Creating a login form

Estimated time needed: 20 minutes

Overview

In this lab, you will be provided with the project structure for a Spring MVC application created using Spring web packages for web pages, Thymeleaf for templating, and Spring Web Security for authentication. You will be adding authorization based on roles in the application to render different home pages.

Learning objectives

After completing this lab, you will be able to:

- Clone the Spring MVC app, import it in the workspace, and test it
- Modify the `User` model to handle roles
- Modify the security configuration to allow authentication and authorization
- Create webpages forms with Thymeleaf specific to roles
- Create Spring MVC validation on the model object using the @Valid annotation and validate the data based on constraints defined in the model class
- Run the application

Prerequisites (optional)

You should know basic Java programming before you get started with this lab. Please ensure that you complete the labs sequentially as the are constructed progressively. Some background in HTML, CSS will be useful.

Clone the Spring Project and run

1. Open a terminal and run the following command to clone the Spring MVC project.

   ```
   git clone https://github.com/ibm-developer-skills-network/qpwlw-spring_secure_auth_lab.git
   ```

2. Change to the project directory.

   ```
   cd /home/project/qpwlw-spring_secure_auth_lab/secureauthapp
   ```

3. Use `mvn clean` (maven clean) to clean any existing class files and and `mvn install` (maven install) to compile the files in the project directory and generate the runnable jar file.

   ```
   mvn clean install
   ```

4. Run the following command to start the application. It will start in port `8080`.

   ```
   mvn exec:java -Dexec.mainClass="com.example.secureauthapp.SecureappApplication"
   ```

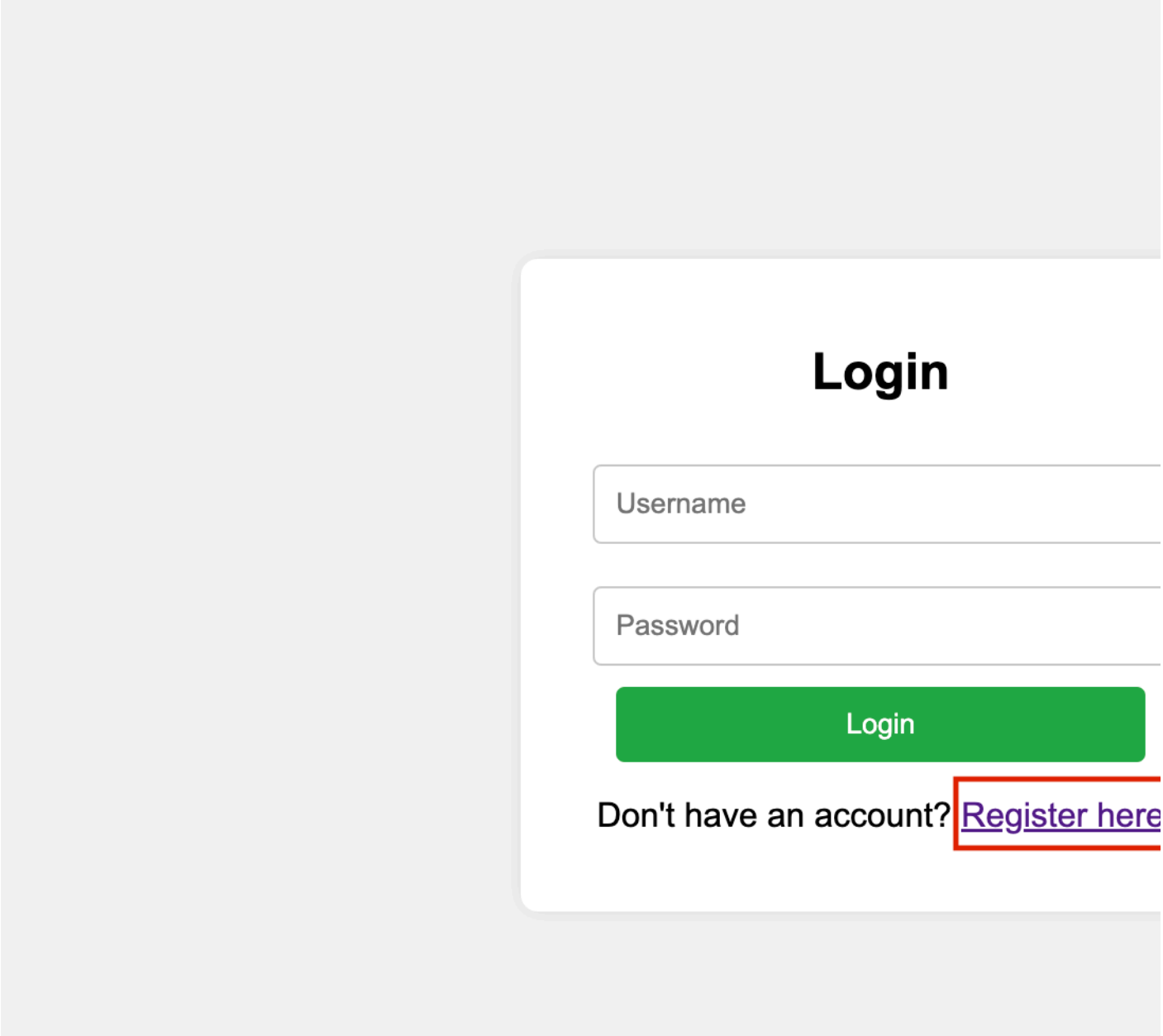
The server will start in port 8080.

