

Spring Security using spring boot , spring JPA , MySQLI and Thymeleaf

Tutorial

Introduced by

Elwalied AbdElrahim Elnour

Senior Java web Developer

Email : Wad3935@gmail.com

Git: <https://github.com/waliedAfro>

project name: Customers registration system

what we learn in this tutorial:

1. Learn how to create and build a spring MVC web application using spring boot.
2. Learn how to configure MySQL Database in spring boot.
3. Learn how to create JPA entities.
4. Learn how to configure spring security in spring boot.
5. Learn how to implement spring security with backend database.
6. Learn How to develop end to end Customer Registration implementation.
7. Learn how to develop custom login.
8. Learn how to integrate spring security in Thymeleaf and how to display security details.
9. Learn how to implement spring security logout.

Project requirement:

Create web application for Customers registration system

Functionality:

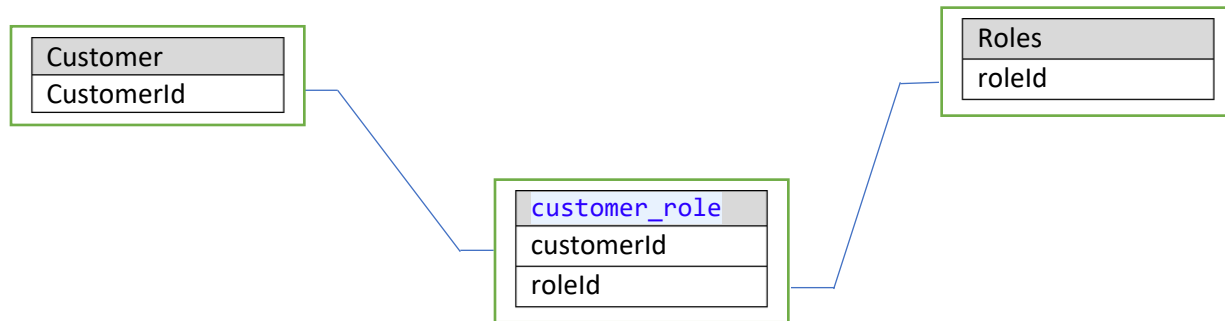
- 1- Customer registration page.
- 2- Login page.

Required database tables:

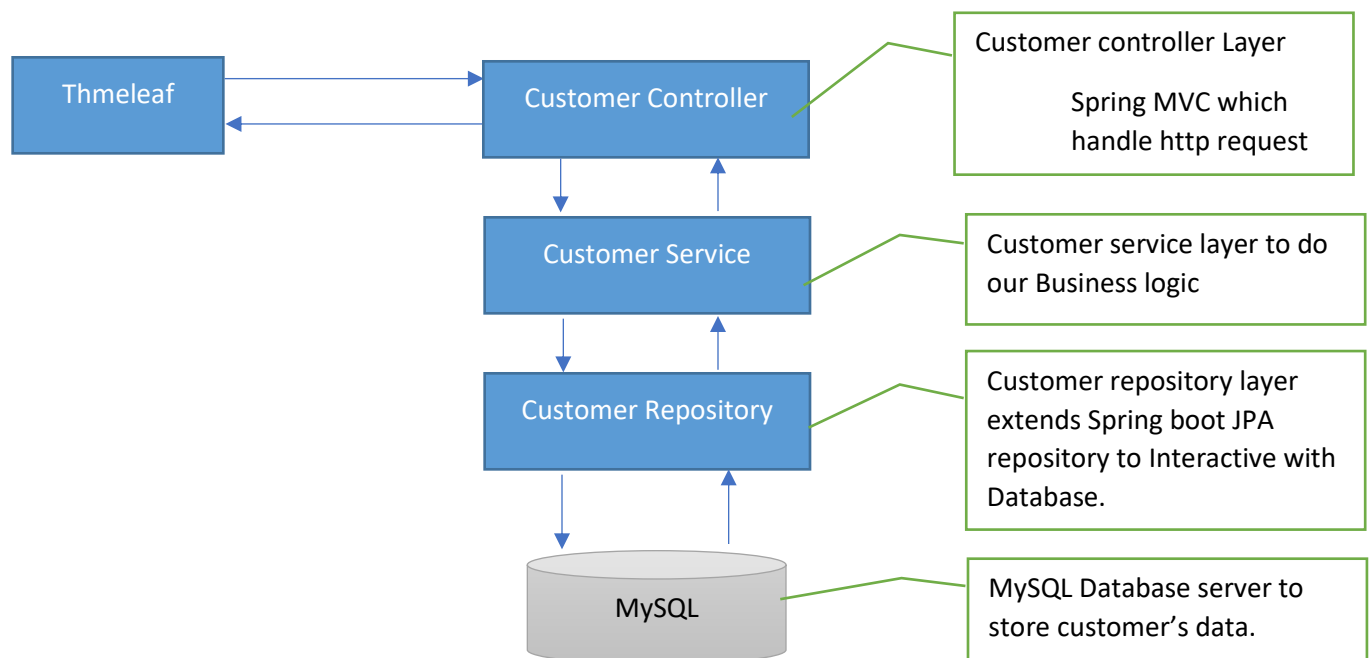
Table (customer)		
Field name	Datatype	Restriction
CustomerId	Long	PK
firstName	String	Not null
Surname	String	Not null
Mobile	String	Not null , unique
Password	String	Not null
Gender	String	Not null

Table (role)		
Field name	Datatype	Restriction
roleId	Long	PK
roleName	String	Not null

