

Spring Web Development

Objectives

- ◆ Building web applications using the Spring Framework.
 - Spring MVC (Model-View-Controller)
 - Spring WebFlux
 - Spring Boot
 - RESTful Web Services
 - Security
 - Integration with other technologies
- ◆ Understand a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text (Thymeleaf)

Spring Web Development Introduction

- ◆ The Spring Framework offers a robust and flexible platform for developing web applications, with comprehensive support for integration, scalability, security, and testing.
- ◆ With Spring Framework, a strong community support and alignment with modern architectural patterns make it a compelling choice for building modern and reliable web applications.

Spring Web Development Introduction

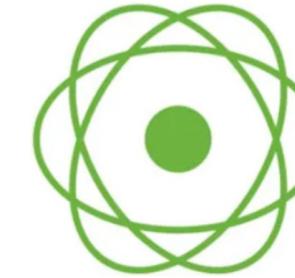
- ◆ Spring provides a comprehensive and flexible platform for building web applications, whether you're using traditional MVC patterns or adopting newer reactive programming paradigms.
 - Spring MVC (Model-View-Controller)
 - Spring WebFlux
 - Spring Boot
 - RESTful Web Services
 - Security
 - Integration with other technologies
 - Testing

Spring MVC (Model-View-Controller)

- ◆ Spring MVC is the traditional approach to building web applications with Spring.
- ◆ It provides a robust architecture for organizing your web application into components such as controllers, views, and models.
- ◆ Controllers handle incoming requests, models represent the data, and views render the response.



Spring WebFlux



- ◆ **Spring WebFlux** is the reactive approach introduced in Spring 5.
- ◆ It's based on Project Reactor and the Reactive Streams specification.
- ◆ WebFlux allows you to build non-blocking, reactive web applications, which are particularly useful for handling high concurrency and streaming data scenarios.
- ◆ The reactive-stack web framework, Spring WebFlux is fully non-blocking, supports Reactive Streams (<https://www.reactive-streams.org/>) back pressure, and runs on such servers as Netty, Undertow, and Servlet containers.

Spring Boot

- While not exclusively for web development, Spring Boot greatly simplifies the process of creating stand-alone, production-grade Spring-based applications.
- Consider using Spring Boot to quickly bootstrap your Spring applications.
- Spring Boot offers a convention-over-configuration approach, simplifying the setup and configuration of Spring-based applications.
- Spring Boot is an excellent choice for rapidly developing web applications with minimal setup and boilerplate code.



Spring Security

- ◆ Security: Spring Security is the de facto standard for securing Spring-based applications.
- ◆ It provides comprehensive security features for web applications, including authentication, authorization, session management, and more.
- ◆ It supports both traditional server-rendered applications (like Spring MVC) and modern single-page applications (like those built with Angular or React).



Spring REST

- ◆ If your web applications require RESTful services, consider using Spring's support for building REST APIs.
- ◆ Spring's REST capabilities, combined with Spring MVC or Spring WebFlux, provide powerful tools for creating and consuming RESTful services.
- ◆ Spring MVC and Spring WebFlux can both be used to create RESTful services, with features like content negotiation, request mapping, and input validation.

Integration with other technologies

- Spring integrates seamlessly with other technologies commonly used in web development, such as databases (via Spring Data), messaging systems (via Spring Messaging), and frontend frameworks (via RESTful APIs or WebSocket support).

Spring Data

- ◆ For data access, Spring Data offers a range of options, including JPA, MongoDB, Redis, and more.
- ◆ Choose the appropriate Spring Data module based on your data storage requirements to simplify data access and manipulation within your web applications.

Templating engines

- When it comes to templating engines, consider using Thymeleaf or Freemarker.
- Both provide powerful template processing capabilities, enabling the seamless integration of dynamic content into web pages.
- Thymeleaf, in particular, is a popular choice for Spring-based web applications due to its natural integration with Spring MVC.

Thymeleaf - a Java-based server side template engine

What is Thymeleaf?