6. Click on the button below to open the browser page to access the end point.

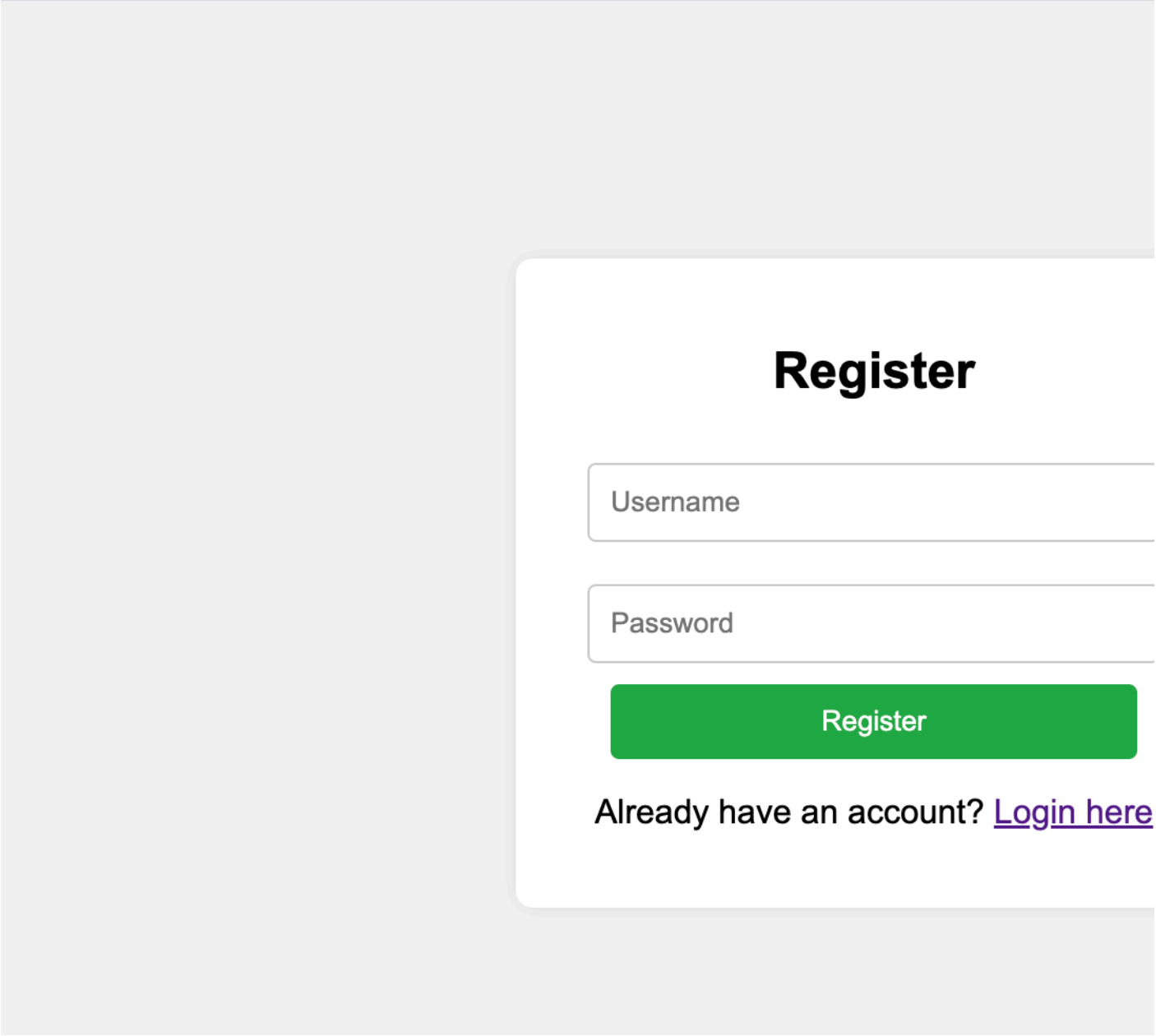
Website

It should automatically redirect you to the login page.

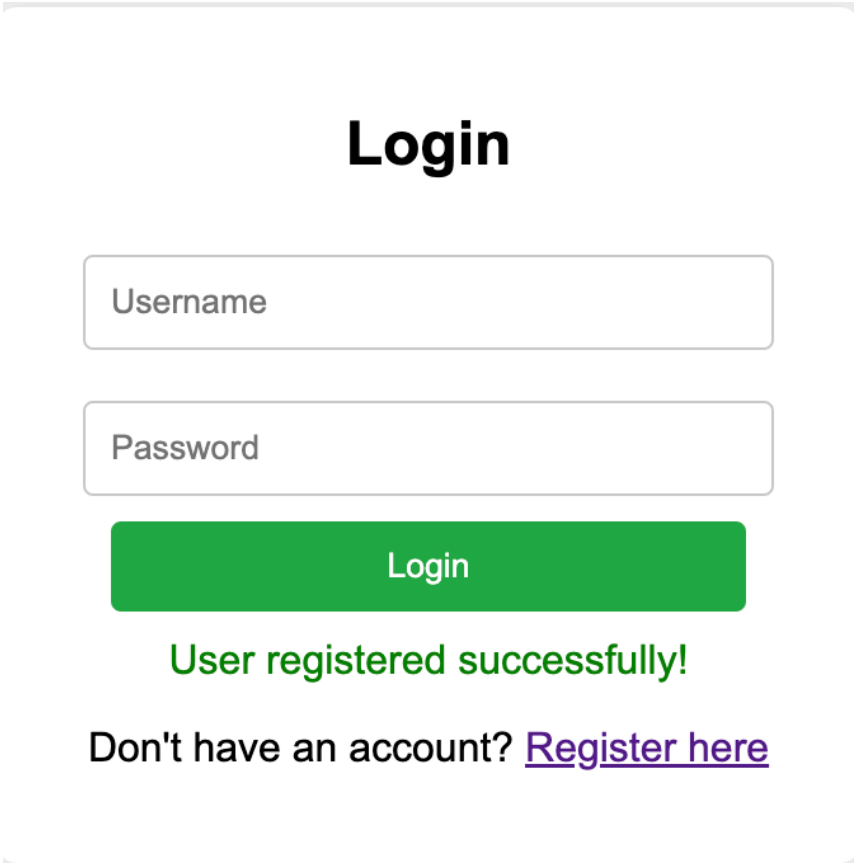
You will see the login page as give below.



7. First you need to register a user to login. Click on the `Register Here` link. This will take you to the register page.



8. Enter the user login and password and select `Register`. Once the user registers, the registration is confirmed and login page is displayed.



9. Log in with the username and password you registered with. Once the login is successful, `Hello` greeting is displayed along with the username.