Many to many relationships



Application Architecture



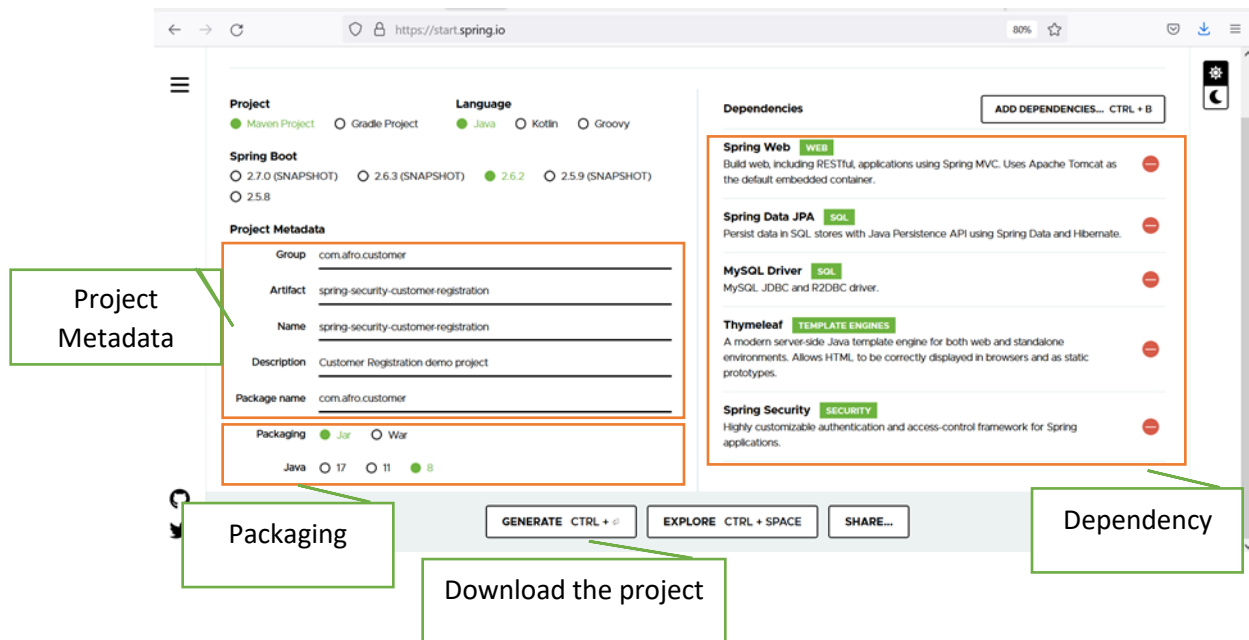
Technologies and tools used

- Spring boot 2.6.2
- Maven 3.2 +
- Java 1.8
- Thymeleaf
- Bootstrap css
- MySql
- Eclipse

1. Learn how to create and build a spring MVC web application using spring boot.

We are going to create spring boot template project from spring.start.io web site and download it, extract it and open it in eclipse here are the steps

1. Got <https://start.spring.io/> and add below dependency



- Spring web: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Spring data JPA: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- MySQL driver: MySQL JDBC driver.
- Thymeleaf : A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.
- Spring security : Highly customizable authentication and access-control framework for Spring applications.

Add also below dependency from maven repository website to POM file

```
<!-- https://mvnrepository.com/artifact/org.thymeleaf.extras/thymeleaf-extras-springsecurity5 -->
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>
  <version>3.0.4.RELEASE</version>
</dependency>
```

In order to integrate spring security in Thymeleaf and to display security details.

Project: select maven project,

Language: select Java,

Spring Boot: select 2.6.2 it's stable version and recommended by spring team

Packaging: select Jar.

Fill project metadata

Group

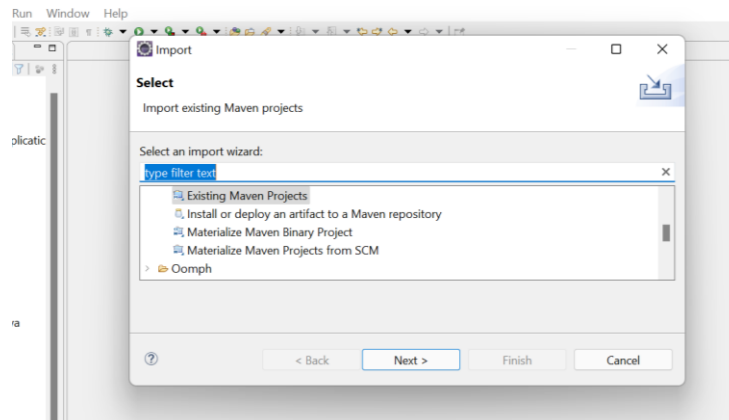
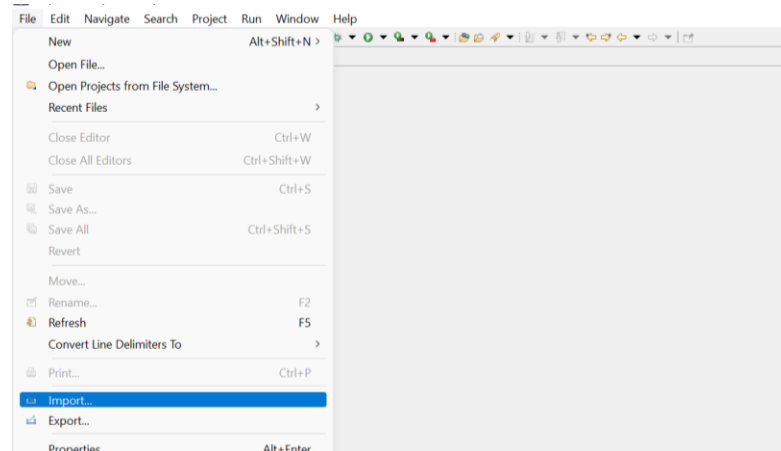
Artifact

