tr>
		<tr>
			<td colspan="2" align="center">
				<input type="submit" value="login"/>
			</td>
		<tr>
	</table>
	<span th:if="${param.error}">Invalid Login details (Authentication
failed)</span>
	<span th:if="${param.logout}">User Logout successfully</span>
</form>
```

IUserDetailsRepo.java

```
	public Optional<UserDetails> findByUname(String uname);
```

IUserDetailsRepo.java

```
	@Override
	public org.springframework.security.core.userdetails.UserDetails
loadUserByUsername(String username)
			throws UsernameNotFoundException {
		//Get Model class object (com.sahu.model.UserDetails)
		Optional<com.sahu.model.UserDetails> opt =
userDetailsRepo.findByUname(username);
		if (opt.isEmpty())
			throw new IllegalArgumentException("User not found");
```

```
                else {
                        com.sahu.model.UserDetails details = opt.get();
                        /*//convert Set<String> type roles to set<SGA> type
roles
                        Set<GrantedAuthority> roles = new HashSet<>();
                        for (String role : details.getRoles()) {
                                SimpleGrantedAuthority authority = new
SimpleGrantedAuthority(role);
                                roles.add(authority);
                        }
                        //Convert model class
object(com.sahu.model.UserDetails) to Spring security User object
                        User user = new User(details.getUname(),
details.getPwd(), roles);*/
                        User user = new User(details.getUname(),
                                        details.getPwd(),
                                        details.getRoles().stream().map(role->new
SimpleGrantedAuthority(role)).collect(Collectors.toSet()));
                        return user;
                }
        }
```

CSRF Problem & Solution

- CSRF: Cross Site Requesting Forgery (fishing).
- It fishing or hacking technique of hackers or attackers who makes the innocent end-user sending his data to user sites and accounts.
- E.g., send spam emails, trapping emails and etc.
- For different website the attacker make victim to send following details by showing lottery ticket benefit.

```
<form action="https:///icicibank.com/trasferFunds">
        <input type="hidden" name="amount" value="10000">
        <input type="hidden" name="srcAccount" value="4455666">
        <input type="hidden" name="destAccount" value="6567788">
        <input type="submit" value="Click here to win the Lottery">
</form>                                (or)
<a
href="https://icicibank.con/trasnferfunds?amount=10000&srcAccount
=4455666&destAccount=6567788"> Click here to win the lottery </a>
```

Feeling CSRF Problem Practically

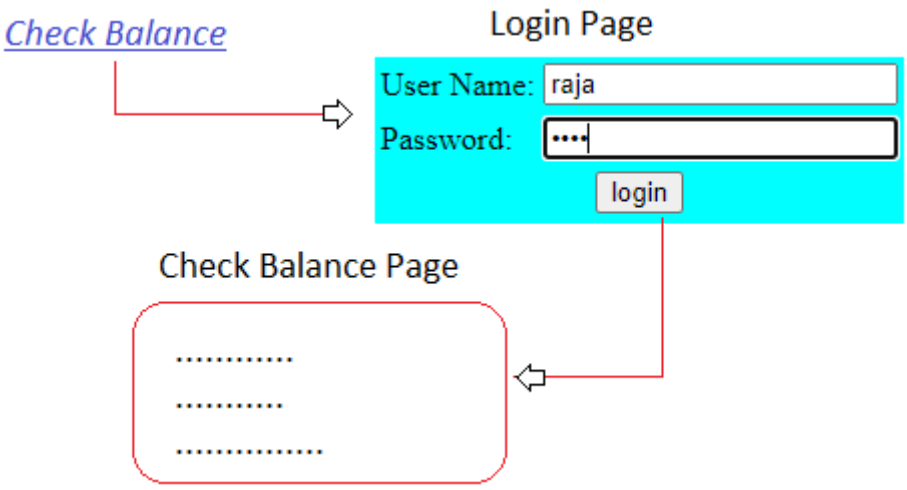
Step 1: Disable CRSF protection in Spring Boot Security Application

SecurityConfig.java

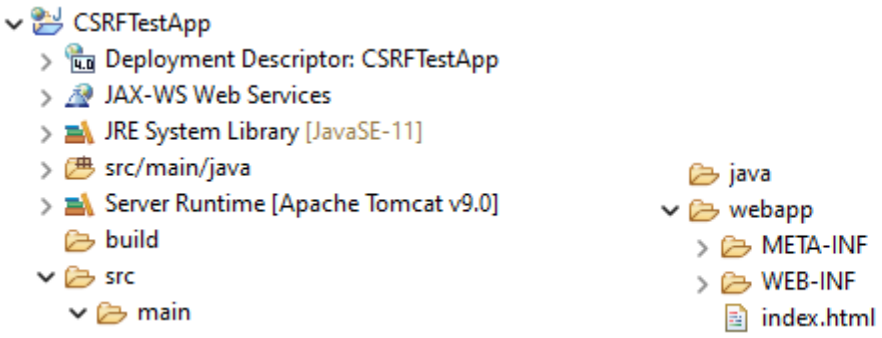
```
@Override
public void configure(HttpSecurity http) throws Exception {
        ……————————————
        //Disable or enable CSRF protection by default it is enabled
        http.csrf().disable();
}
```

Step 2: Run Spring Boot Security original application, complete authentication and to use one or two services (like check balance service by using "raja" user) and do not logout.

URL: http://localhost:2525/BootSecurityProj03-Form-DB-SpringBootDataJPA/bank/balance



Step 3: Create another normal Dynamic web project having the following code in any HTML page (index.html).



index.html

```
<form action="http://localhost:2525/BootSecurityProj03-Form-DB-SpringBootDataJPA/bank/balance" method="POST">
        <input type="submit" value="Click Here to Win the Lottery">
</form>
```

Step 4: Run new application along with old application from the old browser window and feel the CSRF or fishing problem by clicking on the hyperlink.

Note:

- ✓ GET mode request is read only request i.e., it does not do anything in the server whereas the POST mode request is updating the request i.e., they change data of the server environment.
- ✓ CSRF problem can be solved only for the request URLs that are expecting POST requests.

Solving CSRF Problem

Step 1: Make CSRF is not disabled in SecurityConfig. By default, CSRF protection is enabled in Spring Boot class

```
//http.csrf().disable();
```

Step 2: Add CSRF related token to the login form page and to the other form page by taking the support hidden box

custom_login.html

```
<input type="hidden" th:name="${_csrf.parameterName}"
th:value="${_csrf.token}"/>
```

Step 3: In Another web application's index.html (any html page) try to send the request

index.html

```
<form action="http://localhost:2525/BootSecurityProj03-Form-DB-SpringBootDataJPA/bank/balance" method="POST">
        <input type="submit" value="Click Here to Win the Lottery">
</form>
```

Make sure this URL based handler method is @PostMapping in the controller class as shown below.

```
@PostMapping("/balance")
public String checkBalance(Map<String, Object> map) {
        map.put("balance", new Random().nextInt(1000000000));
        return "show_balance";
}
```

Q. How does CSRF protection works internally?

Ans.

- When CSRF protection is enabled (it is by default Spring security/ Spring Boot security application) one session token will be created as session attribute having "_csrf" as token name and "32" digits hexa-decimal number token value. Using the following hidden box