10. When the username already registered is used to register again, an appropriate error message is displayed. When the username or password provided for login is incorrect, an appropriate error message is displayed.
11. Press `Ctrl+C` on the terminal and stop the server.

Make changes to User model and CustomerUserDetailsService

1. Open `com/example/secureauthapp/model/User.java` for editing.
2. Copy paste the code below and replace the existing content in the file. This includes the role attribute for the user.

```
package com.example.secureauthapp.model;
public class User {
    private String username;
    private String password;
    private String role; // Add a role field
    public User(String username, String password, String role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }
    // Getters and Setters
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
```

```
    public String getRole() {
        return role;
    }
    public void setRole(String role) {
        this.role = role;
    }
}
```

3. The `CustomerUserDetailService` class facilitates user registration. You should make changes in it to take the user role while constructing the user. Open `CustomerUserDetailService.java` to edit.

4. Copy paste the code below and replace the existing content in the file. This includes the role attribute for the user while registering the user. It also returns the role of the user while logging in.

```
package com.example.secureauthapp.service;
import com.example.secureauthapp.model.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;
@Service
public class CustomUserDetailsService implements UserDetailsService {
    private final Map<String, User> users = new HashMap<>();
    private final PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = users.get(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }
        return org.springframework.security.core.userdetails.User.builder()
            .username(user.getUsername())
            .password(user.getPassword())
            .roles(user.getRole())
            .build();
    }
    public void registerUser(String username, String password, String role) throws Exception {
        if(users.containsKey(username)) {
            throw new Exception("User already exists");
        } else {
            String encodedPassword = passwordEncoder.encode(password);
            users.put(username, new User(username, encodedPassword, role));
        }
    }
}
```

Modify WebSecurityConfig and WebsiteController

1. Open the `com/example/secureauthapp/config/WebSecurityConfig.java`.

2. You will include checks for roles in this. When the user is authenticated, you will render the home page. You will also ensure that any other request to the site are authenticated. Copy paste the following code to replace the existing content in `WebSecurityConfig.java`.

```
package com.example.secureauthapp.config;
import com.example.secureauthapp.service.CustomUserDetailsService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
    private final CustomUserDetailsService userDetailsService;
    public WebSecurityConfig(CustomUserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/register", "/login").permitAll() // Allow access to registration and login pages
                .requestMatchers("/admin").hasRole("ADMIN") // Restrict /admin to users with the ADMIN role
                .requestMatchers("/viewer").hasRole("STAFF") // Restrict /viewer to users with the STAFF role
                .anyRequest().authenticated() // Require authentication for all other endpoints
```

```
            )
            .formLogin(form -> form
                .loginPage("/login") // Custom login page
                .defaultSuccessUrl("/home", true) // Redirect to /greet after successful login
                .permitAll()
            )
            .logout(logout -> logout
                .permitAll()
            );
        return http.build();
    }
    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {
        AuthenticationManagerBuilder authenticationManagerBuilder =
            http.getSharedObject(AuthenticationManagerBuilder.class);
        authenticationManagerBuilder
            .userDetailsService(userDetailsService) // Use your custom UserDetailsService
            .passwordEncoder(passwordEncoder()); // Use the password encoder
        return authenticationManagerBuilder.build();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

3. Open the `com/example/secureauthapp/controller/WebsiteController.java`.

4. Make changes in the controller to provide mapping for `/home` API end point. This endpoint should render a different HTML pages for the admin and staff. Copy paste the following code in `WebsiteController.java`.