- ◆ Thymeleaf is a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text.
- ◆ The main goal of Thymeleaf is to provide an elegant and highly-maintainable way of creating templates.
- ◆ Thymeleaf improves communication of design and bridges the gap between design and development teams.
- ◆ Thymeleaf has also been designed from the beginning with Web Standards in mind - especially HTML5 - allowing you to create fully validating templates if ◆ that is a need for you

Kinds of templates in Thymeleaf

- There are two markup template modes (HTML and XML), three textual template modes (TEXT , JAVASCRIPT and CSS) and a no-op template mode (RAW).
- The **HTML template** mode will allow any kind of HTML input, including HTML5, HTML 4 and XHTML. No validation or well-formedness check will be performed, and template code/structure will be respected to the biggest possible extent in output.
- The **XML template** mode will allow XML input. In this case, code is expected to be well-formed – no unclosed tags, no unquoted attributes, etc – and the parser will throw exceptions if well-formedness violations are found. Note that no validation (against a DTD or XML Schema) will be performed.

Kinds of templates in Thymeleaf

- The **TEXT template** mode will allow the use of a special syntax for templates of a non-markup nature. Examples of such templates might be text emails or templated documentation. Note that HTML or XML templates can be also processed as TEXT , in which case they will not be parsed as markup, and every tag, DOCTYPE, comment, etc, will be treated as mere text.
- The **JAVASCRIPT template** mode will allow the processing of JavaScript files in a Thymeleaf application. This means being able to use model data inside JavaScript files in the same way it can be done in HTML files, but with JavaScript-specific integrations such as specialized escaping or natural scripting. The JAVASCRIPT template mode is considered a textual mode and therefore uses the same special syntax as the TEXT template mode.

Kinds of templates in Thymeleaf

- The CSS template mode will allow the processing of CSS files involved in a Thymeleaf application. Similar to the JAVASCRIPT mode, the CSS template mode is also a textual mode and uses the special processing syntax from the TEXT template mode.
- The RAW template mode will simply not process templates at all. It is meant to be used for inserting untouched resources (files, URL responses, etc.) into the templates being processed. For example, external, uncontrolled resources in HTML format could be included into application templates, safely knowing that any Thymeleaf code that these resources might include will not be executed.

Standard Expression Syntax

- ◆ Simple expressions:
 - Variable Expressions: \${...}
 - Selection Variable Expressions: *{...}
 - Message Expressions: #{...}
 - Link URL Expressions: @{...}
 - Fragment Expressions: ~{...}
- ◆ Literals
 - Text literals: 'one text' , 'Another one!' ,...
 - Number literals: 0 , 34 , 3.0 , 12.3 ,...
 - Boolean literals: true , false
 - Null literal: null
 - Literal tokens: one , sometext , main ,...