```
<input type="hidden" th:name="${_csrf.parameterName}"
th:value="${_csrf.token}"/>
```

- We try to get csrf token name and token value to the form page and we send them along with form submission.
- The Security environment of server side takes the token name and value coming from browser and validates with already available session token value, if matched further activities will be allowed. If request comes with invalid token or no token then error will be raised.

Spring Boot Security using LDAP Server

Configuring LDAP Server

Installation, Server, Organization, Entry, Role, User creation

Step 1: Install Apache Directory studio [Download] and LDAP Server and creates users having roles by following given document [Spring Security LDAP Converted] and you can follow the below steps.

Step 2: After installation, lunch the Apache Directory Studio.

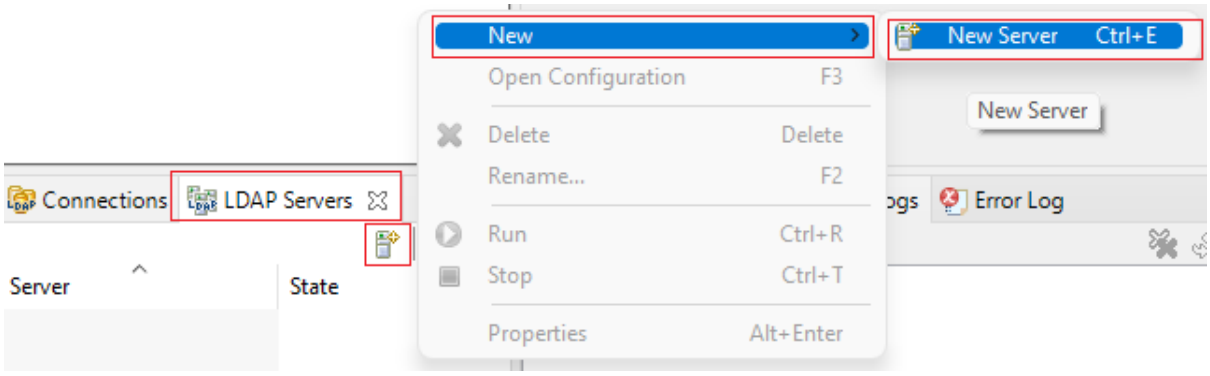
You can install in 2 ways,

- By using installer (.exe file).
- By using ZIP extraction

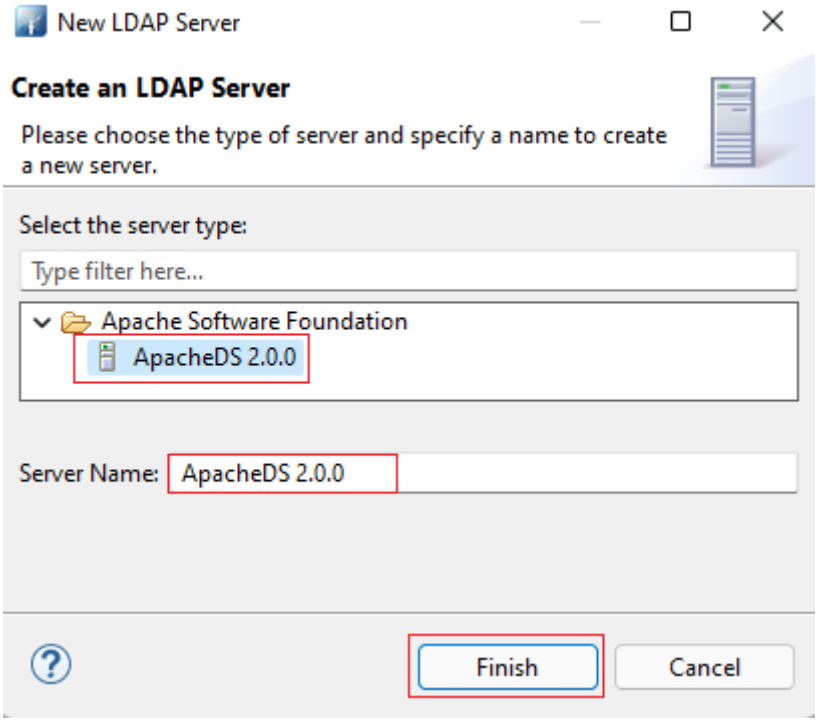
Step 3: For adding a new LDAP server, go to LDAP Servers window right click there, then click on New and New Server.

(or)

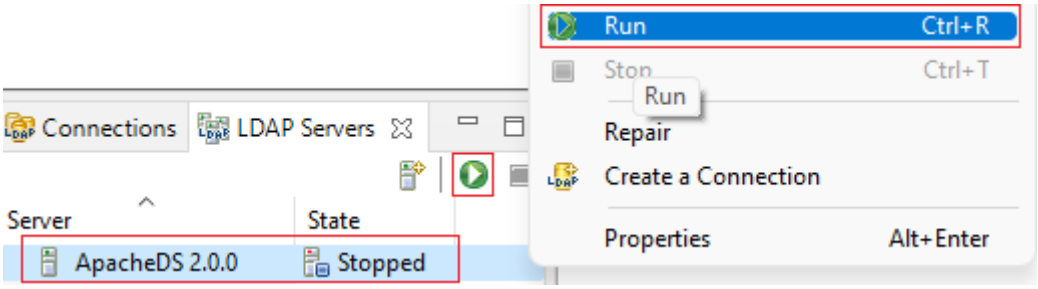
You can directly click the Add symbol. 



Step 4: Then the wizard will open choose ApacheDS 2.0.0 then will show in Server Name then click on finish



Step 5: Now you can see the server. Start the server, right click on the serve then click on Run otherwise click on the Run button.

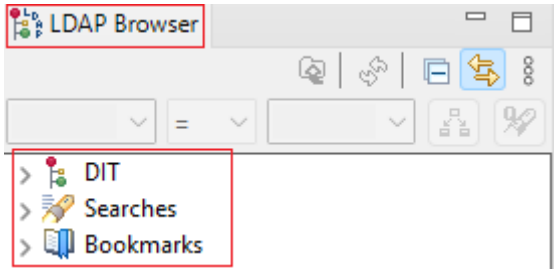


Note: Make your server mode during all the operation.

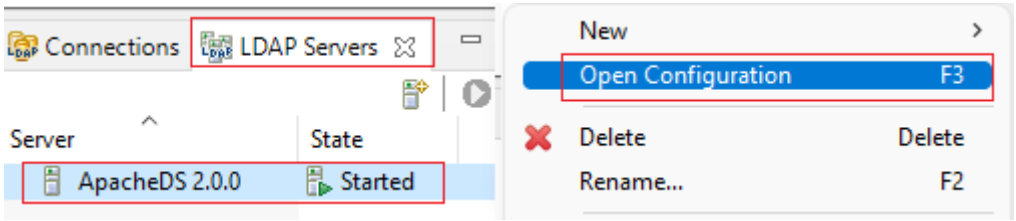
Step 6: Create a collection, for that right click on the Server then click on Create a Connection.

- Otherwise go to Connection's window the click in New Connection… button.

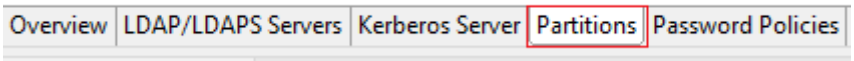
- Then you can see the Connection in LADP browser window.



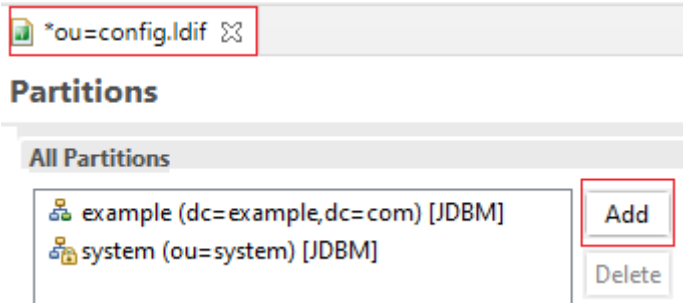
Step 7: Go to the configuration by right click on the server then click on Open Configuration.



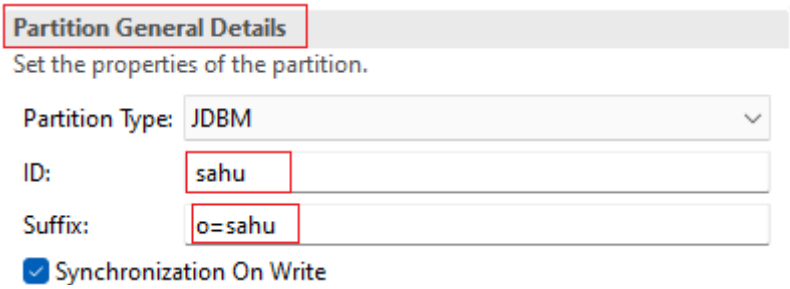
Step 8: Then go to the Partitions section.



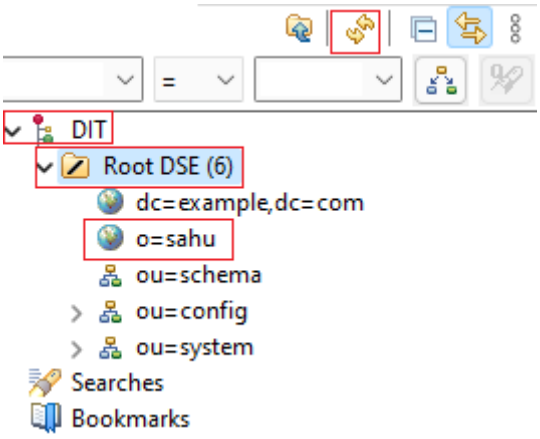
Step 9: To add a Partition there is a button Add click on that.



Step 10: Give/ fill the following details and save the file

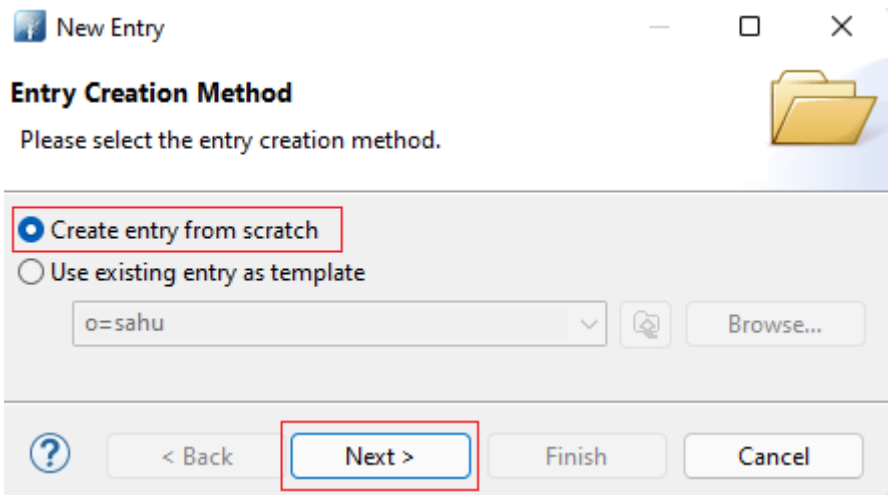


Step 11: After that restart the server and refresh the Root DSE then out organization will appear there.

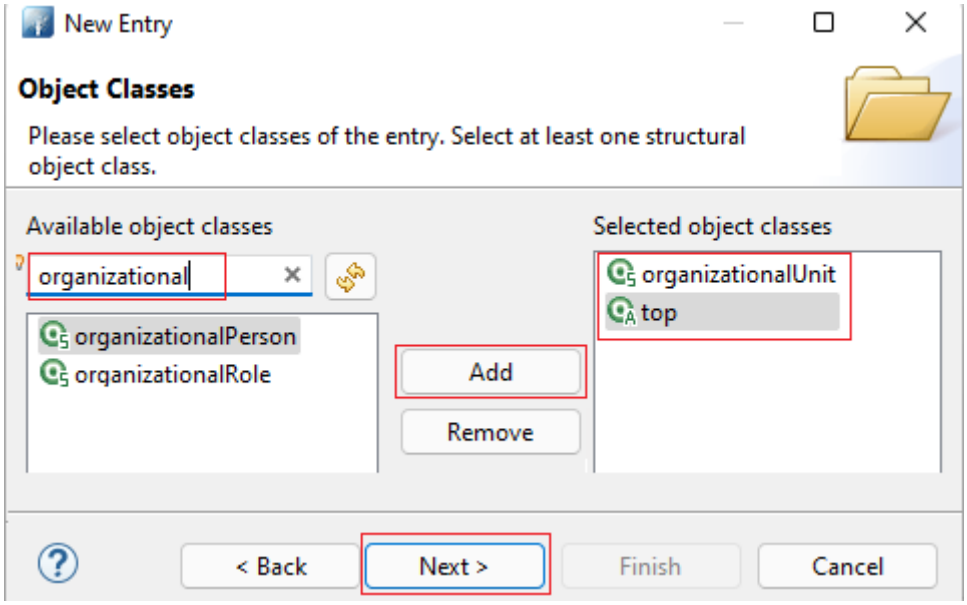


Step 12: To add an Entry right click on our organization then click on New then click on New Entry…

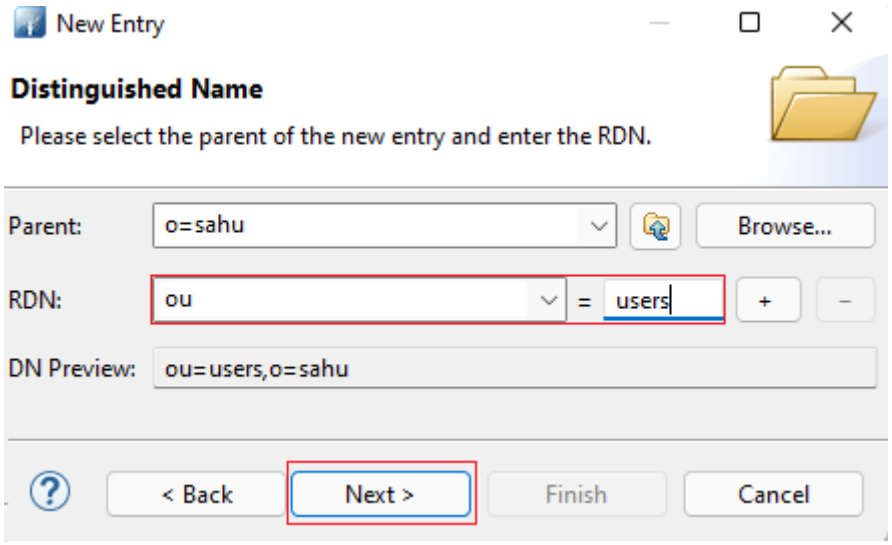
Step 13: Then choose Create entry from scratch (default) then click on Next.



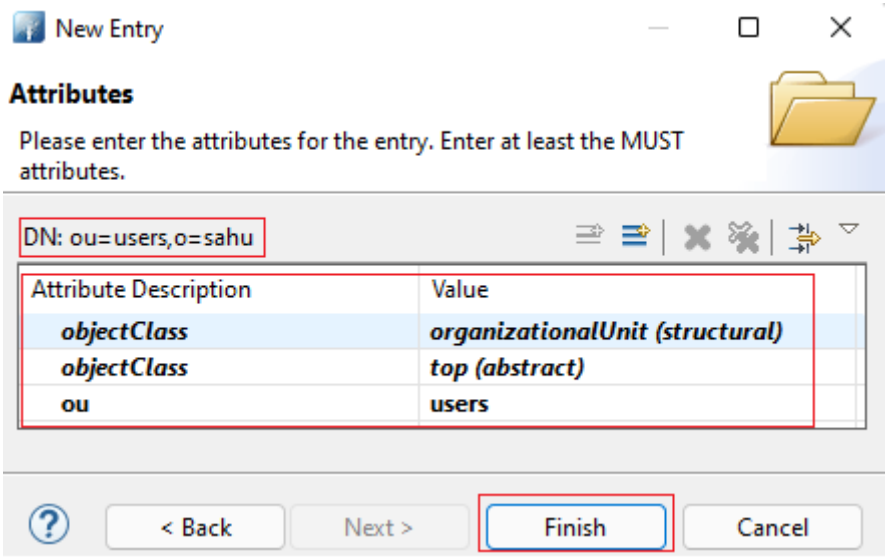
Step 14: Then search organizationalUnit then click on Add then click on Next.



Step 15: Give RDN: ou=users then click on Next

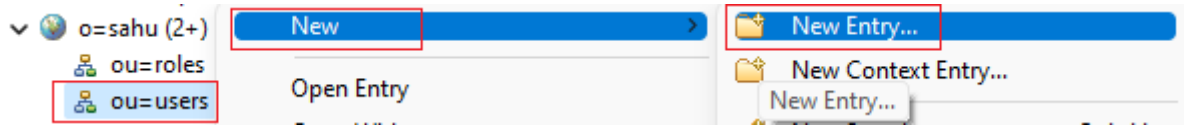


Step 16: Now click on Finish the Entry will create.

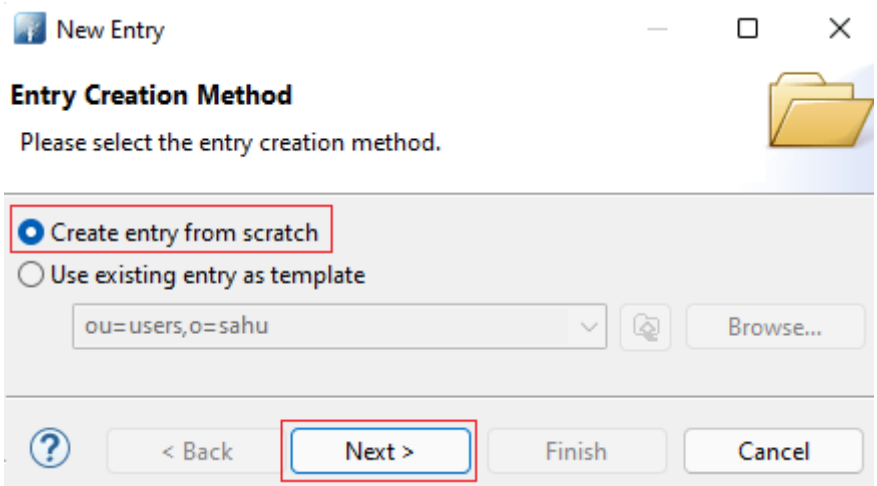


Step 17: Follow the 12 to 16 and create another New Entry i.e., ou=roles.

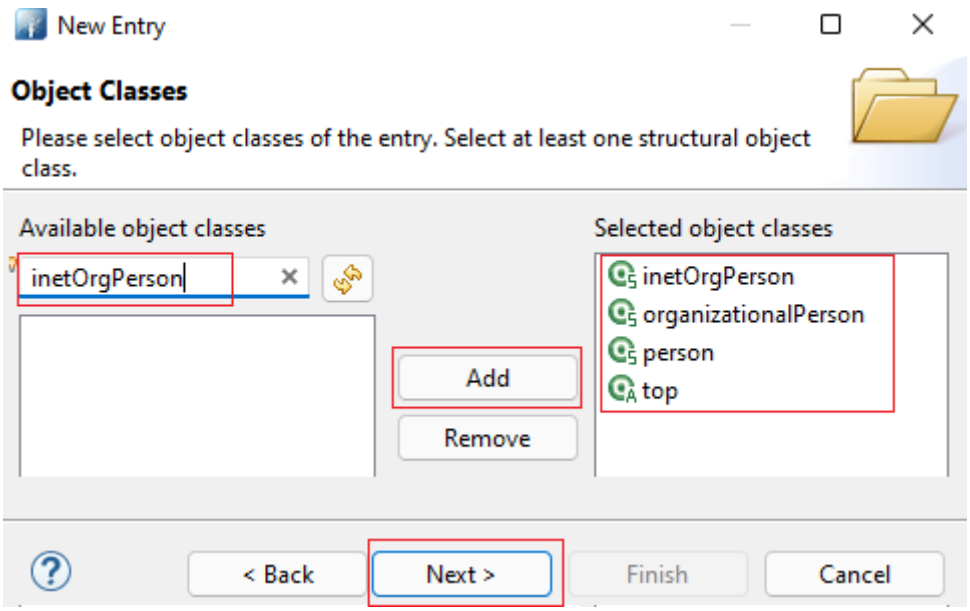
Step 18: To add a user right click on ou=users, click on New then click on New Entry…



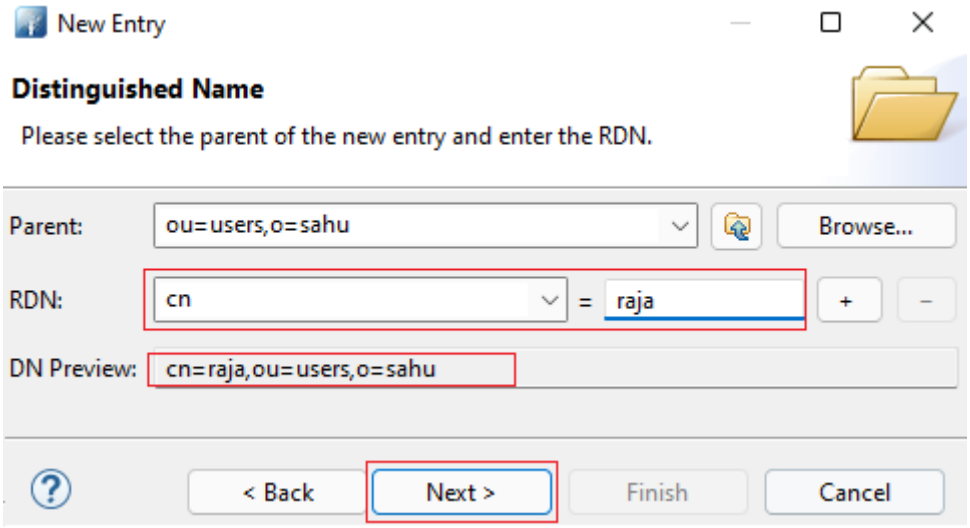
Step 19: Click on Create entry from scratch (default) then click on Next.



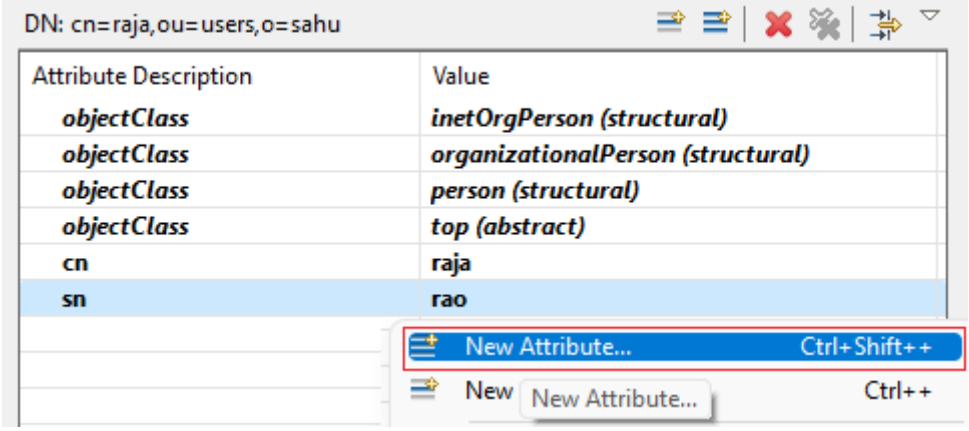
Step 20: Search inetOrgPerson, after that add that one by clicking add button now click on Next.