```
package com.example.secureauthapp.controller;
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import com.example.secureauthapp.service.CustomUserDetailsService;
@Controller
public class WebsiteController {
    private final CustomUserDetailsService userDetailsService;
    private final AuthenticationManager authenticationManager;
    public WebsiteController(CustomUserDetailsService userDetailsService, AuthenticationManager authenticationManager) {
        this.userDetailsService = userDetailsService;
        this.authenticationManager = authenticationManager;
    }
    @GetMapping("/home")
    public String homepage(Model model) {
        // Get the authenticated user's details
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        // Check if the user is authenticated
        if (authentication == null || !authentication.isAuthenticated() || authentication instanceof AnonymousAuthenticationToken) {
            // Redirect to the login page if the user is not authenticated
            return "redirect:/login";
        }
        // Get the username
        String username = authentication.getName();
        model.addAttribute("username", username);
        // Get the user's role
        String role = authentication.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority)
            .findFirst()
            .orElse("ROLE_STAFF"); // Default role if no authority is found
        // Redirect to the appropriate page based on the role
        if (role.equals("ROLE_ADMIN")) {
            return "admin"; // Return the admin.html template
        } else {
            return "viewer"; // Return the viewer.html template
        }
    }
    @GetMapping("/login")
    public String login() {
        return "login"; // Returns the login.html template
    }
    @GetMapping("/register")
    public String register() {
        return "register"; // Returns the register.html template
    }
    // POST endpoint to handle user registration and auto-login
    @PostMapping("/register")
    public String registerUser(
            @RequestParam String username, // Username from the form
            @RequestParam String password, // Password from the form
            @RequestParam String role // Role from the form
    ) {
        // Register the user by storing their details in the HashMap
        try {
            userDetailsService.registerUser(username, password, role);
        } catch (Exception userExistsAlready) {
            // Redirect to the /register endpoint
            return "redirect:/register?error";
```

```
        }
        // Authenticate the user programmatically
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(username, password)
        );
        // Set the authentication in the SecurityContext
        SecurityContextHolder.getContext().setAuthentication(authentication);
        // Redirect to the /login endpoint
        return "redirect:/login?success";
    }
}
```

Modify the Register user page

1. Select the button below to open the `register.html`.

Open **register.html** in IDE

2. Make changes in the page to provide input of role while registering the user. Copy paste the following code in `register.html`.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registration Page</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }
        .registration-container {
            background-color: #fff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            width: 300px;
            text-align: center;
        }
        .registration-container h2 {
            margin-bottom: 20px;
        }
        .registration-container input, .registration-container select {
            width: 80%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        .registration-container button {
            width: 80%;
            padding: 10px;
            background-color: #2d2db0;
            color: #fff;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }
        .registration-container button:hover {
            background-color: #2d2db0;
        }
        .error {
            color: red;
            margin-top: 10px;
        }
    </style>
</head>
<body>
    <div class="registration-container">
        <h2>Register</h2>
        <form th:action="@{/register}" method="post" class="form-group">
            <!-- Username -->
            <input type="text" name="username" placeholder="Username" required>
            <!-- Password -->
            <input type="password" name="password" placeholder="Password" required>
            <!-- Role Dropdown -->
            <select name="role" required>
                <option value="" disabled selected>Select your role</option>
                <option value="ADMIN">Admin</option>
                <option value="STAFF">Staff</option>
            </select>
            <!-- Submit Button -->
            <button type="submit">Register</button>
        </form>
        <!-- Error Message -->
        <div th:if="${param.error}" class="error">
            Invalid username.
        </div>
        <p>Already have an account? <a href="/login">Login here</a></p>
```

```
    </div>
</body>
</html>
```

Add admin and viewer pages

1. Create a new file name `admin.html` under `resources/templates` directory.
2. Paste the following code in the file.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Page</title>
</head>
<body>
    <h1>Admin Dashboard</h1>
    <h1>Welcome, <span th:text="${username}">. You have logged in as admin</span>!</h1>
    <p>Total turnover this month $3500000</p>
    <p>Current stock worth $2500000</p>
    <form th:action="@{/logout}" method="post">
        <button type="submit">Logout</button>
    </form>
</body>
</html>
```

3. Create a new file name `viewer.html` under `resources/templates` directory.
4. Paste the following code in the file.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Viewer Page</title>
</head>
<body>
    <h1>Viewer Dashboard</h1>
    <h1>Hello, <span th:text="${username}"></span>!</h1>
    <h1>Clothing Categories</h1>
    <select name="categories" id="categories">
        <option value="" disabled selected>Select a category</option>
        <option value="mens-clothing">Men's Clothing</option>
        <option value="womens-clothing">Women's Clothing</option>
        <option value="kids-clothing">Kids' Clothing</option>
        <option value="shoes">Shoes</option>
        <option value="accessories">Accessories</option>
        <option value="sportswear">Sportswear</option>
    </select>
    <form th:action="@{/logout}" method="post">
        <button type="submit">Logout</button>
    </form>
</body>
</html>
```