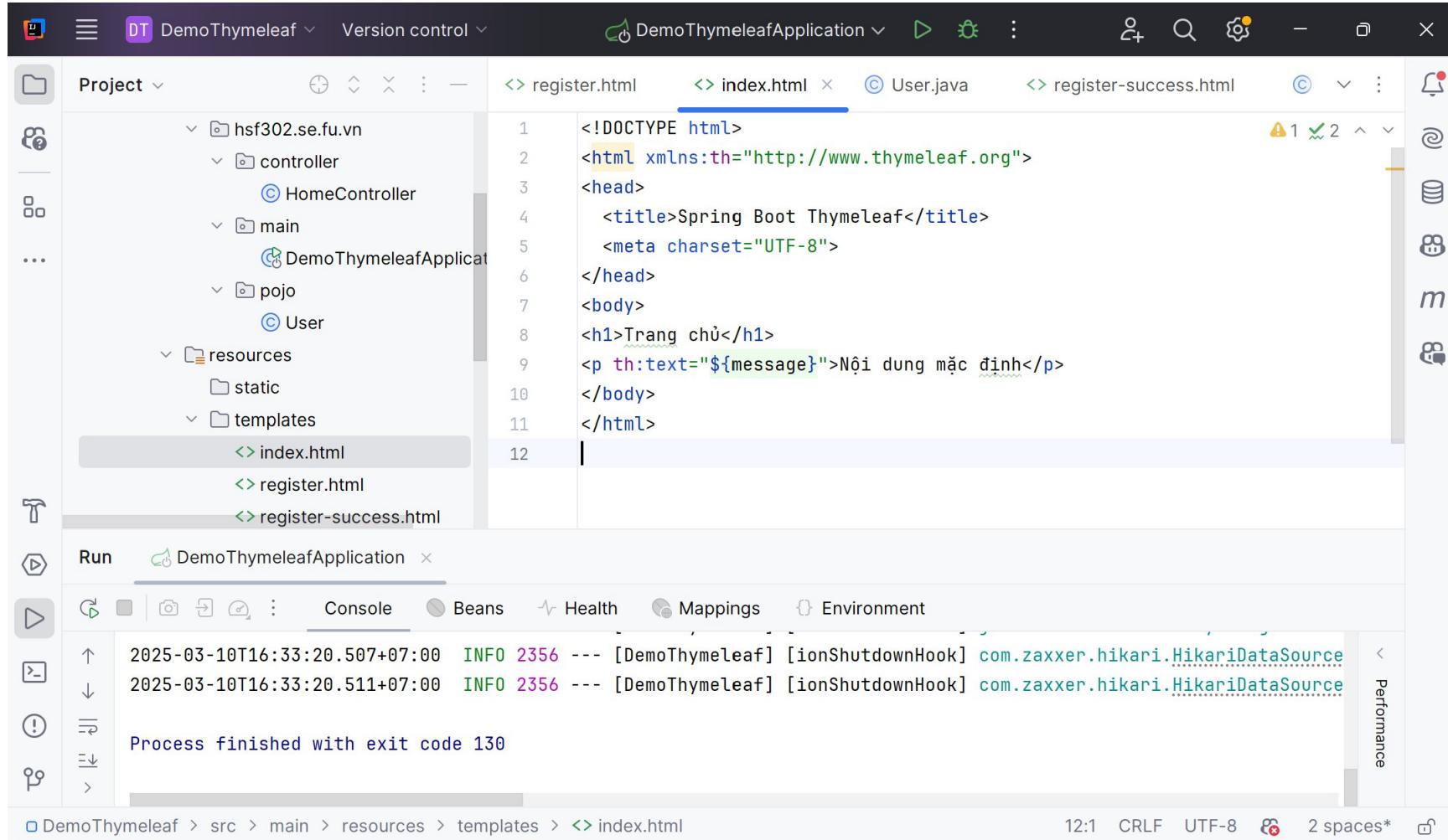
Standard Expression Syntax

- ◆ Text operations:
 - String concatenation: +
 - Literal substitutions: |The name is \${name}|
- ◆ Arithmetic operations:
 - Binary operators: + , - , * , / , %
 - Minus sign (unary operator): -
- ◆ Boolean operations:
 - Binary operators: and , or
 - Boolean negation (unary operator): ! , not

Standard Expression Syntax

- ◆ Comparisons and equality:
 - Comparators: > , < , >= , <= (gt , lt , ge , le)
 - Equality operators: == , != (eq , ne)
- ◆ Conditional operators:
 - If-then: (if) ? (then)
 - If-then-else: (if) ? (then) : (else)
- ◆ Default: (value) ?: (defaultvalue)
- ◆ Special tokens:
 - No-Operation: _

HTML Template Example



The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "DemoThymeleaf". It contains a package "hsf302.se.fu.vn" with sub-directories "controller", "main", "pojo", and "resources". The "resources" directory contains a "static" folder and a "templates" folder which holds "index.html", "register.html", and "register-success.html".
- Code Editor:** The file "index.html" is open. Its content is as follows:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Spring Boot Thymeleaf</title>
    <meta charset="UTF-8">
</head>
<body>
    <h1>Trang chủ</h1>
    <p th:text="${message}">Nội dung mặc định</p>
</body>
</html>
```

- Run Tab:** The application "DemoThymeleafApplication" is selected.
- Console Tab:** Shows log messages:

```
2025-03-10T16:33:20.507+07:00 INFO 2356 --- [DemoThymeleaf] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource
2025-03-10T16:33:20.511+07:00 INFO 2356 --- [DemoThymeleaf] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource
Process finished with exit code 130
```
- Bottom Status Bar:** Displays the path "DemoThymeleaf > src > main > resources > templates > index.html", and file statistics: 12:1 CRLF, UTF-8, 2 spaces*.

Attribute Precedence

Order	Feature	Attributes
1	Fragment inclusion	th:insert th:replace
2	Fragment iteration	th:each
3	Conditional evaluation	th:if th:unless th:switch th:case
4	Local variable definition	th:object th:with
5	General attribute modification	th:attr th:attrprepend th:attrappend
6	Specific attribute modification	th:value th:href th:src ...
7	Text (tag body modification)	th:text th:utext
8	Fragment specification	th:fragment
9	Fragment removal	th:remove

<https://www.thymeleaf.org/doc/articles/layouts.html>

Form Attributes

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Register</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css">
    <link rel="stylesheet" th:href="@{/css/chapter06.css}">
</head>
<body>
<div class="container mt-5">
    <div class="row">
        <div class="col-md-6 offset-md-3">
            <div class="card">
                <div class="card-header bg-primary text-white">
                    <h3 class="text-center">Register</h3>
                </div>
                <div class="card-body">
                    <form th:action="@{/register}" th:object="${user}" method="post">
                        <div class="mb-3">
                            <label for="firstName" class="form-label">First Name</label>
                            <input type="text" th:field="*{firstName}" id="firstName" class="form-control" placeholder="First Name">
                            <span th:if="${#fields.hasErrors('firstName')}" th:errors="*{firstName}" class="error"></span>
                        </div>

                        <div class="mb-3">
                            <label for="lastName" class="form-label">Last Name</label>
                            <input type="text" th:field="*{lastName}" id="lastName" class="form-control" placeholder="Last Name">
                            <span th:if="${#fields.hasErrors('lastName')}" th:errors="*{lastName}" class="error"></span>
                        </div>

                        <div class="mb-3">
                            <label for="email" class="form-label">Email</label>
                            <input type="email" th:field="*{email}" id="email" class="form-control" placeholder="Email">
                            <span th:if="${#fields.hasErrors('email')}" th:errors="*{email}" class="error"></span>
                        </div>

                        <div class="mb-3">
                            <label for="password" class="form-label">Password</label>
                            <input type="password" th:field="*{password}" id="password" class="form-control" placeholder="Password">
                            <span th:if="${#fields.hasErrors('password')}" th:errors="*{password}" class="error"></span>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
```