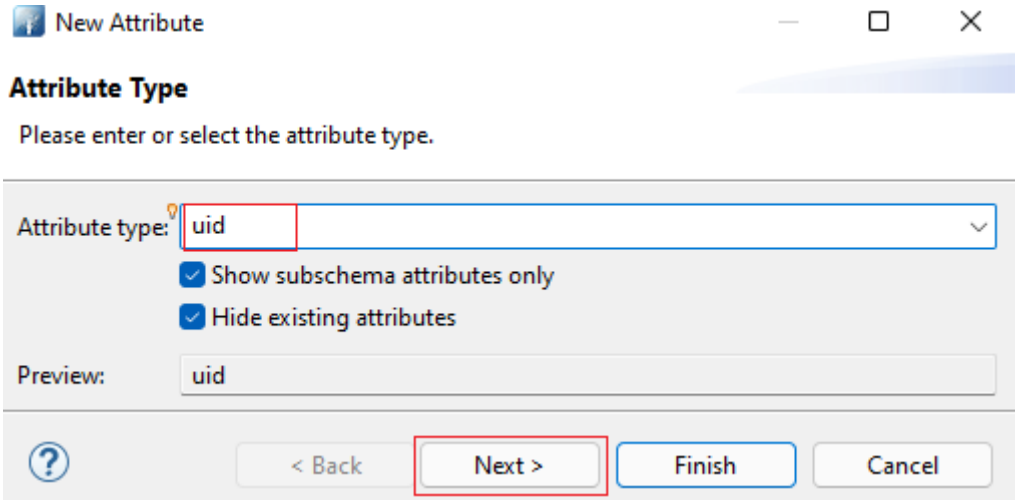
Step 21: Give the RDN: cn=raja then click on Next.



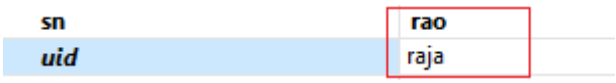
Step 22: To add an attribute right click, then click on New Attribute…



Step 23: Search or choose uid as Attribute type then click on Next, & Finish.

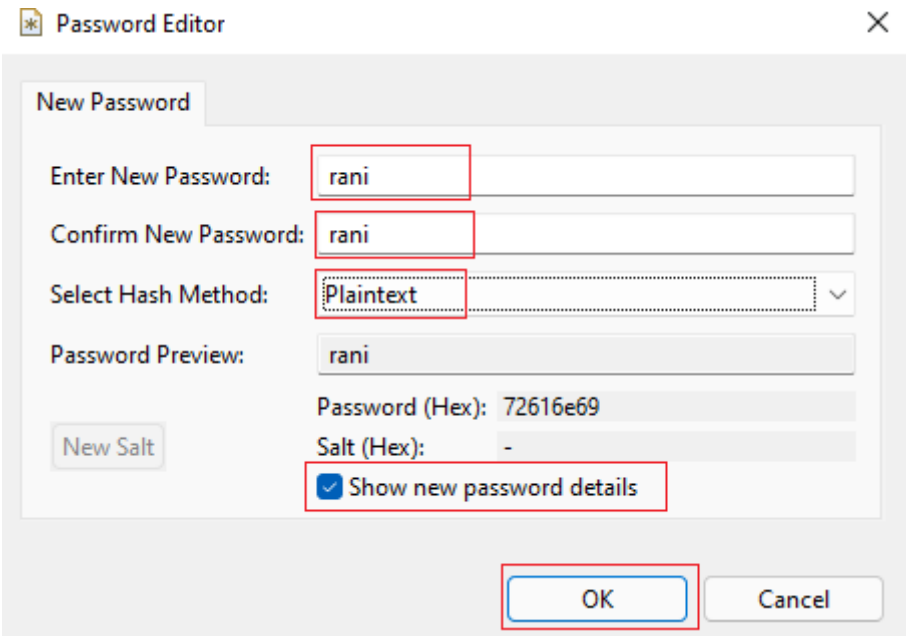


Step 24: Give the value sn=rao and uid = raja.



Step 25: Follow step 22 and 23 to add another attribute i.e., userPassword.

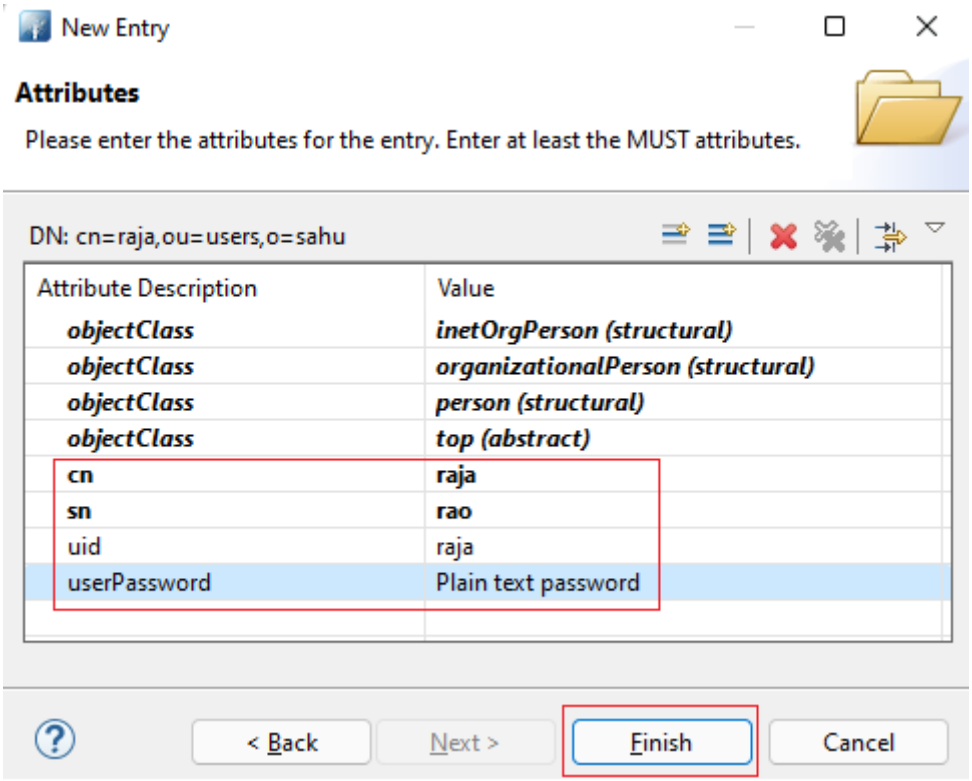
Step 26: After add that attribute a Password Editor fill the New Password, Confirm New Password, then click on OK.



Note: You can change the Hash method by choose different method from Select Hash Method dropdown, and you can see the password details by

choose Show new password details check box.

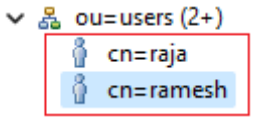
Step 27: Now you can see the below details them click on Finish, now the user is added with all the details.



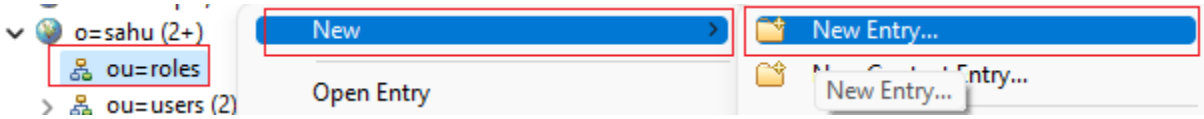
Step 28: Follow steps 18 to 27 and create another user of the following details

cn	**ramesh**
sn	**kumar**
uid	ramesh
userPassword	Plain text password

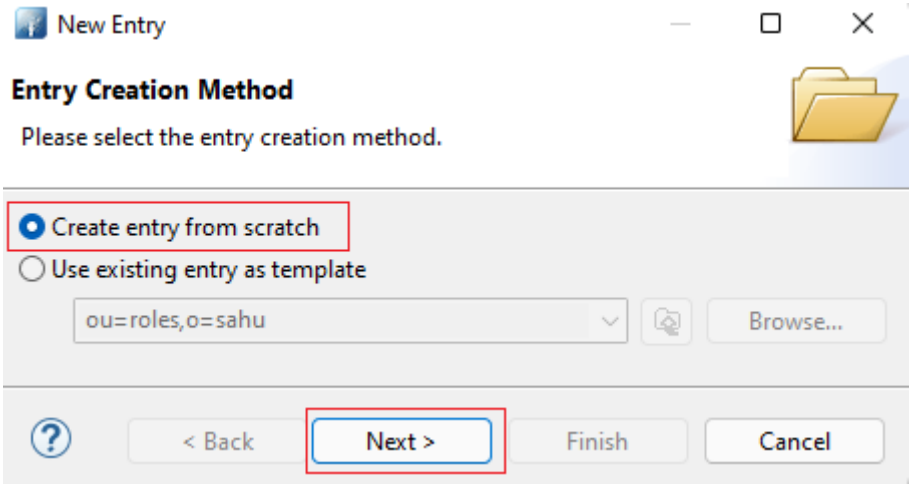
Step 29: After created two user you can see under ou=users tab.



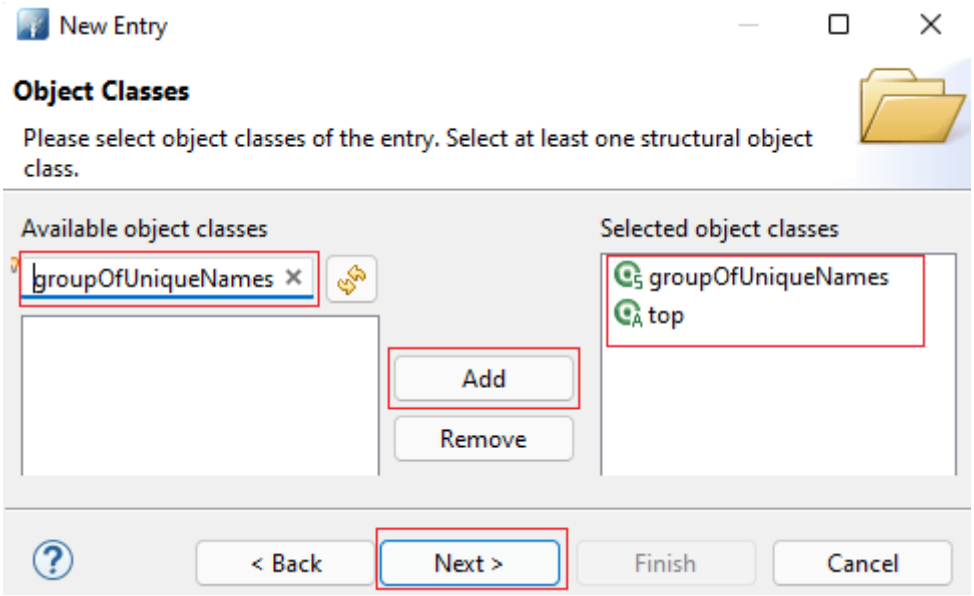
Step 30: To create roles right click on ou=roles then click on New then click on New Entry…



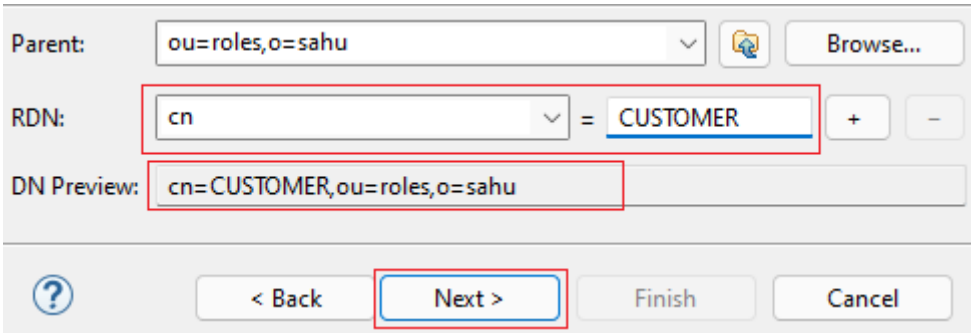
Step 31: Click on Create entry from scratch (default) then click on Next.



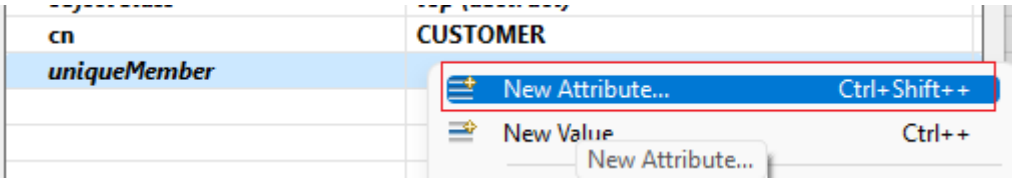
Step 32: Search groupOfUniqueNames, after that add that one by clicking add button now click on Next.



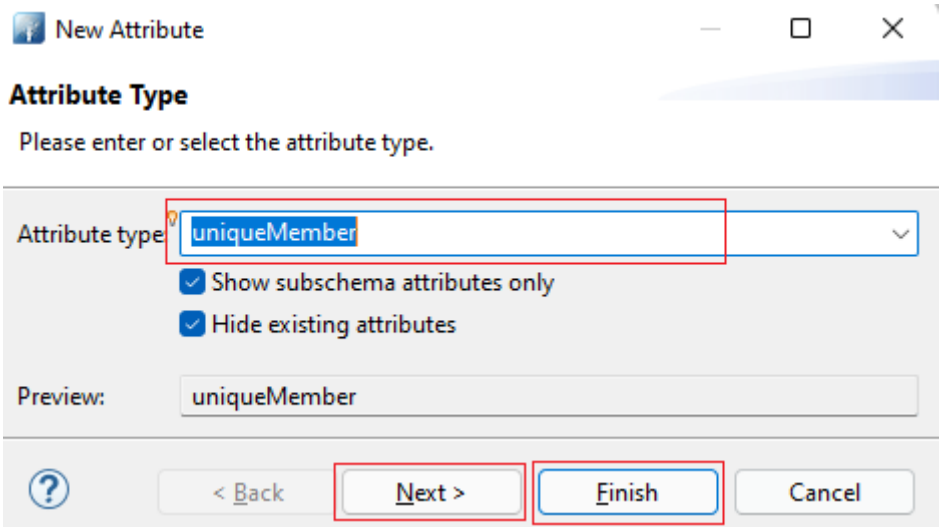
Step 33: Give the RDN: cn=CUSTOMER then click on Next.



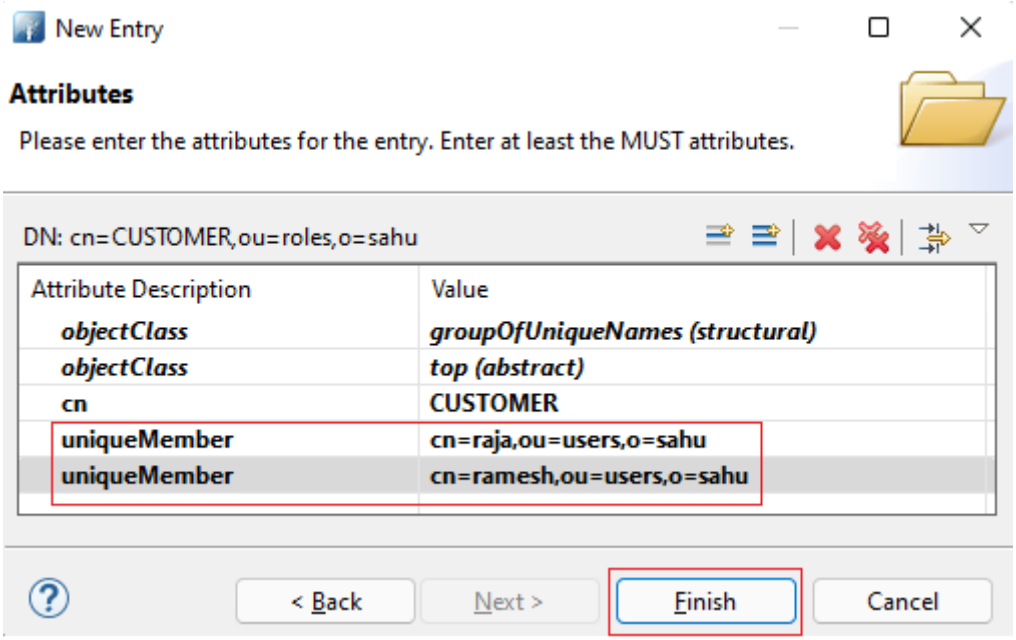
Step 34: To add an attribute right click, then click on New Attribute…



Step 35: Search or choose uniqueMember as Attribute type then click on Next, & Finish.

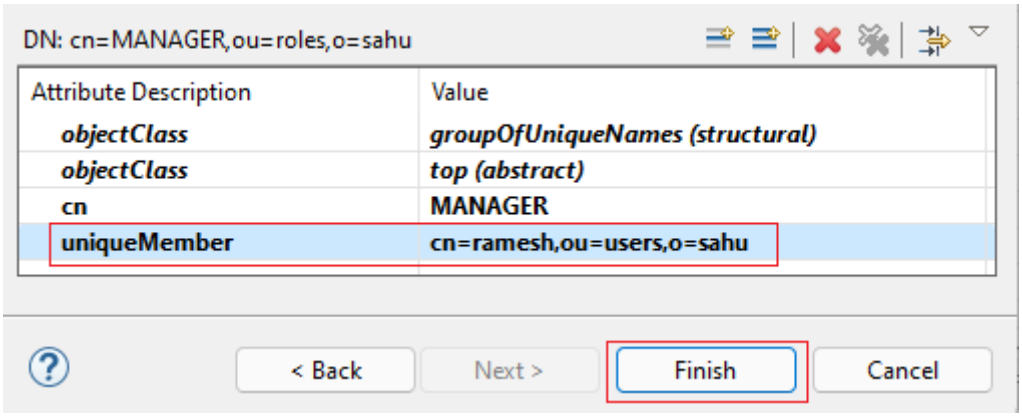


Step 36: Give the value of uniqueMember attribute as shown below then click on Finish.

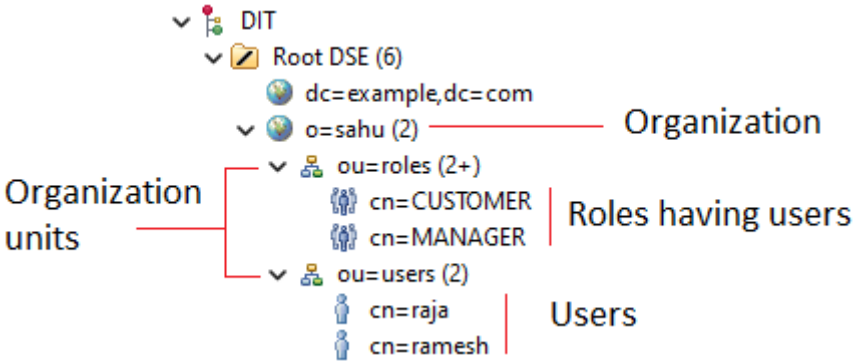


Step 37: Follow steps 30 to 33 to add another attribute role i.e., cn=MANAGER.

Step 38: Fill the uniqueMember details as shown below then click on Finish.



Everything has done, make sure you will get the following structure.

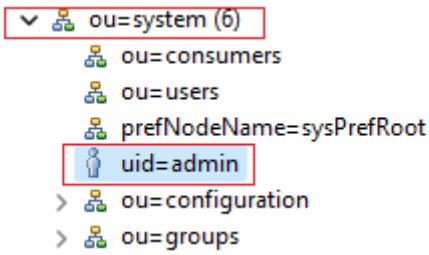


Configuring LDAP server as Authentication provider in Spring Boot Security application

Step 1: Follow the above steps to set your LDAP server.

Step 2: Try to gather LDAP Server URL and admin user name and password, for that follow the below sub steps.

- To collect username and password got ou=system and uid= admin then



- Then you will get the below information on side then the uid is you

username and userPassword is your password by double click on that a check on the Show current password details checkbox you can get the password details.

DN: uid=admin,ou=system

Attribute Description	Value
objectClass	*inetOrgPerson (structural)*
objectClass	*organizationalPerson (structural)*
objectClass	*person (structural)*
objectClass	*top (abstract)*
cn	**system administrator**
sn	**administrator**
displayName	Directory Superuser
uid	admin
userPassword	Plain text password

- To collect the URL, go to Modification Logs or Search Logs then you will get there and by click on the server also in overview tab you will get the port number.

Modification Logs | Search Logs | Error Log