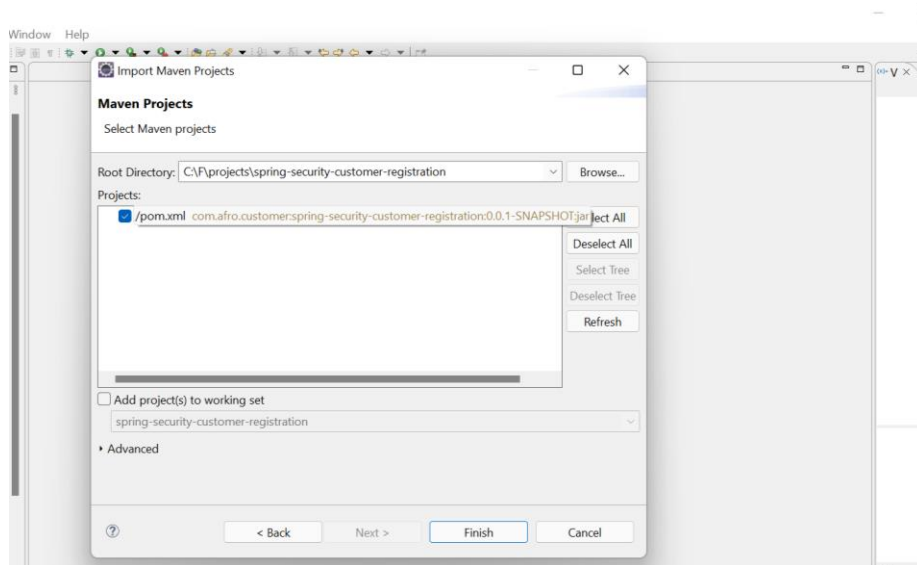
Name

Description

Package name

2. Open eclipse > file > import > existing maven project > browse your project > finish





Now our project will open in eclipse and download the libraries.

2. Learn how to configure MySQL Database in spring boot.

From our project got resources > open application.properties and add below properties

```
server.port = 8081
```

```
# database config
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/openMarket
```

```
spring.datasource.username=root
```

```
spring.datasource.password=root
```

```
#JPA Hibernate
```

```
spring.jpa.properties.hibernate.dialect = org.hibernate.MySQL5InnoDBDialect
```

```
spring.jpa.hibernate.ddl-auto= update
```

it's self-explain spring boot will take these values and set up our database connection

3. Learn how to create JPA entities.

we'll learn about the basics of entities along with various annotations that define and customize an entity in JPA.

Entity

Entities in JPA are nothing but POJOs representing data that can be persisted to the database. An entity represents a table stored in a database. Every instance of an entity represents a row in the table.

Entity annotation

Let's say we have a POJO called *customer* which represents the data of a customer and we would like to store it in the database.

```
public class Customer {  
  
    // fields, getters and setters  
  
}
```

In order to do this, we should define an entity so that JPA is aware of it.

So let's define it by making use of the *@Entity* annotation. We must specify this annotation at the class level.

```
@Entity  
public class Customer {  
  
    // fields, getters and setters  
  
}
```

Id annotation

Each JPA entity must have a primary key which uniquely identifies it. The *@Id* annotation defines the primary key. We can generate the identifiers in different ways which are specified by the *@GeneratedValue* annotation.

We can choose from four id generation strategies with the *strategy* element. **The value can be *AUTO*, *TABLE*, *SEQUENCE*, or *IDENTITY*.**

```
@Entity  
public class Customer {  
    @Id
```



```

        @GeneratedValue(strategy=GenerationType.AUTO)
        private Long customerId;

        private String firstName;

        // getters and setters
    }

```

If we specify *GenerationType.AUTO*, the JPA provider will use any strategy it wants to generate the identifiers.

If we annotate the entity's fields, the JPA provider will use these fields to get and set the entity's state. In addition to Field Access, we can also do Property Access or Mixed Access, which enables us to use both Field and Property access in the same entity.

Table annotation

In most cases, **the name of the table in the database and the name of the entity will not be the same.**

In these cases, we can specify the table name using the *@Table* annotation:

```

@Entity
@Table(name="CUSTOMER")
public class Customer {

    // fields, getters and setters

}

```

We can also mention the schema using the *schema* element:

```

@Entity
@Table(name="CUSTOMER", schema="MY_SCHEMA")
public class Customer {

    // fields, getters and setters

}

```

Schema name helps to distinguish one set of tables from another, If we do not use the *@Table* annotation, the name of the entity will be considered the name of the table.

Column annotation

Just like the *@Table* annotation, we can use the *@Column* annotation to mention the details of a column in the table.

The *@Column* annotation has many elements such as *name*, *length*, *nullable*, and *unique*.

```

@Entity

```

```

@Table(name="CUSTOMER")
public class customer {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long customerId;

    @Column(name="first_NAME", length=50, nullable=false, unique=false)
    private String firstName;

    // other fields, getters and setters
}

```

The *name* element specifies the name of the column in the table. The *length* element specifies its length. The *nullable* element specifies whether the column is nullable or not, and the *unique* element specifies whether the column is unique.

If we don't specify this annotation, the name of the field will be considered the name of the column in the table.

Transient annotation

Sometimes, we may want to **make a field non-persistent**. We can use the *@Transient* annotation to do so. It specifies that the field will not be persisted.

For instance, we can calculate the age of a customer from the date of birth.

So let's annotate the field *age* with the *@Transient* annotation:

```

@Entity
@Table(name="CUSTOMER")
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long customerId;

    @Column(name="FIRST_NAME", length=50, nullable=false)
    private String firstName;

    @Transient
    private Integer age;

    // other fields, getters and setters
}

```

As a result, the field *age* will not be persisted to the table

Temporal Annotation

In some cases, we may have to save temporal values in our table.

For this, we have the *@Temporal* annotation:

```
Entity
@Table(name="CUSTOMER")
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long customerId;

    @Column(name="FIRST_NAME", length=50, nullable=false, unique=false)
    private String firstName;

    @Transient
    private Integer age;

    @Temporal(TemporalType.DATE)
    private Date birthDate;

    // other fields, getters and setters
}
```

Many to Many annotations

in this tutorial, we'll explain only one way to **deal with many-to-many relationships using JPA** **there are other ways to do so.**

We'll use a model of customer, roles, and various relationships between them.

For the sake of simplicity, in the code examples, we'll only show the attributes and JPA configuration that's related to the many-to-many relationships.

Modeling a Many-to-Many Relationship

A relationship is a connection between two types of entities. In the case of a many-to-many relationship, both sides can relate to multiple instances of the other side.

Note that it's possible for entity types to be in a relationship with themselves. Think about the example of modeling family trees: Every node is a person, so if we talk about the parent-child relationship, both participants will be a person.