2

Add CSS and href tag

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>User List</title>

    <link rel="stylesheet" th:href="@{/css/chapter06.css}">

</head>
<body>
<h1>User List</h1>
<div>
    <a th:href="@{/register}" class="btn-create">Add user</a>
</div>
```

Condition and Loop Statement



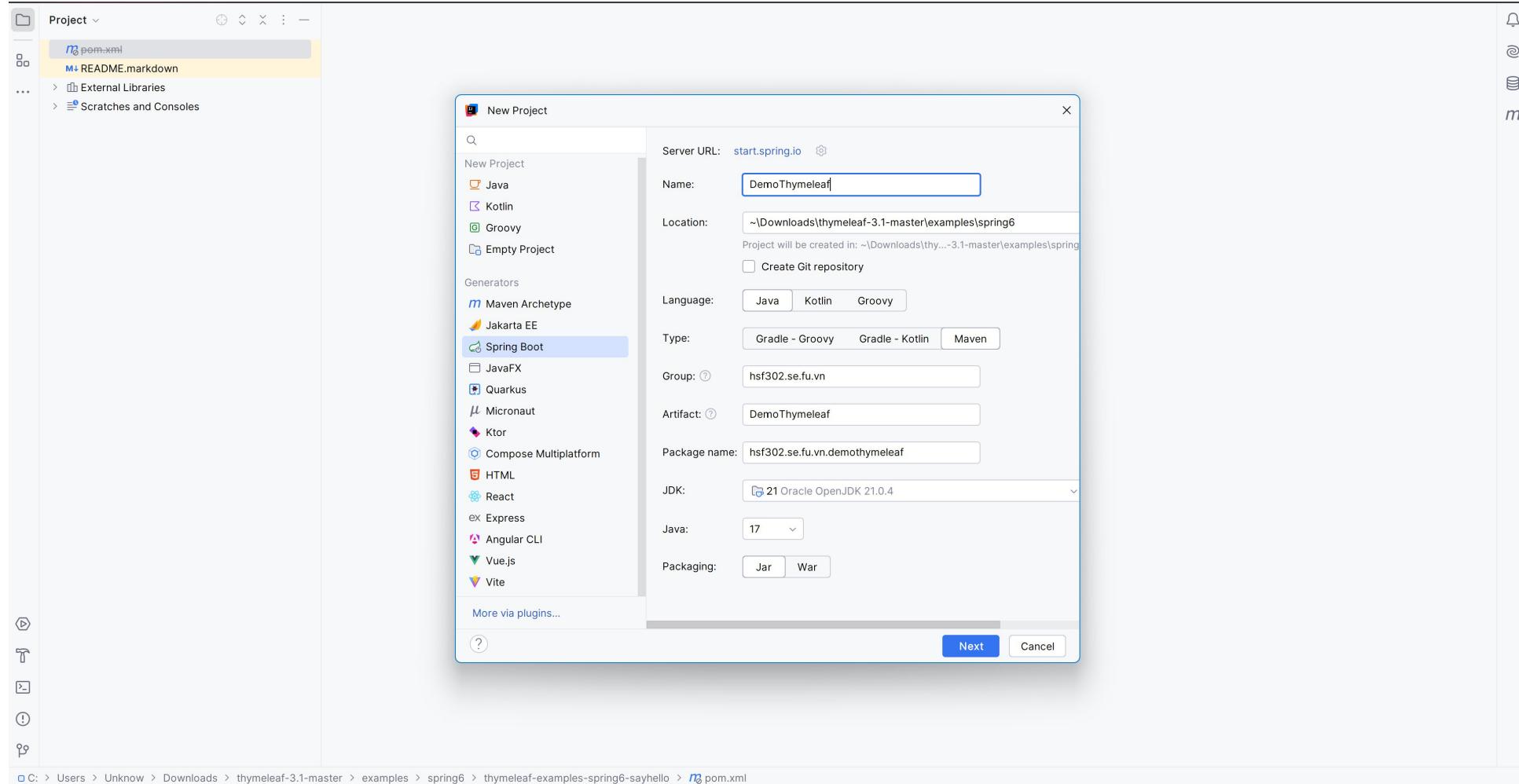
```
<table>
    <thead>
        <tr>
            <th>Province/City</th>
            <th>Date of Birth</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>

        <tr th:each="user, status : ${users}">
            <td th:text="${status.count}">1</td>
            <td th:text ="${user.firstName + ' ' + user.lastName}">Full Name</td>
            <td th:text ="${user.email}">Email</td>
            <td th:text ="${user.gender}">Gender</td>
            <td>
                <ul class="hobby-list">
                    <li th:each="hobby : ${user.hobbies}" th:text ="${hobby}">Hobbies</li>
                </ul>
                <span th:if ="${#lists.isEmpty(user.hobbies)}">Non</span>
            </td>
            <td th:text ="${user.province}">City</td>
            <td th:text ="${#dates.format(user.birthday, 'dd/MM/yyyy')}">Birthday</td>
            <td>
                <a th:href ="@{/users/{email}/edit(email=${user.email})}" class="btn-edit">Edit</a>
                <a th:href ="@{/users/{email}/delete(email=${user.email})}" class="btn-delete"
                   onclick="return confirm('Are you sure ?')">Delete</a>
            </td>
        </tr>