```
#!RESULT OK
#!CONNECTION ldap://localhost:10389
#!DATE 2022-02-17T17:47:49.459
```

- By default, Details
  URL: ldap://localhost:10389
  Username: admin
  Password: secret

Step 3: Keep any Spring Boot Security application ready.

Step 4: Add the Spring Security LDAP dependency (jar dependency from MVN repository) pom.xml

Step 5: Make sure that Apache directory studio's Apache DS server is in running mode

Step 6: Write the code 1st configure(-) of SecurityConfig class to make LDAP server as Authentication provider.

Step 7: Run the application.

Directory Structure of BootSecurityProj05-Form-LDAP:

- Copy and paste the BootSecurityProj01-Basic-InMemoryDB and rename to BootSecurityProj05-Form-LDAP.
- Add the Spring Security LDAP jar in pom.xml.
- After that change the Web Project Setting context root to BootSecurityProj05-Form-LDAP.
- Then change in <artifactId> & <name> tag of pom.xml.
- Then place the following code with in their respective files.

SecurityConfig.java

```
@Override
public void configure(AuthenticationManagerBuilder auth)
throws Exception {

auth.ldapAuthentication().contextSource().url("ldap://localhost:1038
9/o=sahu")

.managerDn("uid=admin,ou=system").managerPassword("secret")
//For connecting LDAP server

.and().userSearchBase("ou=users").userSearchFilter("(cn={0})") //For
authentication