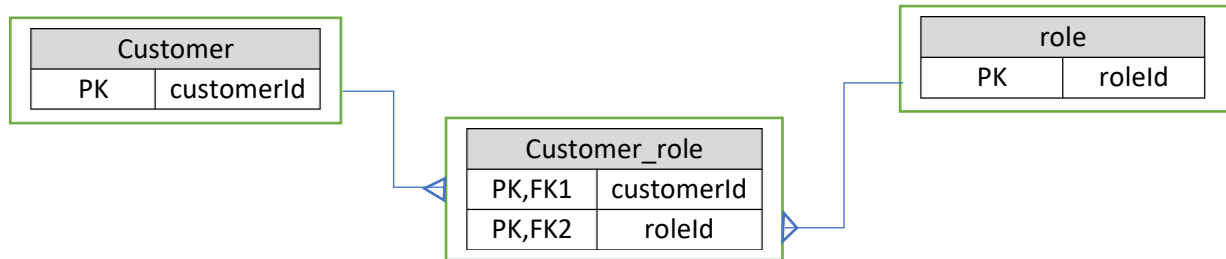
However, it doesn't make such a difference whether we talk about a relationship between single or multiple entity types. Since it's easier to think about relationships between two different entity types, we'll use that to illustrate our cases.

Let's take the example of customer has the roles to access certain resources.

A customer can have **many** roles, and **many** customers can have the same role:



As we know, in RDBMSs we can create relationships with foreign keys. Since both sides should be able to reference the other, **we need to create a separate table to hold the foreign keys**:



Such a table 'Customer_role' is called a join table. In a join table, the combination of the foreign keys will be its composite primary key.

We can do this with the `@JoinTable` annotation in the *Customer* class. We provide the name of the join table ('Customer_role') as well as the foreign keys with the `@JoinColumn` annotations. The `joinColumn` attribute will connect to the owner side of the relationship, and the `inverseJoinColumn` to the other side:

```
@ManyToMany (
    fetch = FetchType.EAGER,
    cascade = CascadeType.ALL)
@JoinTable(
    name = "Customer_role",
    joinColumns = @JoinColumn(name = "customer_id" ,
        referencedColumnName="customerId"),
    inverseJoinColumns = @JoinColumn(name = "role_id" ,
        referencedColumnName="roleId"))
Collection<Role> Roles;
```

Note that using `@JoinTable` or even `@JoinColumn` isn't required. JPA will generate the table and column names for us. However, the strategy JPA uses won't always match the naming conventions we use. So, we need the possibility to configure table and column names.

To create customer entity and role entity to map the database table which were mentioned in Project requirement, first we will create package by name model "com.afro.customer.model" .

1. Create Role class

```

package com.afro.customer.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "ROLE")
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long roleId;

    @Column(nullable = false)
    private String roleName;

    public Long getRoleId() {
        return roleId;
    }

    public String getRoleName() {
        return roleName;
    }

    public void setRoleId(Long roleId) {
        this.roleId = roleId;
    }

    public void setRoleName(String roleName) {
        this.roleName = roleName;
    }

}

```

2. Create Customer Class

```

package com.afro.customer.model;

import java.util.Collection;

import javax.persistence.CascadeType;
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.JoinTable;
import javax.persistence.ManyToMany;

```

```

import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

@Entity
@Table(name = "CUSTOMER", uniqueConstraints = { @UniqueConstraint(name =
"UNQ_CUSTOMER", columnNames = { "mobile" }) })
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long cutomerId;
    private String firstName;
    private String surname;
    private String mobile;
    private String password;
    private String gender;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name = "customer_role", joinColumns = @JoinColumn(name =
"customer_id", referencedColumnName = "cutomerId"), inverseJoinColumns =
@JoinColumn(name = "role_id", referencedColumnName = "roleId"))
    private Collection<Role> roles;

    public Long getCutomerId() {
        return cutomerId;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getSurname() {
        return surname;
    }

    public String getMobile() {
        return mobile;
    }

    public String getPassword() {
        return password;
    }

    public String getGender() {
        return gender;
    }

    public void setCutomerId(Long cutomerId) {
        this.cutomerId = cutomerId;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}

```

```
public void setSurname(String surname) {  
    this.surname = surname;  
}  
  
public void setMobile(String mobile) {  
    this.mobile = mobile;  
}  
  
public void setPassword(String password) {  
    this.password = password;  
}  
  
public void setGender(String gender) {  
    this.gender = gender;  
}  
  
public Collection<Role> getRoles() {  
    return roles;  
}  
  
public void setRoles(Collection<Role> roles) {  
    this.roles = roles;  
}  
}
```

In this section, we learned what JPA entities are and how to create them. We also learned about the different annotations that can be used to customize the entity further.

4. Learn how to configure spring security in spring boot.

Spring security prevent unauthorized users from viewing our resources, if visitors try to access our resources spring security forces them to login before they can see that page.

we do that by configuring Spring Security in spring boot application. If Spring Security is on the classpath, Spring Boot automatically secures all HTTP endpoints with “basic” authentication. However, we can further customize the security settings. The first thing we need to do is add Spring Security to the classpath and already did that by added spring security dependency element in pom.xml, as below :

```
<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-security</artifactId>

</dependency>
```

Let us go to our src and create config package com.afro.customer.config

Create class WebSecurityConfig in config package

```
package com.afro.customer.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.authentication.builders.Authentication
ManagerBuilder;
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.WebSecurityConfigure
rAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    UserDetailsService userDetailsService;

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

        auth.userDetailsService(userDetailsService);

    }
}
```



```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers("/registration**",
        "/css/**",
        "/js/**",
        "/img/**")
        .permitAll()
        .anyRequest()
        .authenticated()
        .and()
        .formLogin()
        .loginPage("/login")
        .permitAll()
        .and()
        .logout()
        .invalidateHttpSession(Boolean.TRUE)
        .clearAuthentication(Boolean.TRUE)
        .logoutRequestMatcher(new
            AntPathRequestMatcher("/logout"))
        .logoutSuccessUrl("/login?logout")
        .permitAll();
}

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

```

The `WebSecurityConfig` class is annotated with `@EnableWebSecurity` to enable Spring Security's web security support and provide the Spring MVC integration. It also extends `WebSecurityConfigurerAdapter` and overrides a couple of its methods to set some specifics of the web security configuration.