Run the application

1. Use `mvn clean` (maven clean) to clean any existing class files and and `mvn install` (maven install) to compile the files in the project directory and generate the runnable jar file.

```
mvn clean install
```

2. Run the following command to start the application. It will start in port `8080`.

```
mvn exec:java -Dexec.mainClass="com.example.secureauthapp.SecureappApplication"
```

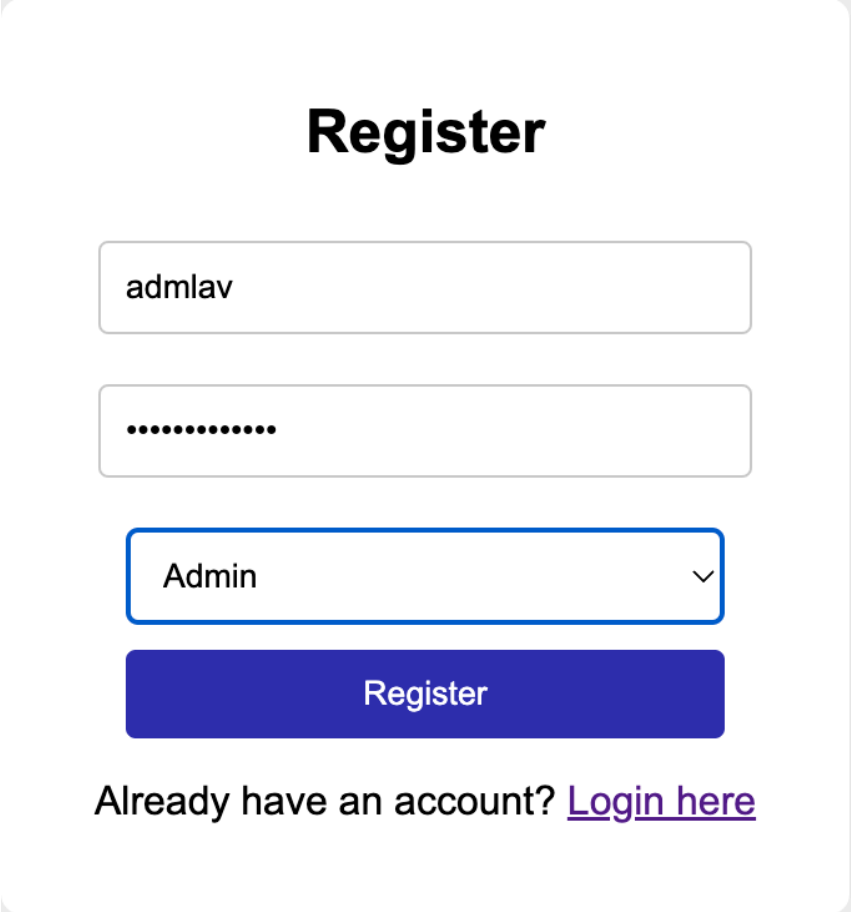
The server will start in port 8080.