.groupSearchBase("ou=roles").groupSearchFilter("(uniqueMember={0
})")
.groupRoleAttribute("cn").rolePrefix("ROLE_"); //For
authorization
}
```

Note: Instead of installing LDAP server separately as Apache Directory studio we can also get it as Eclipse plugin [Try this].

Note: To lean Spring Boot Security using JWT, using OAuth 2.0 better to have Spring Boot Rest and Spring Boot MVC also enough.

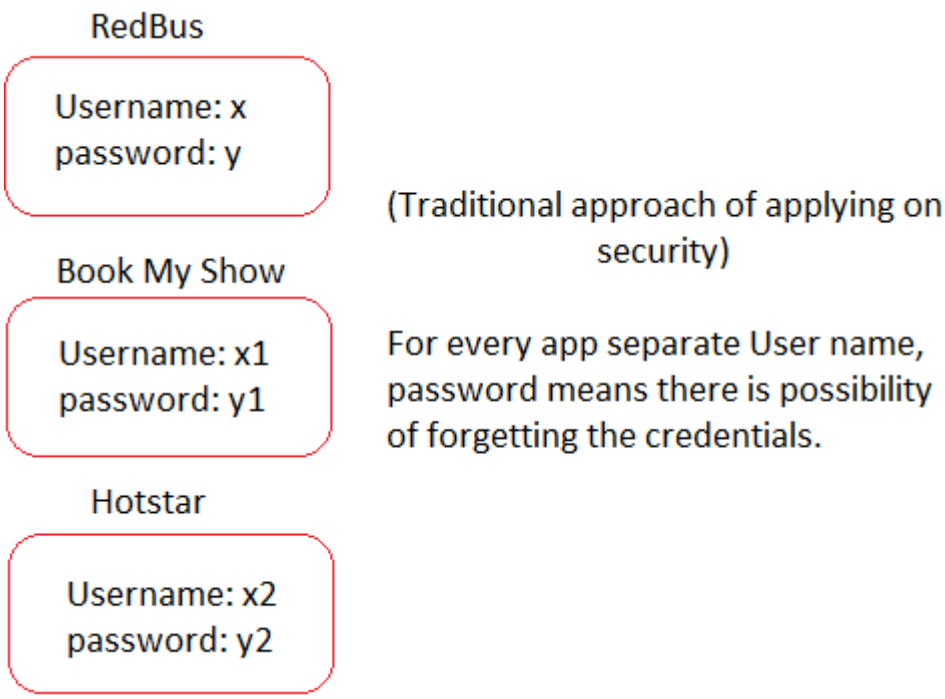
Spring Boot Security with OAuth2.x

- OAuth stands for Open Authorization.
- OAuth 2.x is an open standard and framework for providing 3rd party application services to client apps i.e., security to client apps using third

party applications.

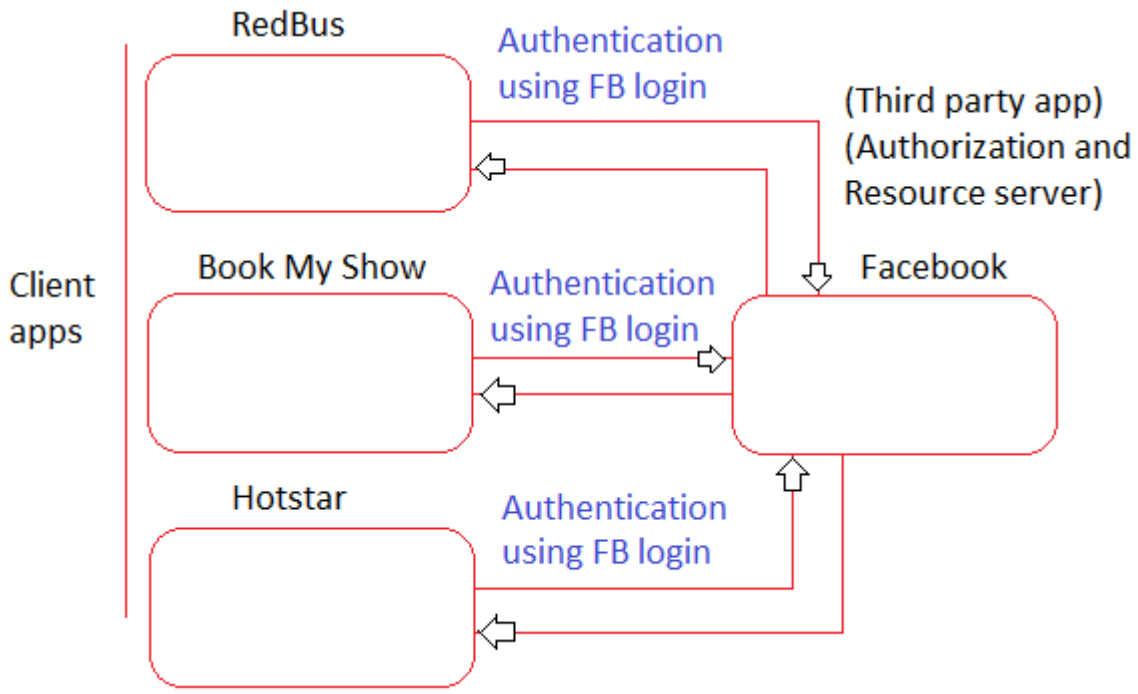
- Our projects or web application behaves like client apps trying to use the services of third-party applications for security.
- Examples for client apps are Swiggy, Zomato, Red Bus, MakeMyTrip, shadi.com, Book My Show, Amazon prim, Netflix, Zerodha, Zepto and etc.
- The third-party applications are technically called Authorization & Resource serves.
- Examples for Authorization and Resource Servers are like Facebook, Gmail, Twitter, Instagram, Linked in, GitHub and etc.
- Different client apps that are listed above tries to use third party applications that are listed above for simplifying Login and authentication activities

Problem:

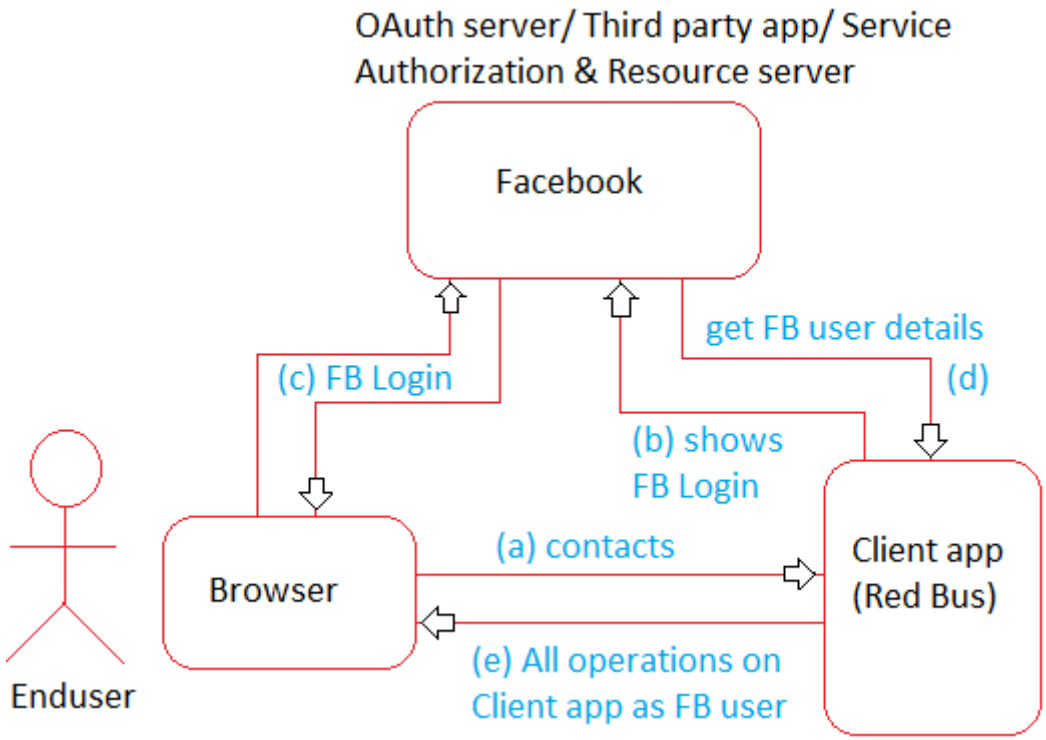


Solution:

- OAuth 2.x supports SSO (Single Sing on Feature) i.e., by login to one third party app like FB, Google and we can login multiple other apps like Red Bus, Zomato, Book My Show, Amazon prim, Netflix, Zerodha, Zepto and etc.
- By login to Gmail account, we can start accessing YouTube, Google Drive, Google Plus and all Google services.



Basic Architecture diagram of OAuth2.x:



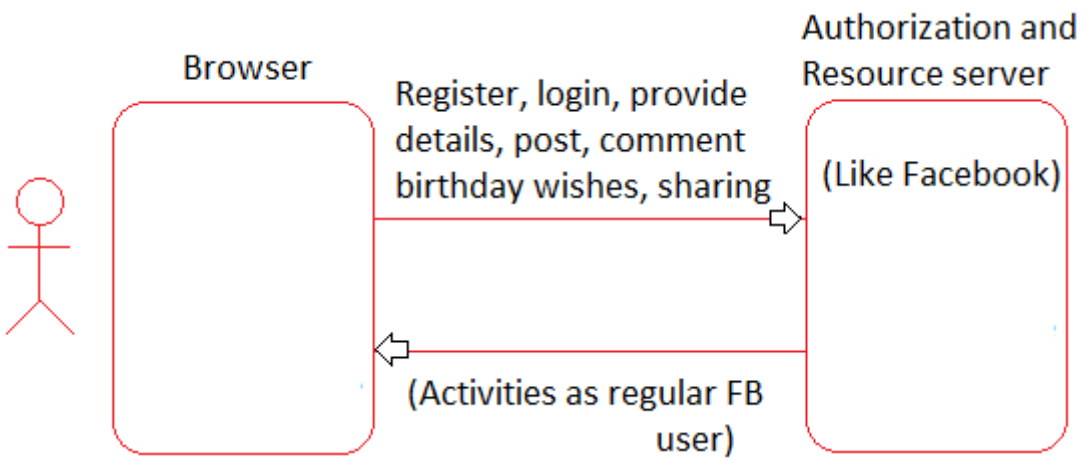
OAuth 2.x features

- It is very popular open standard for securing applications.
- It can be used and implemented in any technical domain like Java, .Net and etc.
- Spring Boot gives lots of abstraction towards using and implementing OAuth 2.x service.

- Very much implemented in day-to-day activity apps like Tick booking apps, E-commerce apps and etc.
- Integration of client app with OAuth 2.x is very easy towards connecting and using Third party service.
- This is not that much recommended to financial services applications like Bank account creation, Credit card issuing and etc. because some third-party services like FB contains lots of fake identities.
- Allows to implement SSO on applications as discussed above.
- If we enable OAuth 2.x based Login activity in our client app then there is no need of implementing LDAP server concepts separately in client app development [If OAuth 2.x is used fully in our client apps development, then there is no need of implementing Spring Security forms, Spring Security LDAP, Spring Security with JWT separately].
- All Third-party apps or services like FB, Gmail provides two ways of interactions,
    a) Direct interaction for end-user
       [To use the services of third-party apps directly as end-user]
       E.g., we using FB directly
    b) Interaction as developer
       [To make other client apps like Red Bus using third party app services]
       E.g., OAuth 2.x implementation

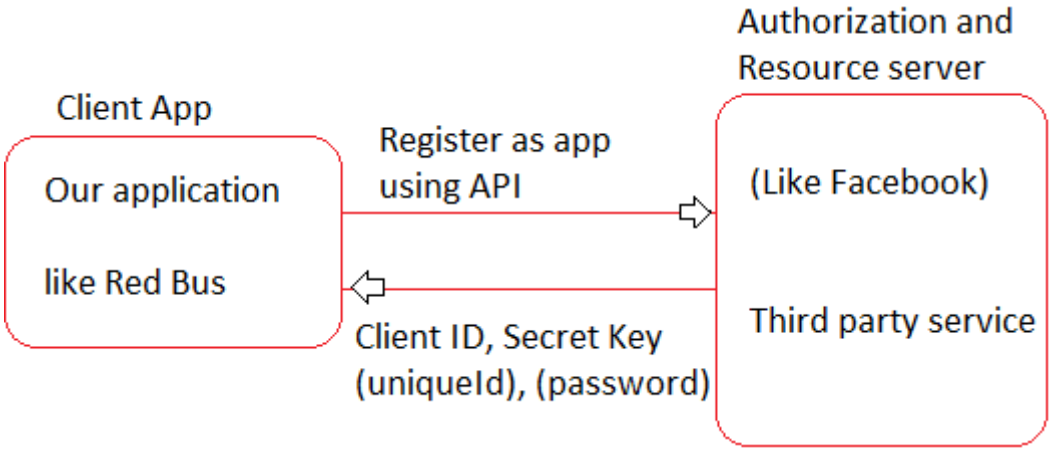
OAuth 2.x Implementation

1. Register and provide user details with Authorization and Resource server.

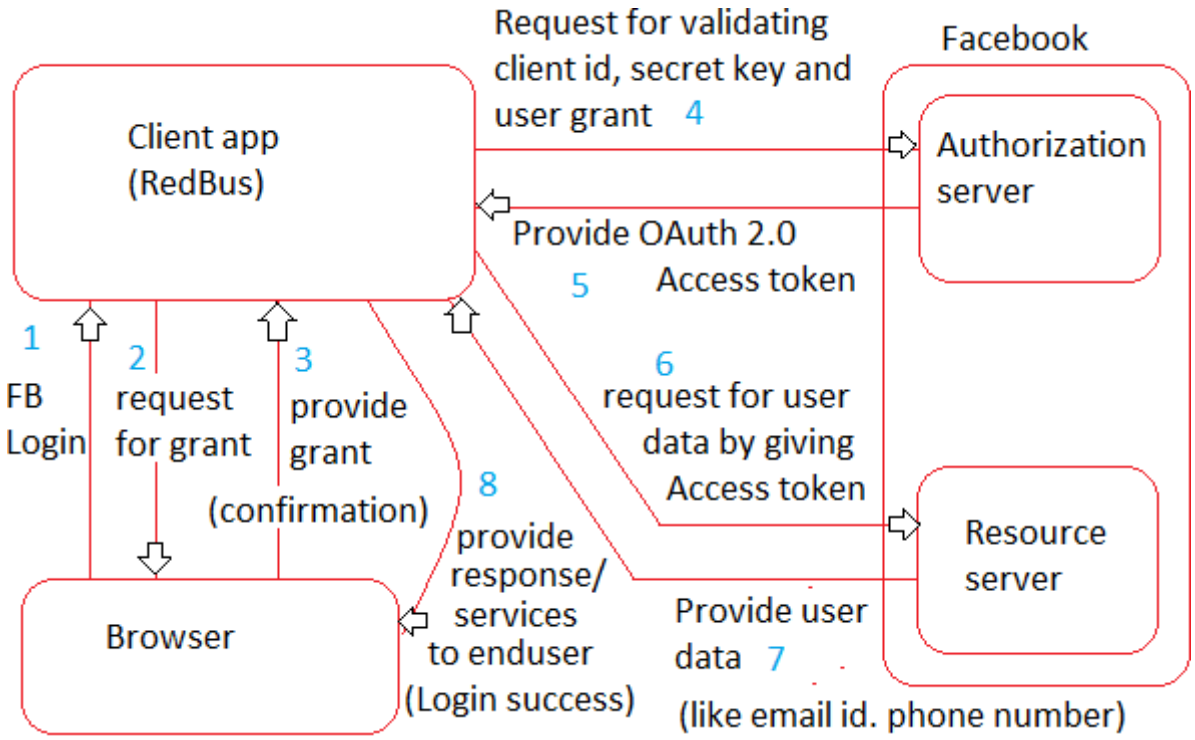


2. Register our client app with Authorization and Resource server (Third party app or service) to get Client ID and Secret Key.

[For this we need to use Third party app or service provided API as developer].



3. Validate end-user and client app in Authorization and Resource server. FB, Gmail, Linked In and etc. maintains two parts.
    a. Authorization Server: Contains authentication logic (FB Login).
    b. Resource Server: Gives access to various details of authenticated user (getting FB user details)



With respect diagram of Step 3:

1. In browser that is pointing client app screen (Red Bus screen) end user clicks on FB Login.
2. The client app (Red bus) makes you provide login credentials of FB to

provide permission to use FB user details (request for access grant).

3. Once we complete FB Login and "continue as certain <user>" we can say Access Granted (Permission provided).
4. Client app (Red Bus) goes to Authorization server of FB for client app user details validation by carrying client id, secret key, login details.
5. FB Authorization server validates the client app and user provides one Access token (ID) which is valid for current user and current client app to perform certain operations in the Resource server (if end-user tries to change it then it will not work).
6. Client app (Red bus) makes a request to Resource server of FB having that Access token (ID).
7. Resource server of FB validates the access token and provides the required and permitted user data to client app.
8. Finally, client app gets user data and uses that data to provides various services to End user.

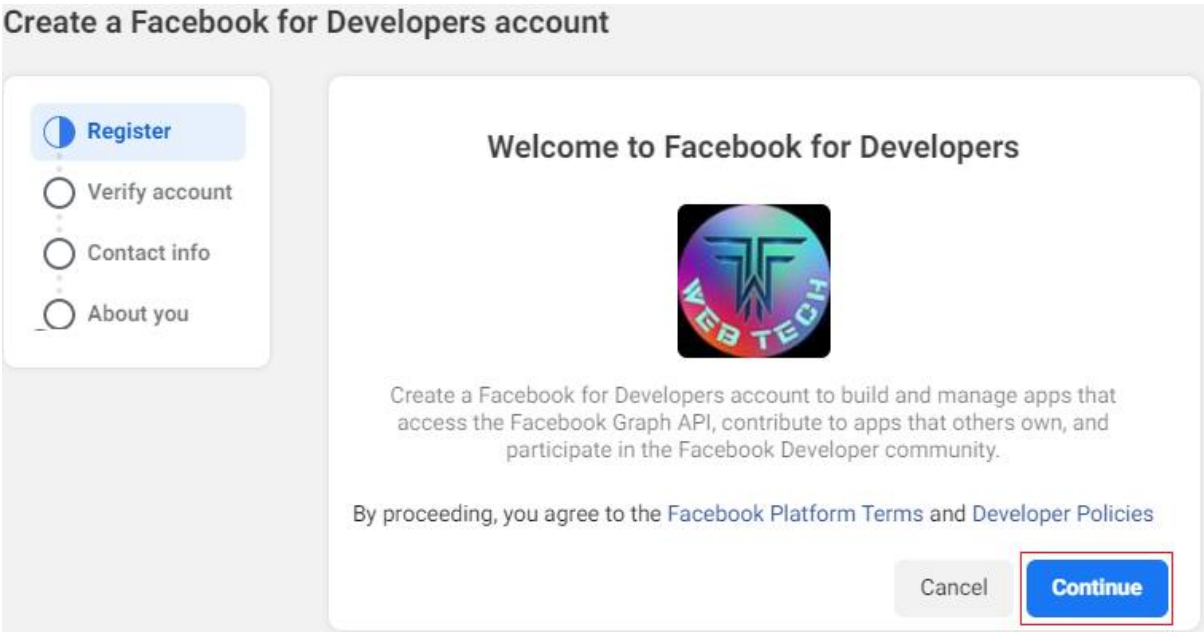
Facebook Developer account creation Process

Registering our web application (client app) with FB to get Client Id and Secret Key

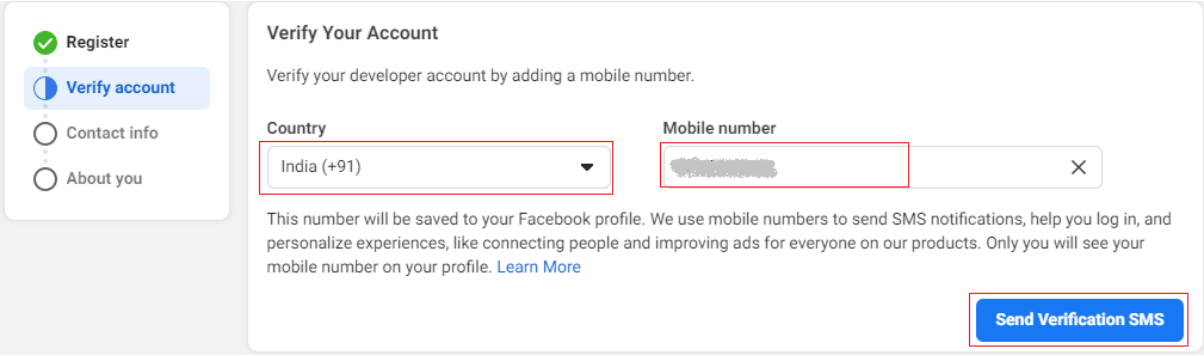
Step 1: Go to FB developers Meta for Developers

Step 2: Create a Facebook for Developers account for that click ok Get Started and follow the bellow steps.

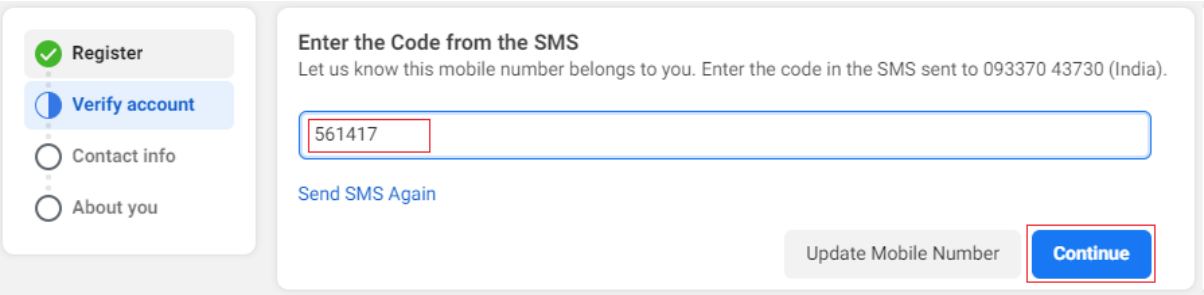
a. Click on Continue button.



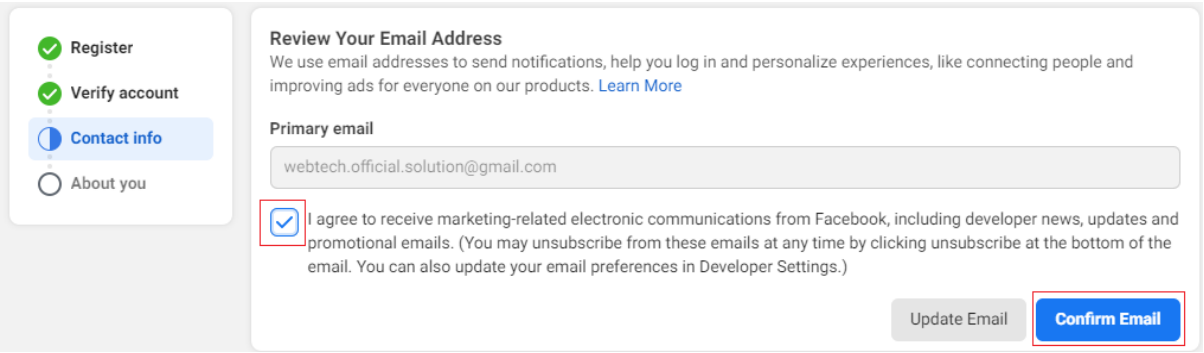
b. Choose Country, give the Mobile number then click on send Verification Code button.



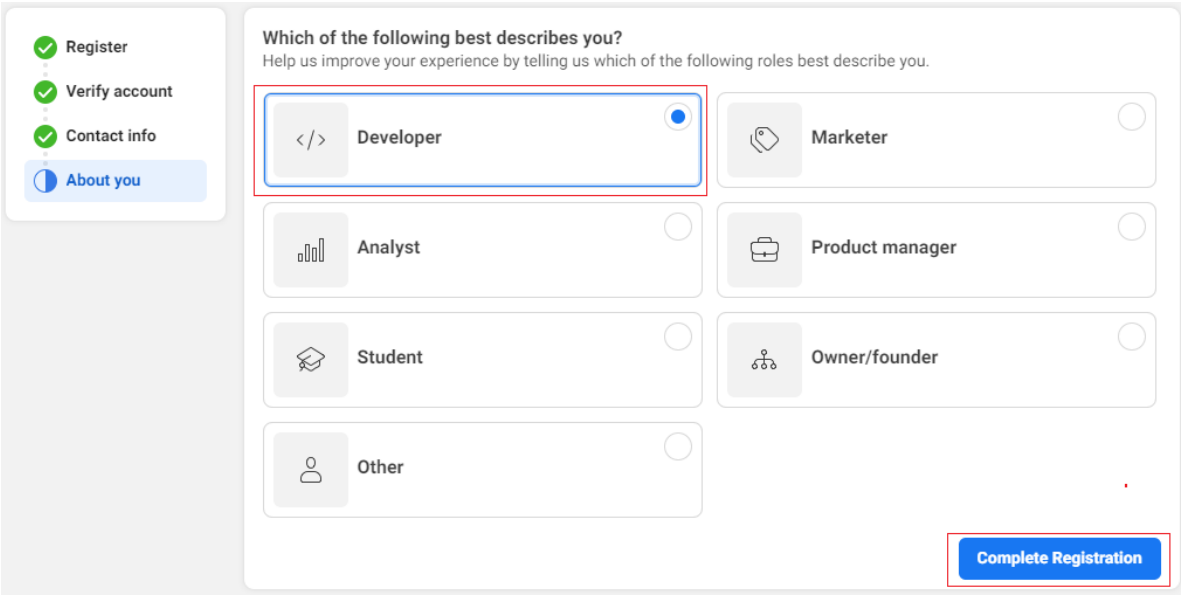
c. Fille the verification code that you received to your Mobile number the click on Continue button.



d. Click ok I agree check box then click on Confirm Email button.

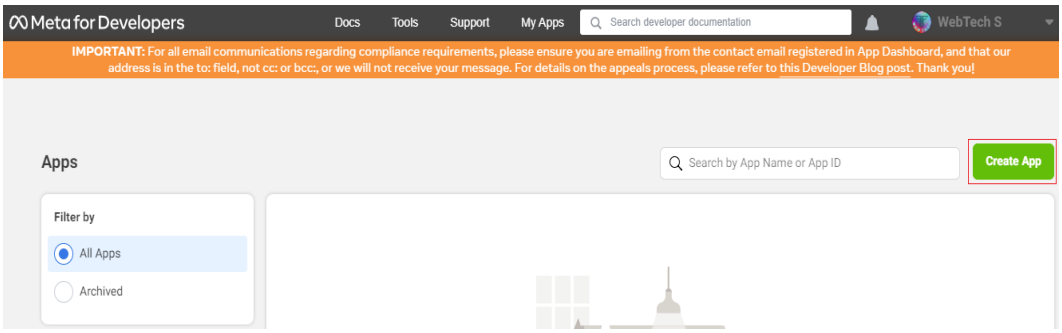


e. Choose from the option which of the following best describes you? i.e., if you are a developer, then click on developer, if you are a student then click on student, according to your profession choose any one the option.
Then click on Complete Registration button.

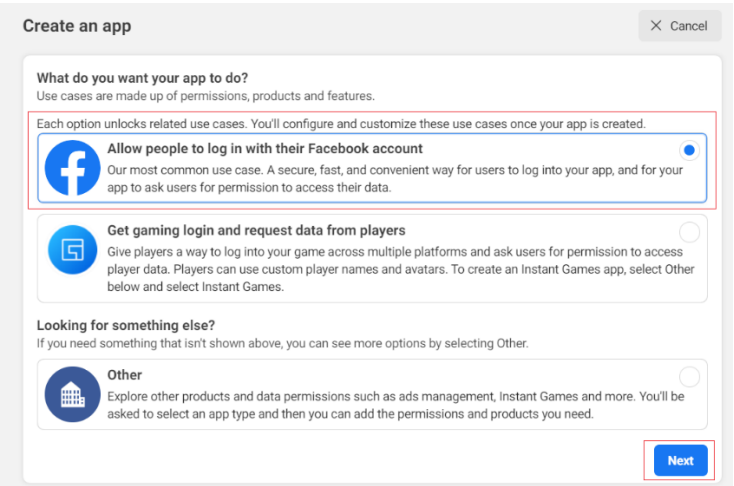


Step 3: Create FB application follow the below steps

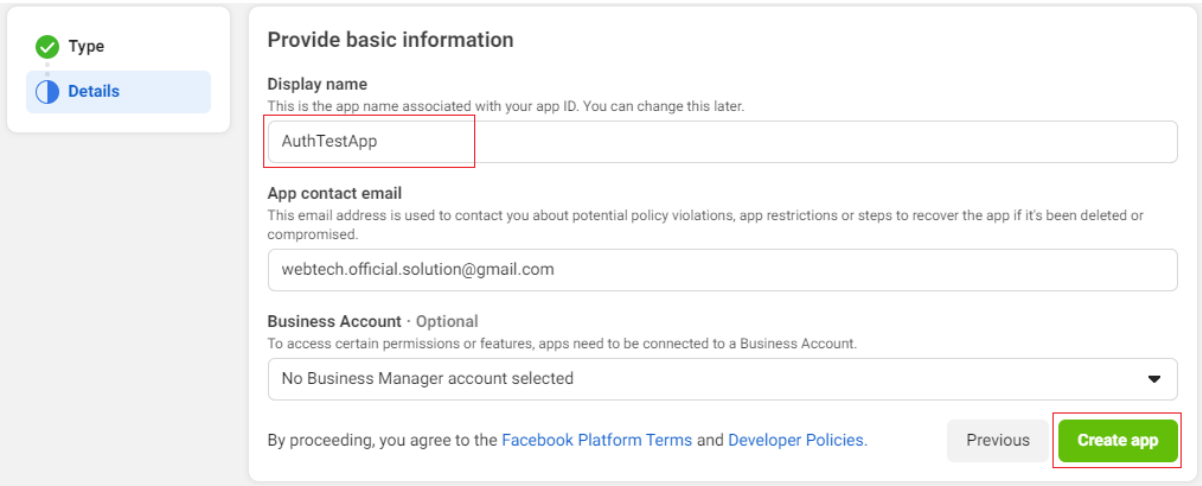
a. Click on Create Application button.
   For second time FB Developers home page My Apps then click on Create App



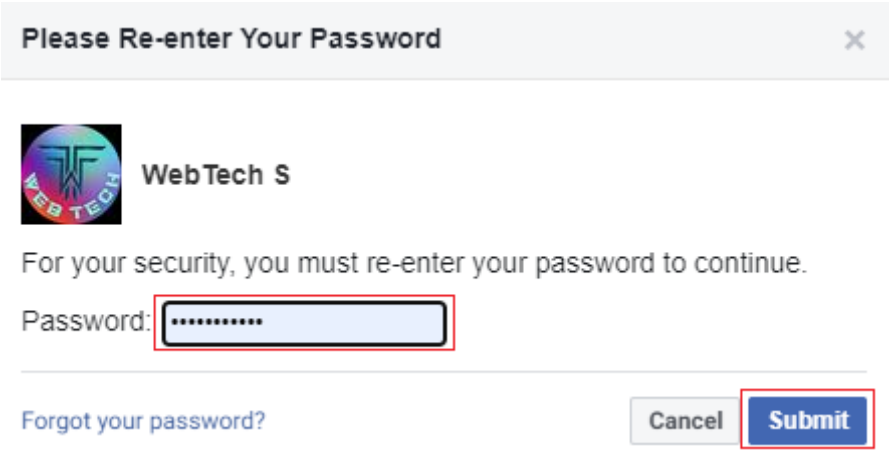
b. Choose "Allow people to log in with their Facebook account" option then click on Next button.



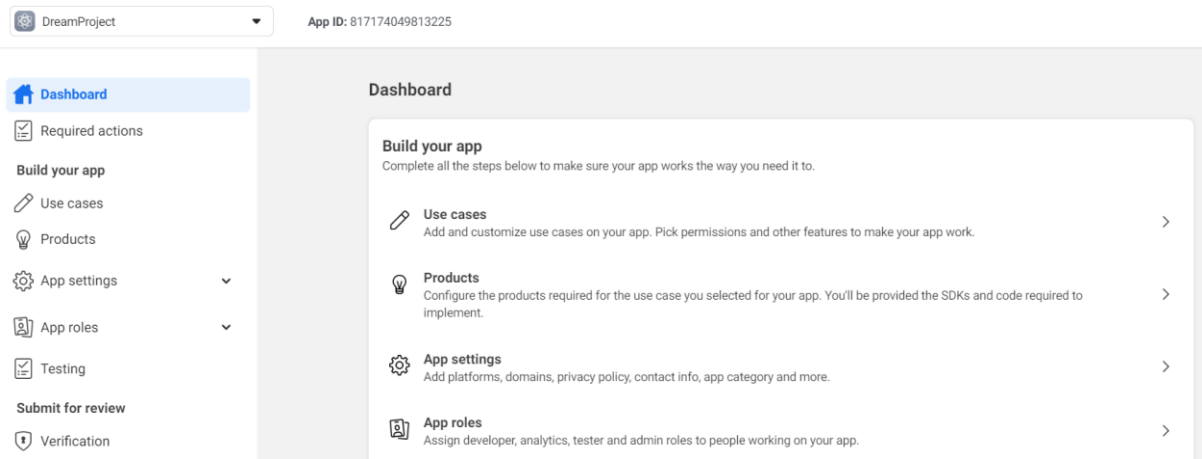
c. Provide basic information like Display name then click on Create App button.



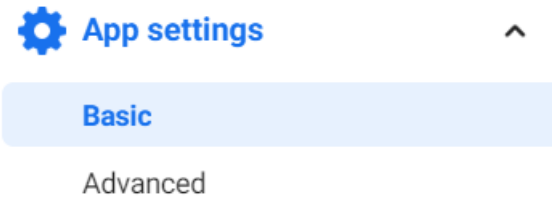
d. Then it will open a popup give your FB Password then click on Submit.



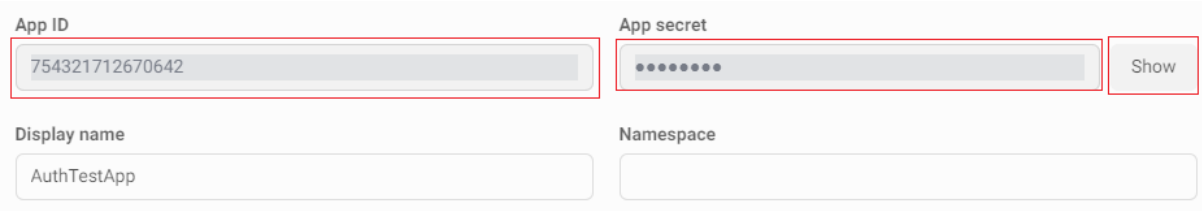
e. This is the Dashboard of our application that we created.



Step 4: To gather App ID and App Secret of the App go to App Settings then click on Basic option.



Now you can see the App ID and App secret if you click on show button then the App secret will also show.



Developing single Sign in application using Facebook

Step 1: Make sure that you have FB developer account and you logged into that account.

Step 2: Collect App ID and App secret from Facebook developer account app

App ID: 754321712670642

App secret: 5934d3f9441cedfe4sdfsdb336cfb13

Step 3: Develop Spring Boot Rest/ Spring Boot MVC application as client app having Spring Security OAuth Support.

a. Create Spring Boot starter project having the following starters

X Spring Security
X OAuth2 Client
X Thymeleaf
X Spring Web

b. Add the following entries in application.properties file