The `configure(HttpSecurity)` method defines which URL paths should be secured and which should not. Specifically, the `/registration`, `/css`, `/img`, `/js` and `/login` paths are configured to not require any authentication. All other paths must be authenticated.

When a user successfully logs in, they are redirected to the previously requested page that required authentication. There is a custom `/login` page (which is specified by `login()`), and everyone is allowed to view it.

The `UserDetailsService` interface is used to retrieve user-related data. It has one method named `loadUserByUsername()` which can be overridden to customize the process of finding the user.

in this section we've shown how to configure spring security in spring boot .

5. Learn how to implement spring security with backend database.

In this section, we will show how to create a custom database-backed *UserDetailsService* for authentication with Spring Security.

In order to provide our own user service, we will need to implement the *UserDetailsService* interface.

We'll create a class called *MyUserDetailsService* implements *UserDetailsService* and override the *loadUserByUsername()* methods , to retrieve the *User* object using the *JPA Repository* , and if it exists, wrap it into a *MyUserDetails* object, which implements *UserDetails*, and returns it:

Retrieving a User

For the purpose of retrieving a user associated with a username, we will create a *CustomerRepository* interface using *Spring Data* by extending the *JpaRepository* interface:

Create package by name repository

```
package com.afro.customer.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import com.afro.customer.model.Customer;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

    public Optional<Customer> findByMobile(String mobile);

}
```

The UserDetailsService

In order to provide our own customer *MyUserDetailsService*, we will need to implement the *UserDetailsService* interface.

We'll create service package and a class called *MyUserDetailsService* that overrides the method *loadUserByUsername()* of the interface.

In this method, we retrieve the *User* object using the *CustomerRepository*, and if it exists, wrap it into a *MyUserDetails* object, which implements *UserDetails*, and returns it:

```
package com.afro.customer.model;

import java.util.Collection;
```

```

import java.util.List;
import java.util.stream.Collectors;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

public class MyUserDetails implements UserDetails {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private Customer customer;

    public MyUserDetails(Customer customer) {
        super();
        this.customer = customer;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        // TODO Auto-generated method stub
        return getGrantedAuthority(customer.getRoles());
    }

    @Override
    public String getPassword() {
        // TODO Auto-generated method stub
        return customer.getPassword();
    }

    @Override
    public String getUsername() {

        return customer.getMobile();
    }

    @Override
    public boolean isAccountNonExpired() {
        // TODO Auto-generated method stub
        return true; // just change to true hard code
    }

    @Override
    public boolean isAccountNonLocked() {
        return false; // just change to true hard code
    }

    @Override
    public boolean isCredentialsNonExpired() {

        return true; // just change to true hard code
    }
}

```

```

@Override
public boolean isEnabled() {

    return true; // just change to true hard code

}

// -----
private List<GrantedAuthority> getGrantedAuthority(Collection<Role> roles) {
    return roles.stream().map(role -> new
SimpleGrantedAuthority(role.getRoleName())).collect(Collectors.toList());
}

}

package com.afro.customer.service;

import java.util.Optional;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import com.afro.customer.model.Customer;
import com.afro.customer.model.MyUserDetails;
import com.afro.customer.repository.CustomerRepository;

@Service
public class MyUserDetailsService implements UserDetailsService {

    private CustomerRepository customerRepository;

    public MyUserDetailsService(CustomerRepository customerRepository) {
        super();
        this.customerRepository = customerRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

        Optional<Customer> customer = customerRepository.findByMobile(username);

        customer.orElseThrow(() -> new UsernameNotFoundException("INVALID USER
NAME OR PASSWARD"));

        return customer.map(MyUserDetails::new).get();
    }

}

```

in this section we've learn how to implement spring security with backend database.

6. Learn How to develop end to end Customer Registration implementation.

We are going create below :

- a. CustomerServiceImpl class to write our business logic
- b. CustomerController class to handle http request.
- c. CustomerRepository interface extends Jpa repository to manage connection with MySQL database .
- d. Thymeleaf as presentation layer .

a. CustomerServiceImpl :

We need to add one DTO

- CustomerDTO it's DTO class DTO stand for Data Transfer Object , it's design pattern to transfer data between client and server .
- CustomerService interface
- CustomerServiceImpl class implement CustomerService interface

CustomerDTO :

```
package com.afro.customer.dto;

public class CustomerDTO {

    // Data Transfer Object to transfer data between client and server
    private String firstName;
    private String surname;
    private String mobile;
    private String password;
    private String gender;

    public String getFirstName() {
        return firstName;
    }
    public String getSurname() {
        return surname;
    }
    public String getMobile() {
        return mobile;
    }
    public String getPassword() {
        return password;
    }
}
```

```

    }
    public String getGender() {
        return gender;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }
    public void setMobile(String mobile) {
        this.mobile = mobile;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
}

```

CustomerService interface

```

package com.afro.customer.service;
import com.afro.customer.dto.CustomerDTO;
import com.afro.customer.model.Customer;

public interface CustomerService {

    public Customer saveCustomer(CustomerDTO dto);

}

```

CustomerServiceImpl

```

package com.afro.customer.service;

import java.util.Arrays;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import com.afro.customer.dto.CustomerDTO;

```

```

import com.afro.customer.model.Customer;
import com.afro.customer.model.Role;
import com.afro.customer.repository.CustomerRepository;

@Service
public class CustomerServiceImpl implements CustomerService {

    @Autowired
    private BCryptPasswordEncoder getBCryptPasswordEncoder;
    // inject BCryptPasswordEncoder Bean in encode password

    private CustomerRepository customerRepository ;

    public CustomerServiceImpl(CustomerRepository customerRepository) {
        super();
        this.customerRepository = customerRepository;
    }

    @Override
    public Customer saveCustomer(CustomerDTO dto) {

        return customerRepository.save(
            new Customer(dto.getFirstName(),
                dto.getSurname(),
                dto.getMobile(),

            getBCryptPasswordEncoder.encode(dto.getPassword()),
                dto.getGender(),
                Arrays.asList(new Role("USER")))) ;
    }
}