3. Click on the button below to open the browser page to access the end point.

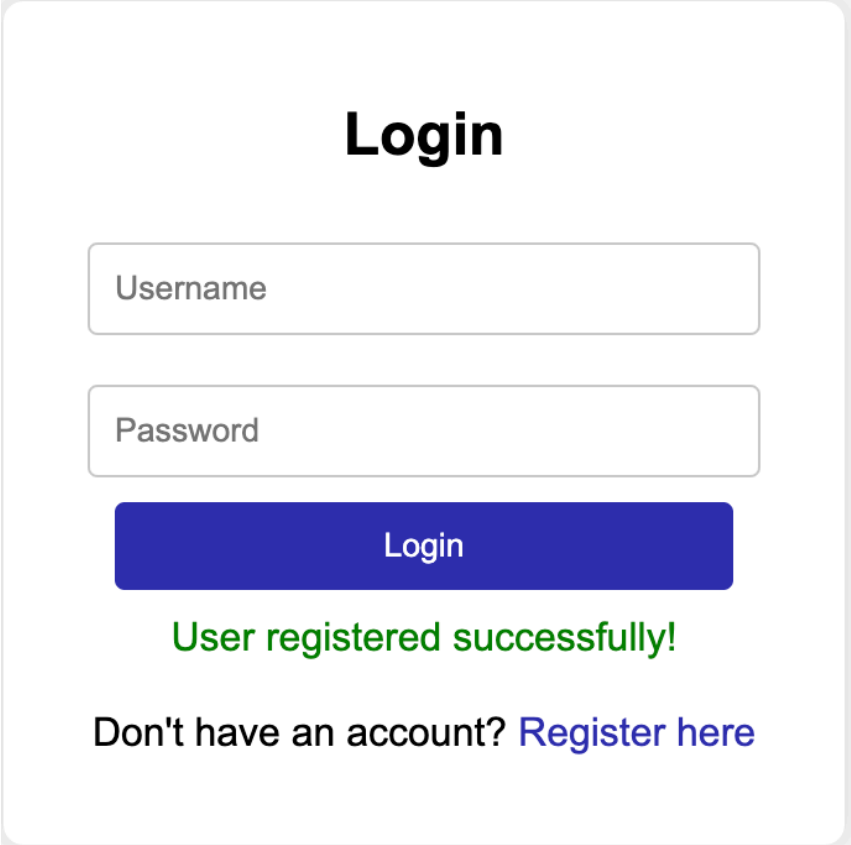
Website

It should automatically redirect you to the login page.

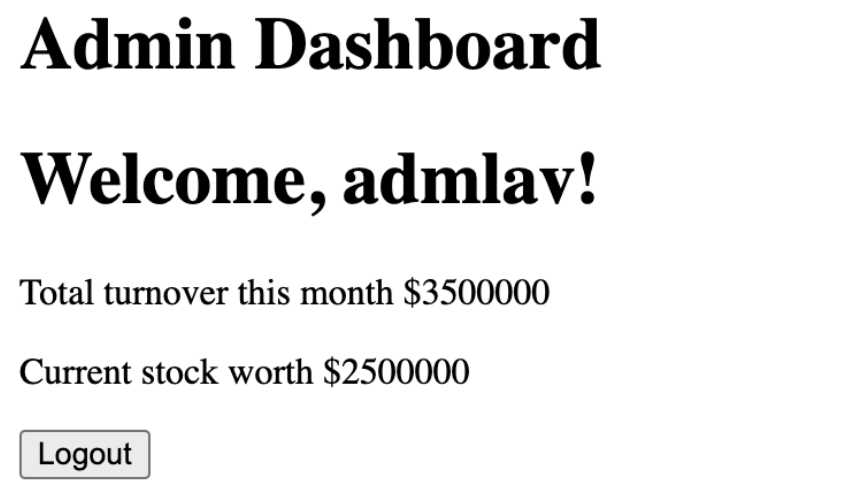
4. First you need to register a user to login. Click the `Register Here` link. This will take you to the register page.