```
#Application name
spring.application.name=SpringSecurityOAuth2.xApp
#Server Port number
server.port=4041

#Cofigure FB Auth server credentials
```

spring.security.oauth2.client.registration.facebook.client-id=754321712670642
spring.security.oauth2.client.registration.facebook.client-secret=5934d3f9441cedfe4sdfsdb336cfb13
spring.security.oauth2.client.registration.facebook.scope=email, public_profile

c. Develop Security configuration class.

d. Develop Rest controller/ MVC controller component.

e. Add login.html having OAuth Facebook hyperlink.

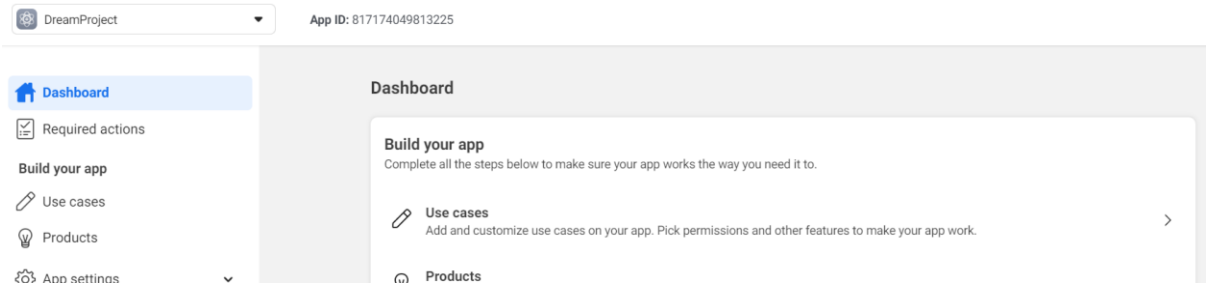
Step 4: Run the application as Spring Boot application.

Note:

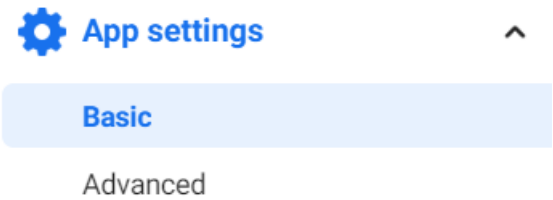
- ✓ Principal object holds currently logged in user name.
- ✓ Authentication object holds multiple details currently logged in user like credentials (username, password), additional details and etc.

Before run we have to do Some setup on the Facebook app

Step 1: Go to your app in Facebook Developer account.

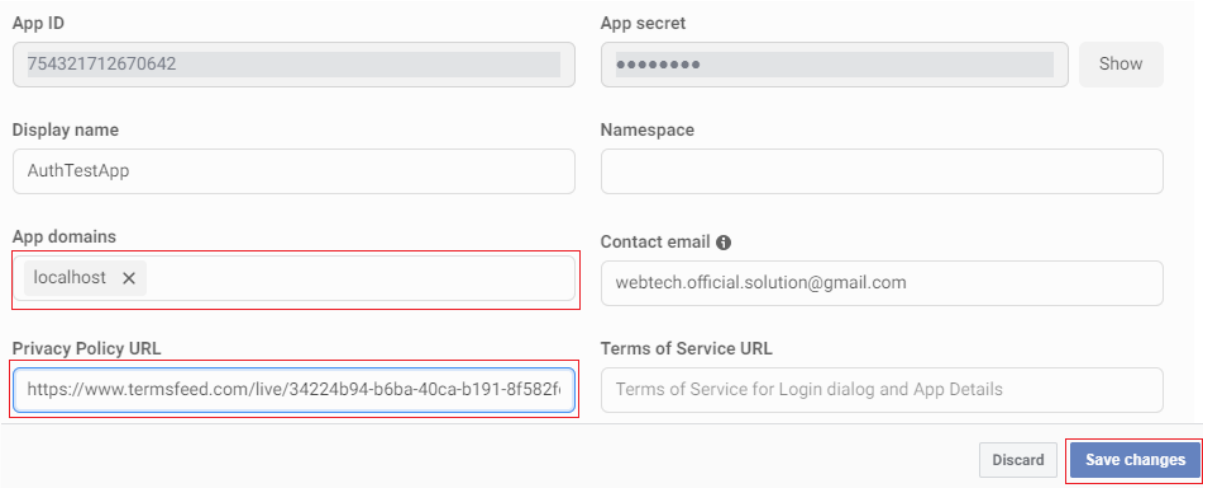


Step 2: Then go to Settings, Basic section.



Step 3: Fill the App domains a “localhost” and Privacy Policy URL then click on

Save changes button.



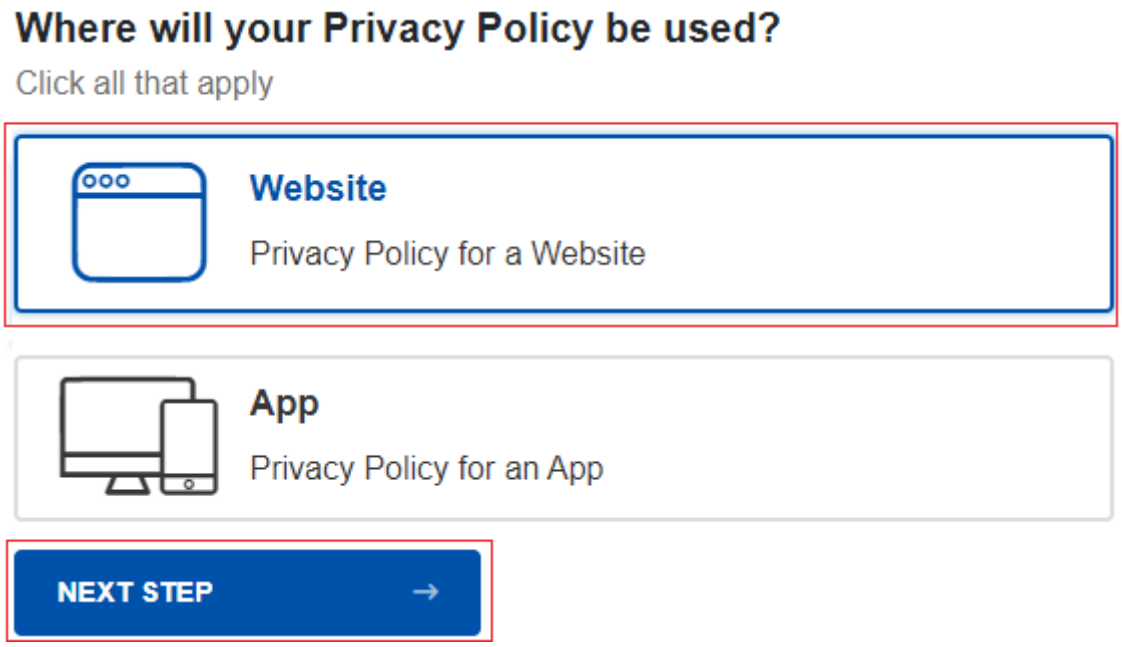
Note:

- ✓ If currently developing an app for Facebook, you may be required to enter the URL of your Privacy Policy at the "**Privacy Policy URL**" field because they make is mandatory. The reason behind that is a Privacy Policy agreement is required by law if you collect personal information (email address, name, photo, and so on) from users.

To generate Free Privacy Policy URL

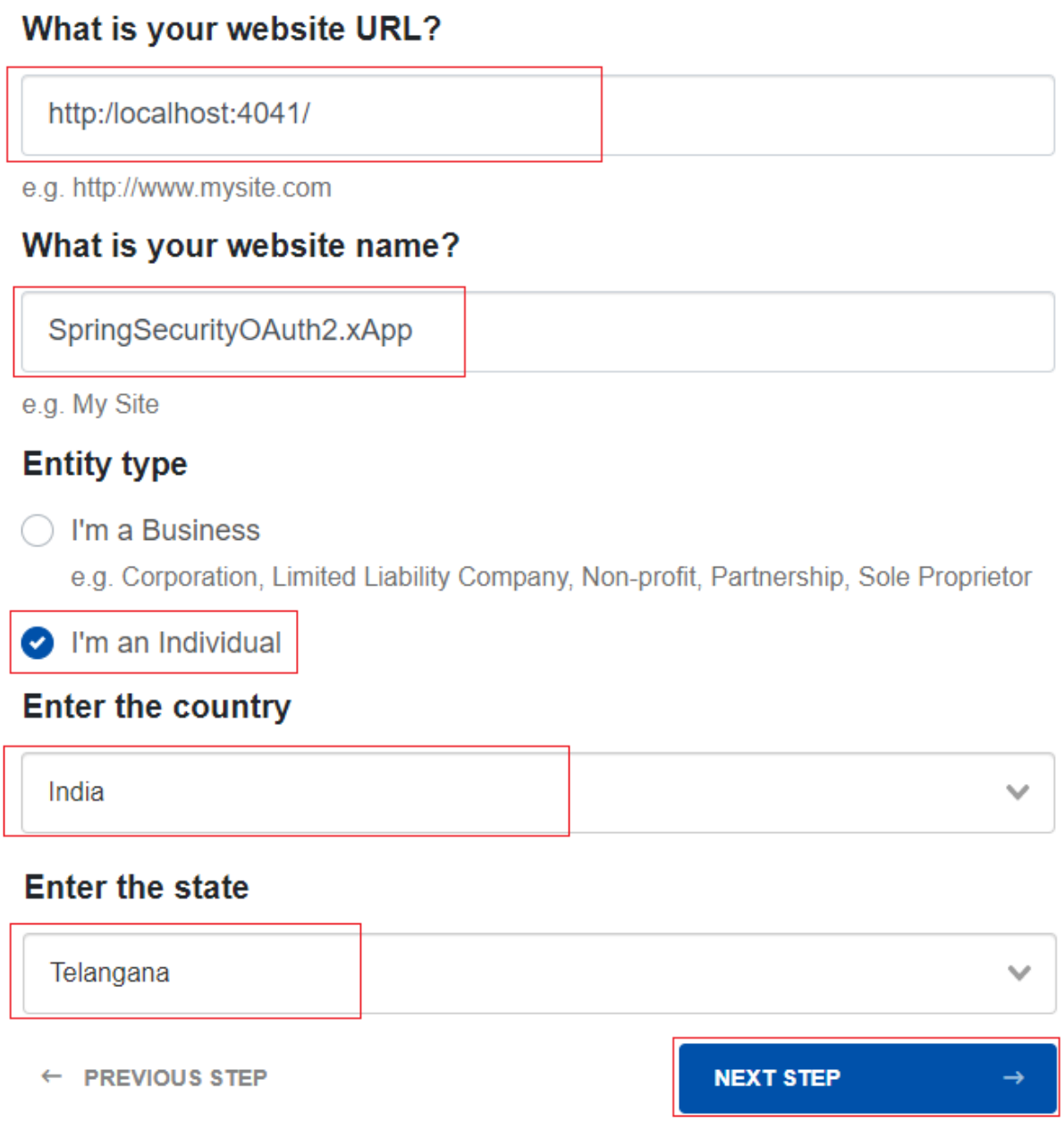
Step 1: Go to [Privacy Policy Generator]

Step 2: Choose Where will you Privacy Policy be use you can choose both and then click on NEXT STEP button.



Step 3: Fille the required information as like below then click on NEXT STEP button.

What is your website URL? – http://localhost:4041/ (your URL)
What is your website name? – SpringSecurityOAuth2.xApp (any name)
Entity type – I'm an Individual (If you have a business then choose, I'm a Business)
Enter the country – India (Your country)
Enter the state – Telangana (Your state)



Step 4: Choose What kind of personal information do you collect form users then click on NEXT STEP button.

What kind of personal information do you collect from users?

Click all that apply

- [x] Email address
- [x] First name and last name
- [x] Phone number
- [x] Address, State, Province, ZIP/Postal code, City
- [ ] Social Media Profile information (ie. from Connect with Facebook, Sign In With Twitter)
- [ ] Others

← PREVIOUS STEP NEXT STEP →

Step 5: How can users contact you for any questions regarding your Privacy Policy? choose By email option and give your email id then click on NEXT STEP button.

How can users contact you for any questions regarding your Privacy Policy?

Click all that apply

- [x] By email

What's the email?

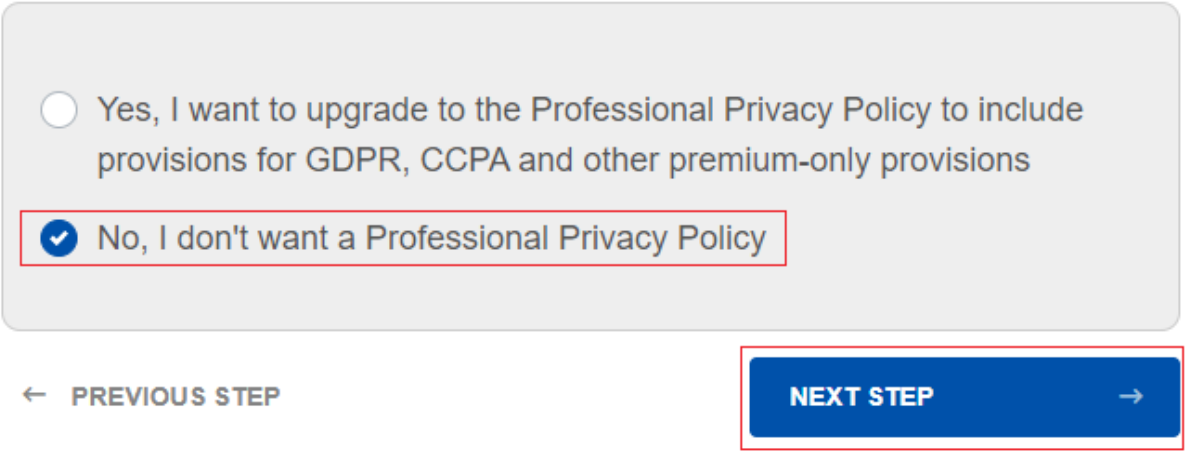
webtech.official.solution@gmail.com

e.g. office@mycompany.com

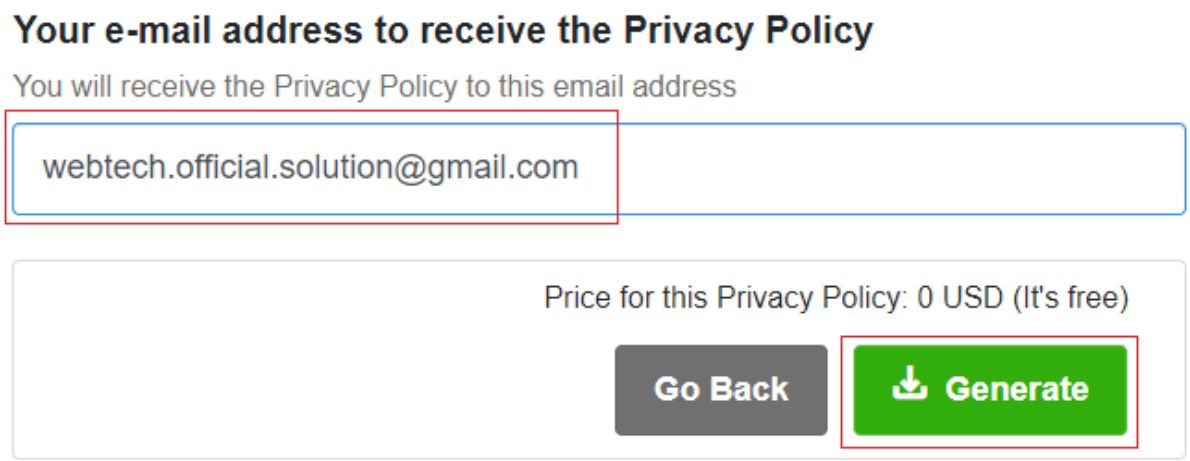
- [ ] By visiting a page on our website
- [ ] By phone number
- [ ] By sending post mail

← PREVIOUS STEP NEXT STEP →

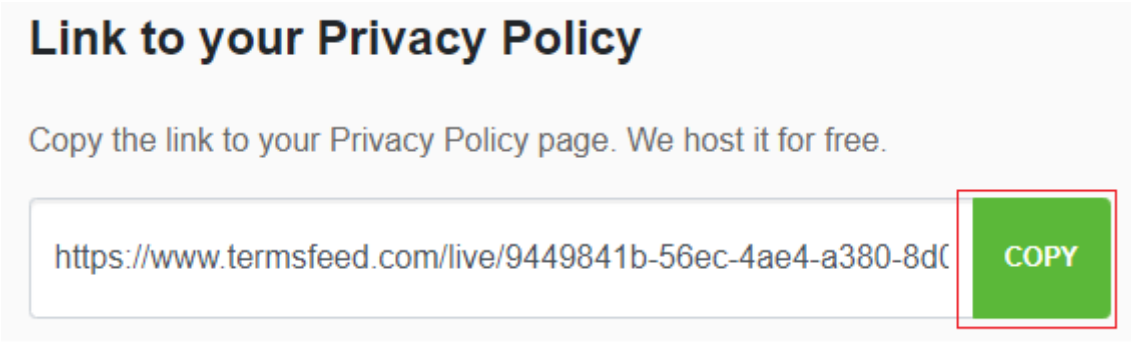
Step 6: Scroll down and choose No, I don't want a Professional Privacy Policy then click on NEXT STEP button.



Step 7: Give you email id then, click on Generate button.



Step 8: Click on COPY button to copy the Privacy policy URL. Now you can use this URL in you Facebook developer app.



Now we can run our application. [Link will help in future]

Directory Structure of BootSecurityProj06-OAuth2.xApp:

```
BootSecurityProj06-OAuth2.xApp [boot]
  Deployment Descriptor: BootSecurityProj06-OAuth2.xApp
  JAX-WS Web Services
  src/main/java
    com.sahu
      Application.java
      ServletInitializer.java
    com.sahu.config
      SecurityConfig.java
    com.sahu.controller
      RedBusUserOperationController.java
  src/main/resources
      static
    templates
        login.html
      application.properties
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  Deployed Resources
  src
  target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

- Develop the above directory structure using Spring Starter Project option packaging as war and create the package, classes, and HTML files.
- Use the following starters during project creation.

  X Spring Security
  X OAuth2 Client
  X Thymeleaf
  X Spring Web

- Then place the following code with in their respective files.

application.properties

```
#Application name
spring.application.name=SpringSecurityOAuth2.xApp

#Server Port number
server.port=4041

#Cofigure FB Auth server credentials
spring.security.oauth2.client.registration.facebook.client-
id=427670002236454
```

```
spring.security.oauth2.client.registration.facebook.client-secret=305289f51fa64f42a3a372180a1b0314

spring.security.oauth2.client.registration.facebook.scope=email, public_profile
```

SecurityConfig.java

```
package com.sahu.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter  {

	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.authorizeHttpRequests().antMatchers("/", "/login", "/home").permitAll()
		.anyRequest().authenticated()
		.and().formLogin()
		.and().oauth2Login()
		.and().csrf().disable();
	}

}
```

RedBusUserOperationController.java

```
package com.sahu.controller;

import java.security.Principal;
```

```
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class RedBusUserOperationController {

	@GetMapping("/home")
	public String showHome() {
		return "Hello, Welcome to home page RedBus.com";
	}

	@GetMapping("/after")
	public String afterLoginPage() {
		return "Hello, Successfully logged into Redbus.com";
	}

	@GetMapping("/user")
	public Authentication showUserDetails(Principal principal) {
		System.out.println("Logged in details: "+principal.getName());
		Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
		return auth;
	}

}
```

login.html

```
<html xmlns:th="http://www.thymeleaf.org">

<h1 style="color: blue; text-align: center;">Login page</h1>

<form th:action="@{/login}" method="POST">
	<table border="0" bgcolor="cyan" align="center">
		<tr>
			<td>User Name:</td>
			<td><input type="text" name="username"/></td>
		<tr>
		<tr>
```

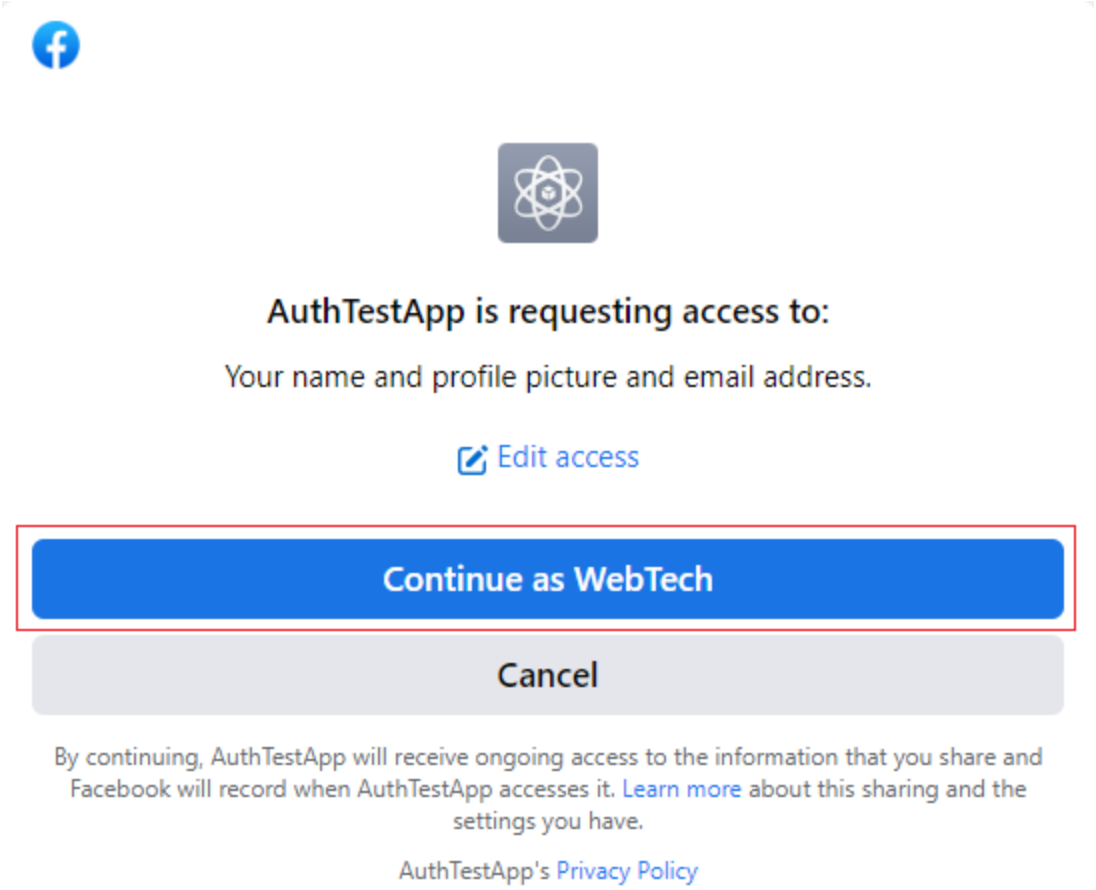
```
                <td>Password:</td>
                <td><input type="password" name="password"/></td>
            <tr>
            <tr>
                <td colspan="2" align="center">
                    <input type="submit" value="login"/>
                </td>
            <tr>
        </table>
        <span th:if="${param.error}">Invalid Login details (Authentication
failed)</span>
        <span th:if="${param.logout}">User Logout successfully</span>
</form>
<br>
<h1>
        <a href="/oauth2/authorization/facebook">Facebook Login</a>
</h1>
```

Follow the below steps for execution

Step 1: Go to browser and type http://localhost:4041/home/ you will get the output.

Step 2: Change the URL to http://localhost:4041/after then it you go to login page http://localhost:4041/login then click on Facebook link

Step 3: Now click on Continue as WebTech.



Step 4: Now you will get http://localhost:4041/#_=_ this UAL in address bar, so change it to http://localhost:4041/after then you will get the output and now you can use http://localhost:4041/user also.

Q. Why Spring Security deprecates the use of WebSeurityConfigurerAdapter class and how we can fix?

Ans. The main reason behind that, the developer of the Spring framework encourages users to move toward a component-based security configuration.

Previously we use,

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //Configure HttpSecurity
    }
```

```
        @Override
        public void configure(WebSecurity web) throws Exception {
                //Configure WebSecurity
        }

}
```

This is fine up to Spring Security (version) 5.6.5 or older and with Spring Boot (version) 2.6.8 or older. But if your project uses Spring Security 5.7.1 or newer and Spring Boot 2.7.0 or newer then you will get a deprecated warning in your IDE like "*The type WebSeurityConfigurerAdapter is deprecated*".

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

The type WebSecurityConfigurerAdapter is deprecated

So, instead of extending from WebSecurityConfigurerAdapter class and overriding methods configure(HttpSecurity HTTP) and configure(WebSecurity web) they encourage to declare two beans of type SecurityFilterChain and WebSecurityCustomizer as follows,

```
@Configuration
public class SecurityConfig {

        @Bean
        public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {

        }

        @Bean
        public WebSecurityCustomizer webSecurityCustomizer() {

        }

}
```

Official document Link: [Spring Security]

Change the code in SecurityConfig.java

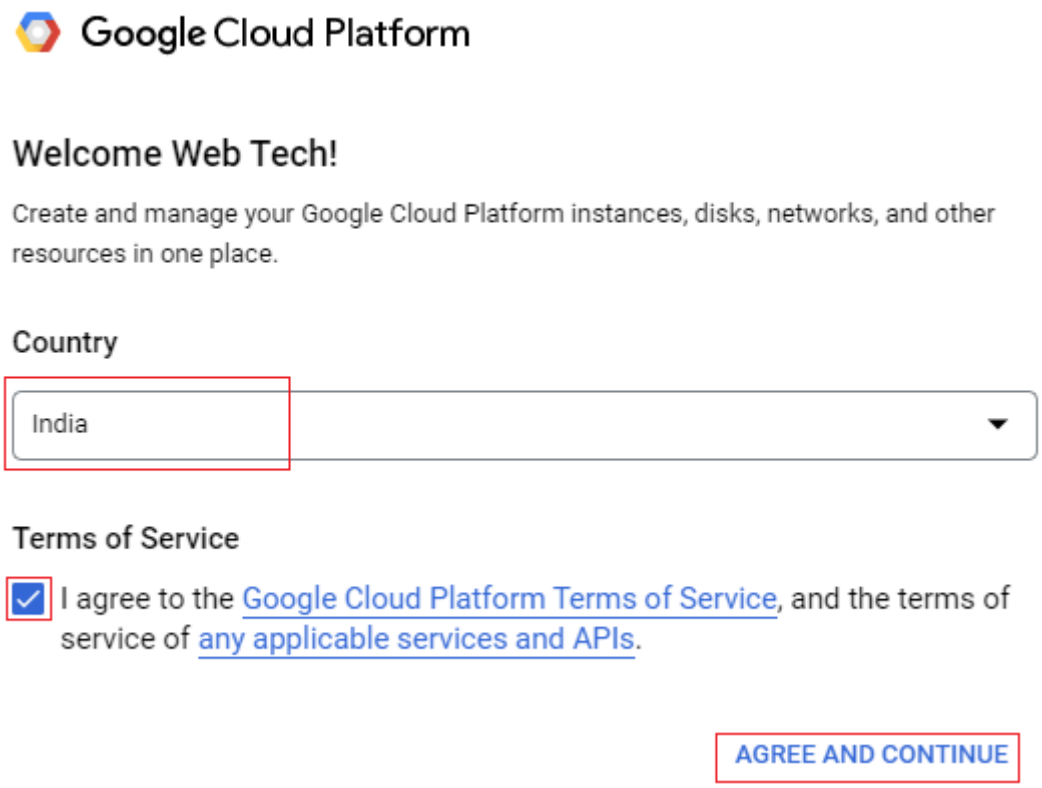
SecurityConfig.java

```
package com.sahu.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
//@EnableWebSecurity
public class SecurityConfig {

	/*@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.authorizeHttpRequests().antMatchers("/", "/login",
"/home").permitAll()
		.anyRequest().authenticated()
		.and().formLogin()
		.and().oauth2Login()
		.and().csrf().disable();
	}*/

	@Bean
	public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
		http.authorizeHttpRequests().antMatchers("/",
"/login").permitAll()
		.anyRequest().authenticated()
		.and().formLogin()
		.and().oauth2Login()
		.and().csrf().disable();