```

b. CustomerController :

```

package com.afro.customer.mvc;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
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.afro.customer.dto.CustomerDTO;
import com.afro.customer.service.CustomerService;

@Controller
@RequestMapping("/registration")

```

```

public class CustomerController {

    //inject Customer service
    private CustomerService customerService ;
    // there are two way to inject it's through @Autowired or constructor
    public CustomerController(CustomerService customerService) {
        super();
        this.customerService = customerService;
    }

    @GetMapping
    public String showCreateAccount(Model model)
    {
        //get request to show create account page
        CustomerDTO customerDto = new CustomerDTO();
        model.addAttribute("customer",customerDto);
        return "createAccount" ;
    }

    @PostMapping
    public String createAccount(@ModelAttribute("customer") CustomerDTO
customerDTO)
    {
        // post mapping to handle save customer data
        customerService.saveCustomer(customerDTO);
        return "redirect:/registration?success" ;
    }

}

```

c. CustomerRepository :

```

package com.afro.customer.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import com.afro.customer.model.Customer;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

    //no need to Annotate interface by @Repository because spring boot will
    take care of it
    // JpaRepository will provide many method out of the box
    //for more information check spring website
    // we added custom method we will use it in login processing
    public Optional<Customer> findByMobile(String mobile);

}

```

d. Thymeleaf :

Thymeleaf is a Java template engine for processing and creating HTML, XML, JavaScript, CSS, and text.

In this article, we will discuss **how to use Thymeleaf with Spring** along with some basic use cases in the view layer of a Spring MVC application.

We are only cover which we need in our tutorial for more details please visit <https://www.thymeleaf.org/>

To create thymeleaf we will go to resources > templates> create createAccount.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Customers Registration System </title>
<link th:href="@{/css/main.css}" rel="stylesheet"/>
</head>
<body>
<!-- nav bar using bootstrap -->
<nav class="navbar navbar-dark bg-dark">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">Customers Registration System</a>
    </div>

    <div >
      <ul >

        <li >

        </li>

      </ul>
    </div>
  </div>
</nav>
<!-- end nav bar -->

<br />

  <div class="container">

    <div class="row">
      <div class="mx-auto col-md-6 col-md-offset-3">

        <div class="row">
          <h1 class="text-center">Create Account</h1>
        </div>

      </div>

    </div>

  </div>

</body>
</html>
```

```

<!-- success message -->
<div th:if="${param.success}">
    <div class="alert alert-info">
        Account Created Successfully
    </div>
</div>

<form th:action="@{/registration}" th:object="${customer}"
method="POST">

    <div class="form-group">
    <div class="md-3">
        <label for="firstName"> First Name </label>
        <input
            type="text"
            name="firstName"
            class="form-control"
            id="firstName"
            placeholder="Please enter First Name"
            th:field="*{firstName}"
            required="required"
            autofocus="autofocus"
        />
    </div>

    <div class="md-3">
        <label for="surname"> Surname </label>
        <input
            type="text"
            name="surname"
            class="form-control"
            id="surname"
            placeholder="Please enter Surname"
            th:field="*{surname}"
            required="required"
        />
    </div>

    <div class="md-3">
        <label for="mobile"> Mobile No </label>
        <input
            type="text"
            name="mobile"
            class="form-control"
            id="mobile"
            placeholder="Please enter Mobile No"
            th:field="*{mobile}"
            required="required"
        />
    </div>

    <div class="md-3">
        <label for="password"> Password </label>
        <input

```

```

        type="password"
        name="password"
        class="form-control"
        id="password"
        placeholder="Please enter password"
        th:field="*{password}"
        required="required"
    />
</div>

<div class="md-3">
    <div class="form-check form-check-inline">
        <input class="form-check-input"
            type="radio"
            name="genderM"
            id="gender"
            value="M"
            th:field="*{gender}"
            checked="checked">
        <label class="form-check-label"
            for="genderM">
                Male
            </label>
        </div>

        <div class="form-check form-check-inline">
            <input class="form-check-input"
                type="radio"
                name="genderF"
                id="genderM"
                th:field="*{gender}"
                value="F">
            <label class="form-check-label"
                for="genderF">
                    Female
                </label>
            </div>
        </div>
        </div>
        <br/>
        <div class="box-footer">
            <button type="submit" class="btn btn-primary">
                Create Account
            </button>
            <span>
                I have Account
                <a th:href="@{/Login}"> Login here </a>
            </span>
        </div>
    </form>

    </div>
    </div>
</div>

```

```
</body>
```

```
</html>
```

We are bootstrap here is how to integrate bootstrap with thymeleaf

there are two to use load bootstrap into thymeleaf

1. from bootstrap website copy css link and paste it in thymeleaf header as below

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94
      WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
      crossorigin="anonymous">
```

2. or copy this address <https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css> and open it in browser and copy the response and paste in under resources > css > main.css and import the main file in thymeleaf header

```
<label for=""> </label> : label
```

```
<input
                                type="text"
                                name="mobile"
                                class="form-control"
                                id="mobile"
                                placeholder="Please enter Mobile No"
                                th:field="*{mobile}"
                                required="required"
                                />
```

th:field="{mobile}" : mobile bind to customer model' mobile property*

```
<form th:action="@{/registration}" th:object="${customer}" method="POST">
Th: = thymeleaf namespace
th:action="@{/registration}" : @ context project path
    /registration : request mapping to CustomerController
```

th:object="\${customer}" : spring MVC model

In this section, we have Learned How to develop end to end Customer Registration implementation.