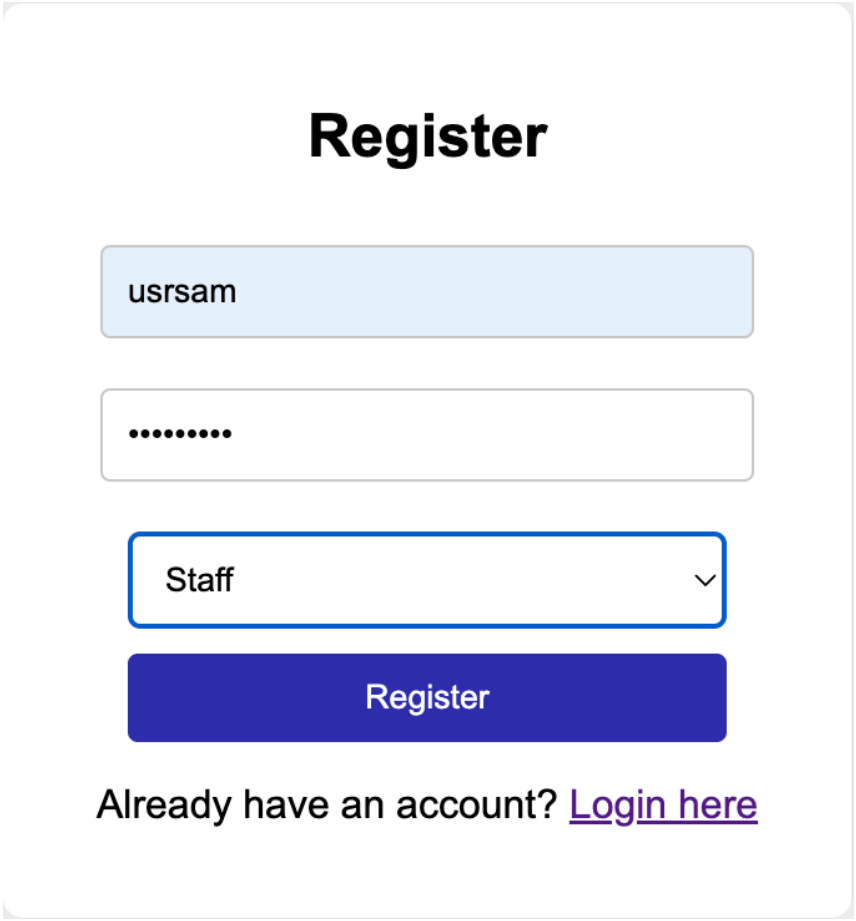
5. Enter the user login, password, and from the drop-down, select the role and select `Register`. Once the user registers, the registration is confirmed and login page is displayed.



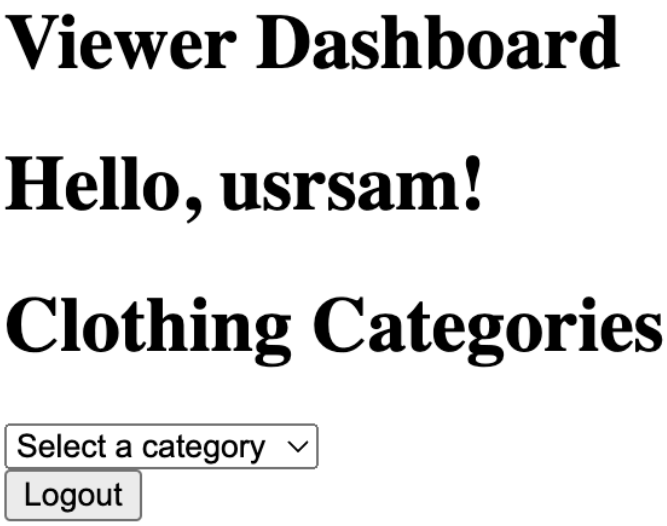
6. Log in with the username and password you registered with. Once the login is successful, if you used the user registered as admin, the admin dashboard is displayed.



7. Logout and register as staff user.



8. Once the user is registered successfully, login with the user credentials and see that a different home page is displayed for non-admin user.



Practice exercise

1. Add `First Name`, `Lastname`, and `Email` fields in the registration form and optionally add validation to the data.
2. Make changes to the `model/User`, `service/CustomUserDetailsService`, and `config/WebSecurityConfig` to consider the additional user attributes.
3. Display the `First Name` in `viewer.html` with a casual greeting.
4. Display the `First Name` in `admin.html` with a formal greeting and showing the current date and time.
5. Run and test the application.

Conclusion

In this lab, you have:

- Cloned the Spring MVC app with security, import it in the workspace, and tested it
- Modified the `User` model to handle additional attributes
- Modified the security configuration to allow authentication and authorization
- Created role specific webpages for the controller to render
- Tested the application end to end for authentication and authorization

Author(s)

Lavanya