		 return http.build();
	}

}
```

Developing Single Sign in application using Google

Step 1: To register client app with Google Developer, go to the Google Cloud Platform

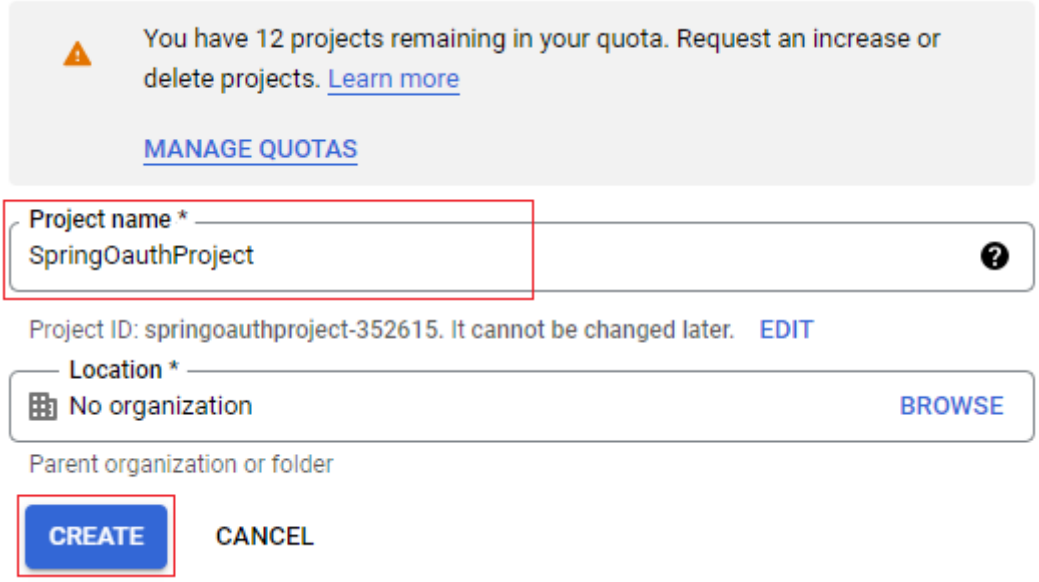
Step 2: If you go for the first time then choose your Country and check Terms of Service then click on AGREE AND CONTINUE.



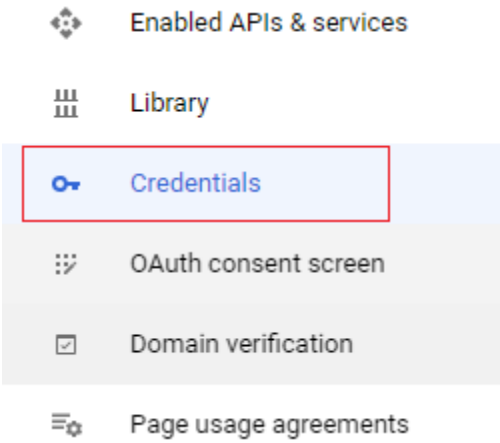
Step 3: To create a new project, you can get to Select a project option

Then click on NEW PROJECT

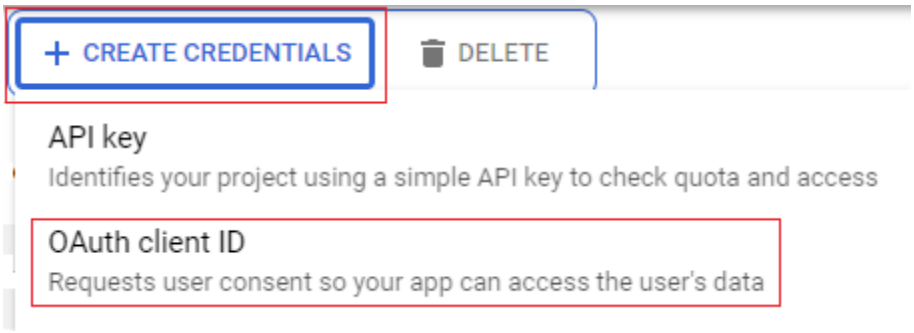
Step 4: Give the Project name the click on CREATE button.
You can see a warning that we can create 12 projects in your quota.



Step 5: After the project created, this will select as recent project then click on the Credentials tab.



Step 6: Click on CREATE CREDENTIALS and then OAuth client ID.



Step 7: Then click on CONFIGURE CONSENT SCREEN button.

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See Setting up OAuth 2.0 for more information. Learn more about OAuth client types.

To create an OAuth client ID, you must first configure your consent screen CONFIGURE CONSENT SCREEN

Step 8: Choose User Type as External then click on CREATE button.

Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.

User Type

Internal

Only available to users within your organization. You will not need to submit your app for verification. Learn more about user type

External

Available to any test user with a Google Account. Your app will start in testing mode and will only be available to users you add to the list of test users. Once your app is ready to push to production, you may need to verify your app. Learn more about user type

CREATE

Step 9: Give App information like App name and User support email,

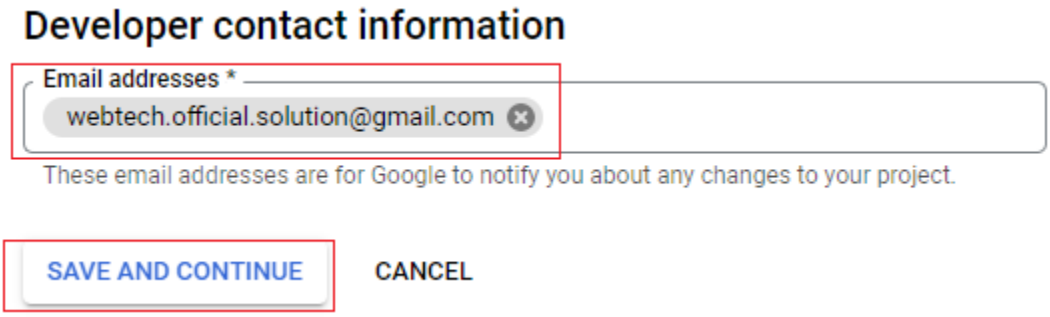
App information

This shows in the consent screen, and helps end users know who you are and contact you

App name *
GCApp1
The name of the app asking for consent

User support email *
webtech.official.solution@gmail.com
For users to contact you with questions about their consent

Then scroll down and give Developer contact information like email address after that click on SAVE AND CONTINUE button.

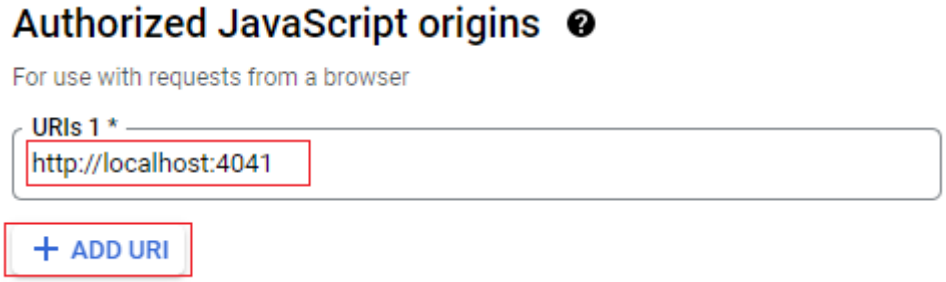


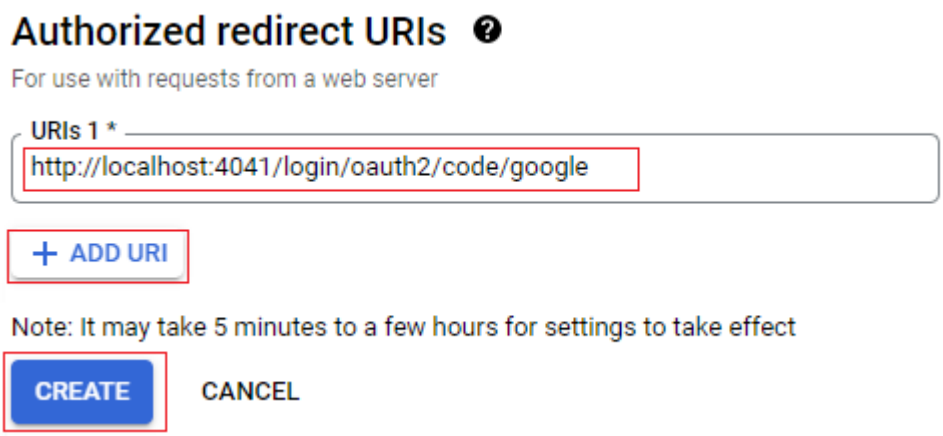
Step 10: After that click SAVE AND CONTINUE button two times then click on the BACK TO DASHBOARD button.

Step 11: Follow Step 5 & 6 and now choose the Application type as Web application, then a lot of options will enable.

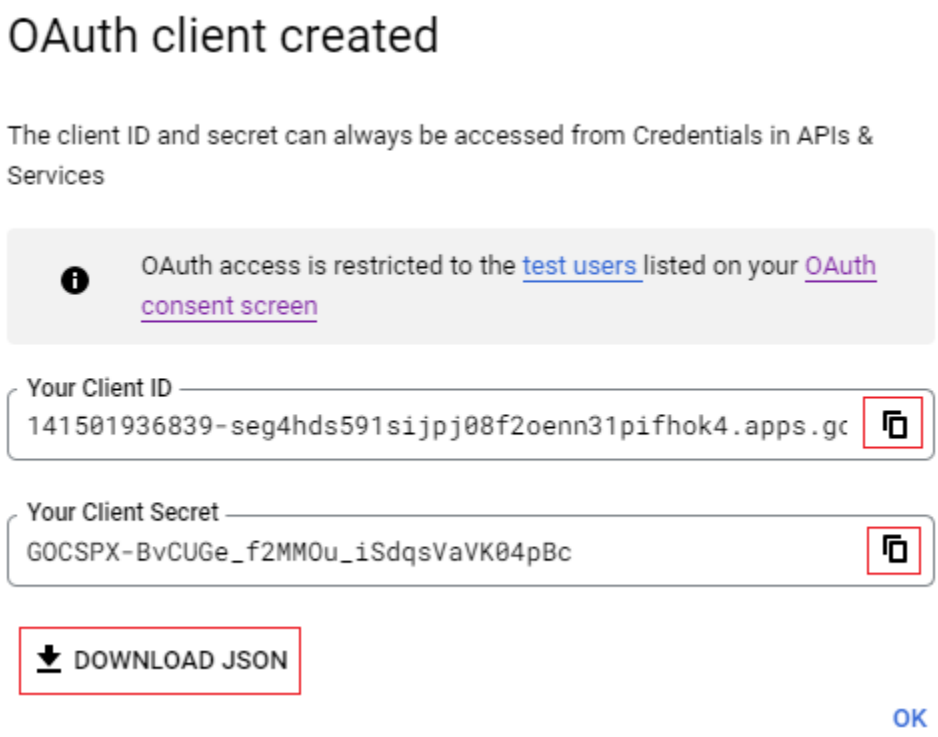
Step 12: Go to Authorized JavaScript origins section and click on ADD URL button and give URL like http://localhost:<port_number> and after that go to Authorized redirect URLs and click on ADD URL button then give the URL like http://localhost:<port_number>/login/oauth2/code/google then click on CREATE button

Note: After port number "login/oauth2/code/google" is fixed for redirect URL.

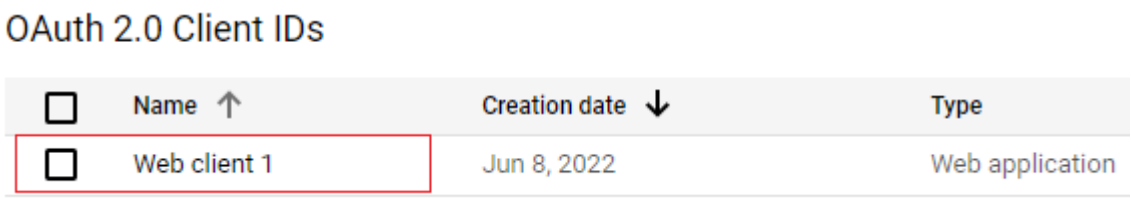




Step 13: Now you will get a popup like below from there you can copy Your Client ID and Your Client Secret and as well as you can download as JSON file by click on DOWNLOAD JSON button.



Step 14: Otherwise, you can collect later by going inside of your OAuth 2.0 Client IDs (Web client 1)



Do the following changes in your application:
Step 1: Add the google related entries in application.properties file.

Step 2: Add the google relate hyperlink in login.html file.

application.properties

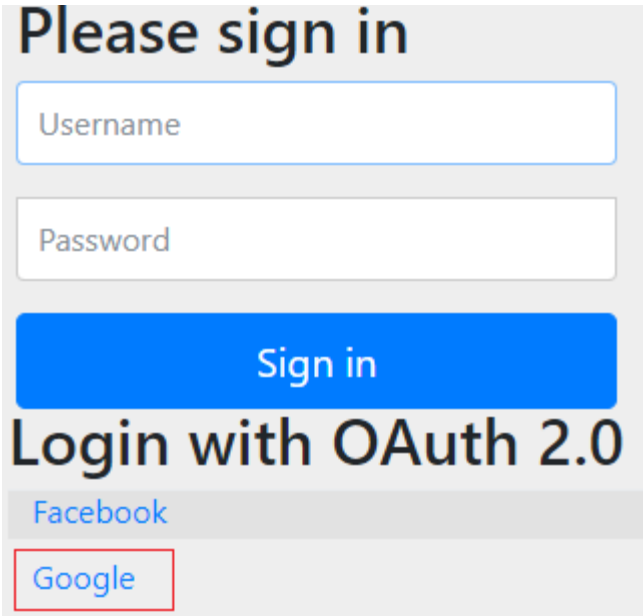
```
#Cofigure Google OAuth 2.0 Client ID credentials
spring.security.oauth2.client.registration.google.client-id=141501936839-seg4hds591sijpj08f2oenn31pifhok4.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=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
spring.security.oauth2.client.registration.google.scope=email, profile
```

application.properties

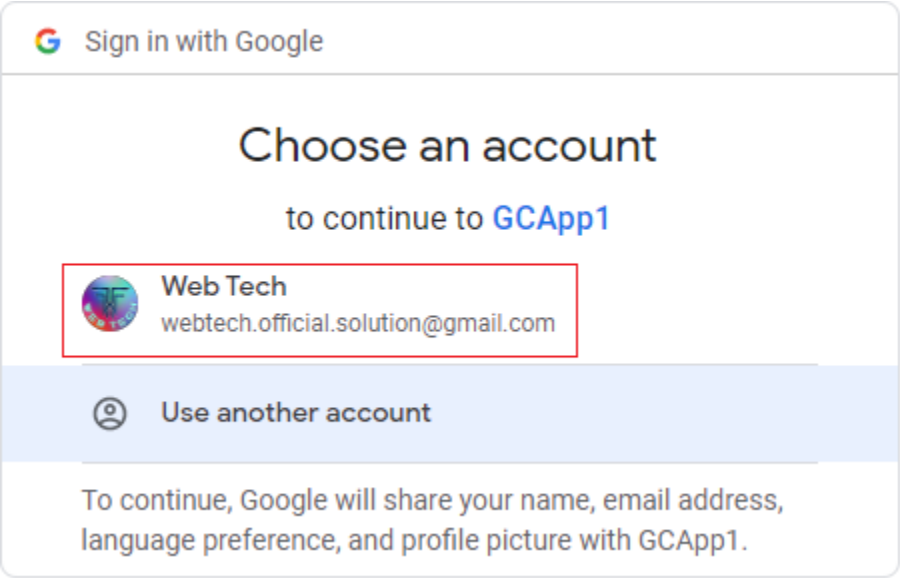
```
<br>
<h1>
        <a href="/oauth2/authorization/google">Google Login</a>
</h1>
```

Follow the below steps for execution
Step 1: Go to browser and type http://localhost:4041/home/ you will get the output.

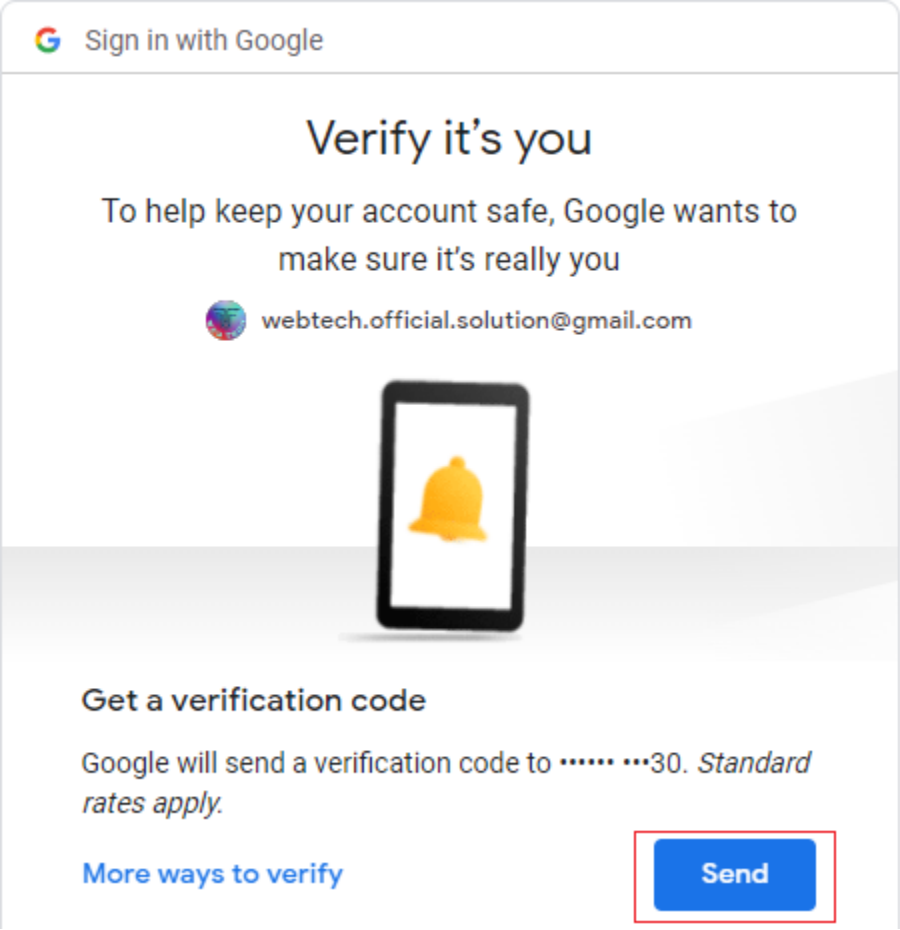
Step 2: Change the URL to http://localhost:4041/after then it you go to login page http://localhost:4041/login then click on google link



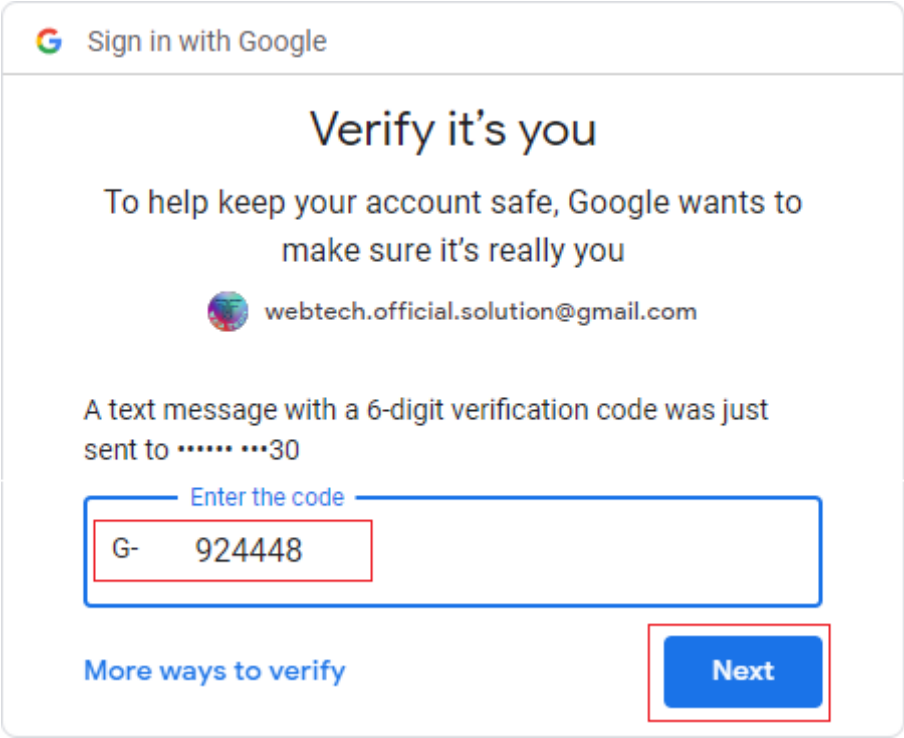
Step 3: Now choose your account.



Step 4: Then it will ask to Verify its's you, click on Send button then you will get a google verification code to your registered mobile number



Step 5: Enter the received google verification code then click on Next button.



Step 6: Now you will get the output and now you can use http://localhost:4041/user also.

---------- The END ----------