7. Learn how to develop custom login.

```
import com.afro.customer.model.Customer;
import com.afro.customer.model.MyUserDetails;
import com.afro.customer.repository.CustomerRepository;

@Service
public class MyUserDetailsService implements UserDetailsService {

    private CustomerRepository customerRepository;

    public MyUserDetailsService(CustomerRepository customerRepository) {
        super();
        this.customerRepository = customerRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
    UsernameNotFoundException {

        Optional<Customer> customer = customerRepository.findByMobile(username);

        customer.orElseThrow(() -> new UsernameNotFoundException("UserName or
    Password Not Valid"));

        return customer.map(MyUserDetails::new).get();
    }
}

package com.afro.customer.model;

import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

public class MyUserDetails implements UserDetails {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
```

```

private Customer customer;

public MyUserDetails(Customer customer) {
    super();
    this.customer = customer;
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    // TODO Auto-generated method stub
    return getGrantedAuthority(customer.getRoles());
}

@Override
public String getPassword() {
    // TODO Auto-generated method stub
    System.out.println("customer.getPassword() :" + customer.getPassword());
    return customer.getPassword();
}

@Override
public String getUsername() {
    System.out.println("customer.getMobile() :" + customer.getMobile());
    return customer.getMobile();
}

@Override
public boolean isAccountNonExpired() {
    // TODO Auto-generated method stub
    return true; // just change to true hard code
}

@Override
public boolean isAccountNonLocked() {
    return true; // just change to true hard code
}

@Override
public boolean isCredentialsNonExpired() {
    return true; // just change to true hard code
}

@Override
public boolean isEnabled() {
    return true; // just change to true hard code
}

// -----
private List<GrantedAuthority> getGrantedAuthority(Collection<Role> roles) {

```

```

        return roles.stream().map(role -> new
SimpleGrantedAuthority(role.getRoleName())).collect(Collectors.toList());
    }

}

```

```

package com.afro.customer.mvc;

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

@Controller
public class SecurityController {

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

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

```

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Customers Registration System </title>
<link th:href="@{/css/main.css}" rel="stylesheet"/>
</head>
<body>

    <!-- nav bar using bootstrap -->
    <nav class="navbar navbar-dark bg-dark">
        <div class="container-fluid">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">Customers Registration
System</a>
            </div>

            <div >
                <ul > <li ></li>
                </ul>
            </div>
        </div>
    </nav>

```

```

        <!-- end nav bar -->
<br />

<div class="container">

    <div class="mx-auto col-md-6 col-md-offset-3 ">

        <div class="row">
            <h1 class="text-center">Login Form</h1>
        </div>

        <!-- logged out message -->
        <div th:if="${param.logout}">
            <div class="alert alert-info">
                You have been Logged out
            </div>
        </div>

        <!-- show Invalid User Name or Password . if customer enter wrong
credential -->
        <div th:if="${param.error}">
            <div class="alert alert-danger">
                Invalid User Name or Password .
            </div>
        </div>

        <br />

        <form th:action="@{/Login}" method="POST">
            <div class="form-group">
                <div class="md-3">
                    <label for="username"> User Name </label>
                    <input
                        type="text"
                        name="username"
                        class="form-control"
                        id="username"
                        placeholder="Please enter Mobile No"
                        autofocus="autofocus"
                    />
                </div>

                <div class="md-3">
                    <label for="password"> Password </label>
                    <input
                        type="password"
                        name="password"
                        class="form-control"
                        id="password"
                        placeholder="Please enter password"
                    />
                </div>
            </div>
        </form>
    </div>

```



```

        />
    </div>

</div>

<br/>
<div class="mx-auto col-md-6">
    <button type="submit" class="btn btn-primary">
        Login
    </button>

</div>
</form>
</div>

<br/>

<div class="mx-auto col-md-6 col-md-offset-3">
    <span>
        New User ?
        <a th:href="@{/registration}">Sign Up </a>
    </span>
</div>

</div>
</div>

</body>

</html>

```

8. Learn how to integrate spring security in Thymeleaf and how to display security details.

To display principal in thymeleaf we have to add below dependency in pom.xml

```

<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>

</dependency>

```

Add below namespace in thymeleaf template page

```

<html
    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">

    sec:authorize="isAuthenticated()" : return true if customer successfully login

```

`` : display user name in thymeleaf template page .

9. Learn how to implement spring security logout.

```
<!DOCTYPE html>
<html
    xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<meta charset="ISO-8859-1">
<title>Customers Registration System </title>
<link th:href="@{/css/main.css}" rel="stylesheet"/>

<style>
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333333;
    float: left;
}

li {
    float: left;
}

li a {
    display: block;
    color: white;
    text-align: center;
    padding: 16px;
    text-decoration: none;
}

li a:hover {
    background-color: #111111;
}
</style>
</head>
<body>
<!-- nav bar -->

<nav class="navbar navbar-dark bg-dark">
    <div class="container-fluid">
        <div class="navbar-header">
            <a class="navbar-brand" href="#">Customers Registration System</a>
        </div>

        <div >
            <ul >
                <li class="active"><a href="#">Home</a></li>
```

```

        <li sec:authorize="isAuthenticated()">
            <a th:href="@{/Logout}"> Logout </a>

        </li>

    </ul>
</div>
</div>
</nav>
<!-- end nav bar -->

<br />

<div class="container">

    <div class="mx-auto col-md-10 col-md-offset-1 ">

        <div class="row">
            <div class="col col-md-3">
                <div class="row">
                    <div class="card">
                        Welcome :
                        <span
sec:authentication="principal.username">
                        </span>
                    </div>
                    <div class="vr" style="height: 300px;"></div>
                </div>
            </div>
            <div class="col col-md-7">
                <h1> Home Page </h1>

            </div>

        </div>

    </div>

</div>
</div>

</body>
</html>

```

Pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>

```

```

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.2</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.afro.customer</groupId>
<artifactId>spring-security-customer-registration-thymeleaf</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>spring-security-customer-registration-thymeleaf</name>
<description>Customer Registration demo project </description>
<properties>
    <java.version>1.8</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

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

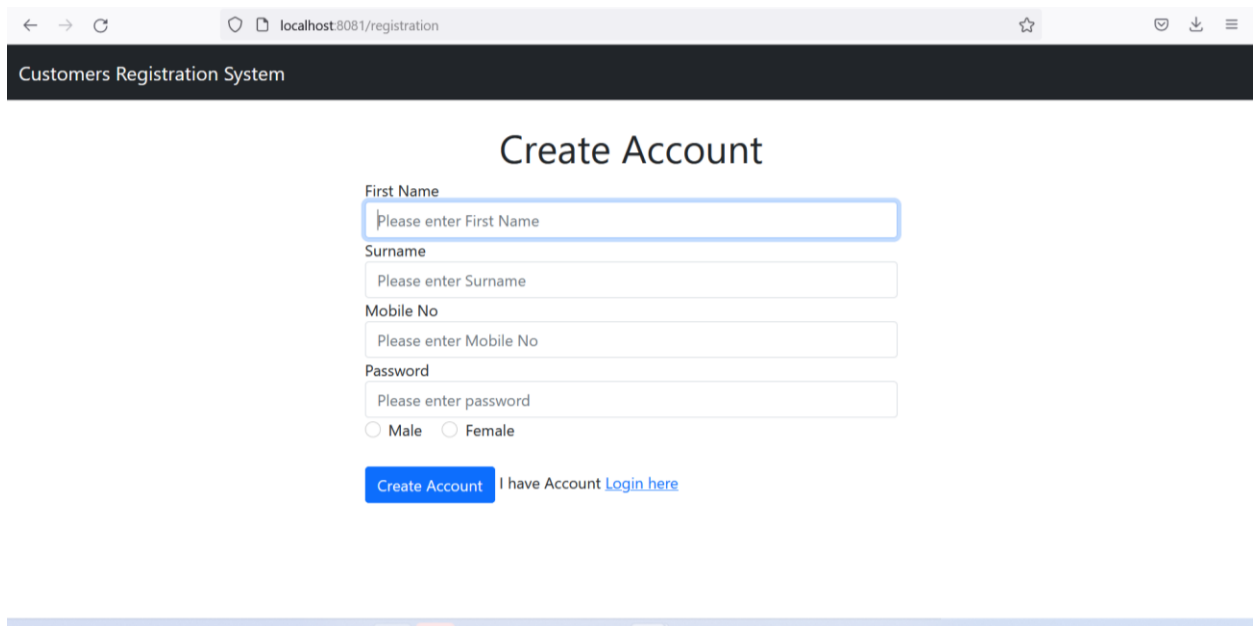
<build>
    <plugins>

```

```
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
      </plugins>
    </build>
  </project>
```

Test we configure embedded port server to 8081

Open browser and type <http://localhost:8081/registration>



The screenshot shows a web browser window with the address bar displaying 'localhost:8081/registration'. The page title is 'Customers Registration System'. The main heading is 'Create Account'. Below the heading, there is a form with the following fields and labels:

- First Name:
- Surname:
- Mobile No:
- Password:
- Gender: ☐ Male ☐ Female

At the bottom of the form, there is a blue button labeled 'Create Account' and a link that says 'I have Account [Login here](#)'.

When we fill the form and click create account

← → ↻ localhost:8081/registration?success ☆ 📄 ⬇ ⋮

Customers Registration System

Create Account

Account Created Successfully

First Name

Surname

Mobile No

Password

☐ Male ☐ Female

[Create Account](#) I have Account [Login here](#)

Login page

← → ↻ localhost:8081/login ☆ 📄 ⬇ ⋮

Customers Registration System

Login Form

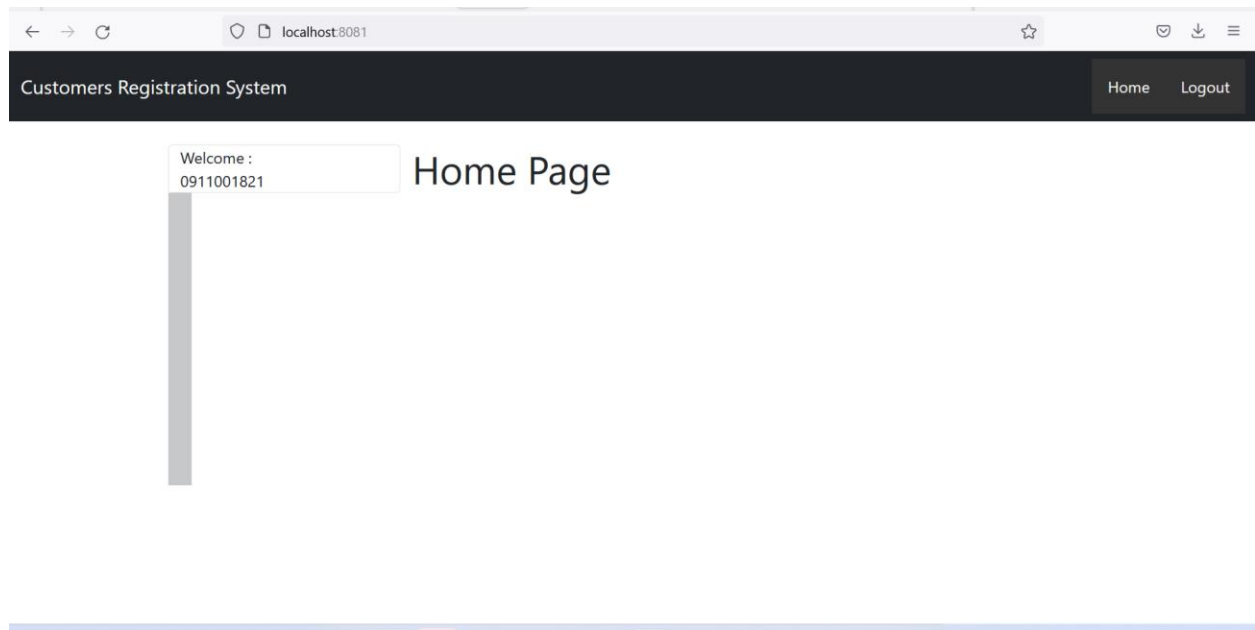
User Name

Password

[Login](#)

New User ? [Sign Up](#)

Home page



References

1. <https://www.baeldung.com>
2. <https://spring.io>
3. <https://www.javaguides.net>
- 4.