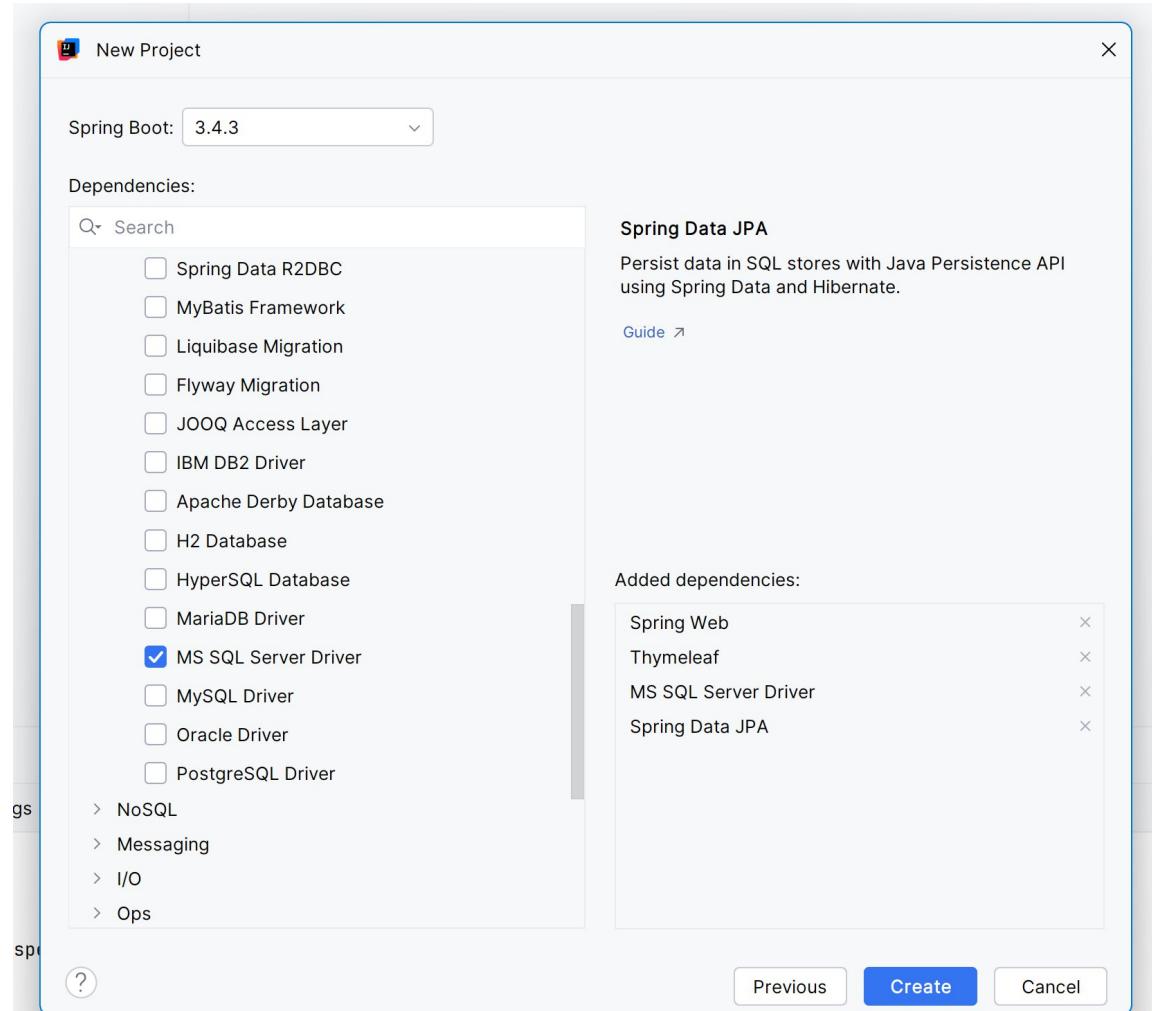
        <!-- Hiển thị thông báo nếu danh sách trống -->
        <tr th:if ="${#lists.isEmpty(users)}">
            <td colspan="8">Empty</td>
        </tr>
    </tbody>
</table>
```

Demo Spring Boot with Thymeleaf Engine

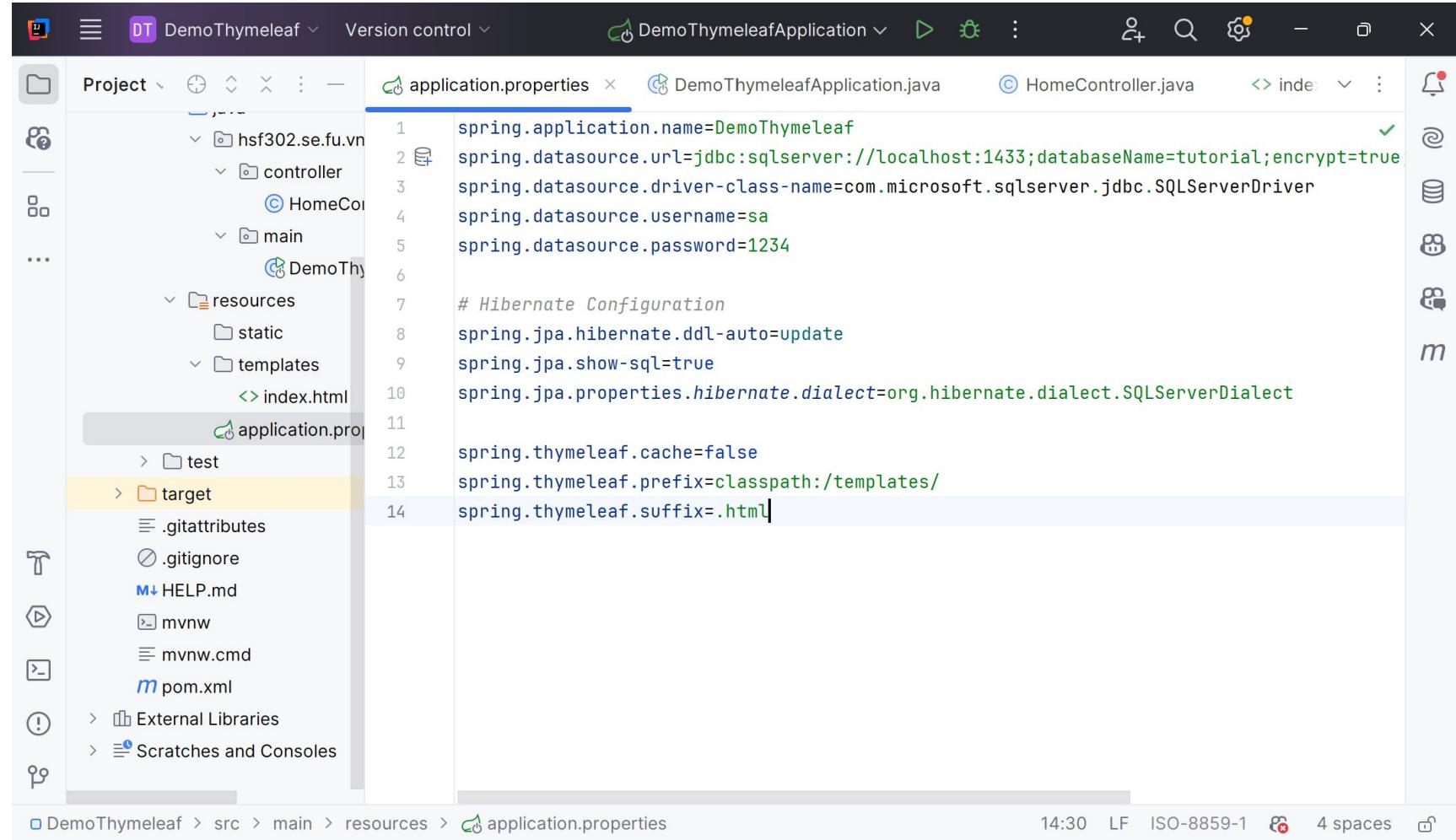
Create a simple project using Thymeleaf



Add dependencies



Update the application.properties



The screenshot shows a code editor interface with the following details:

- Project:** DemoThymeleaf
- File:** application.properties
- Content:**

```
spring.application.name=DemoThymeleaf
spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=tutorial;encrypt=true
spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver
spring.datasource.username=sa
spring.datasource.password=1234

# Hibernate Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect

spring.thymeleaf.cache=false
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
```

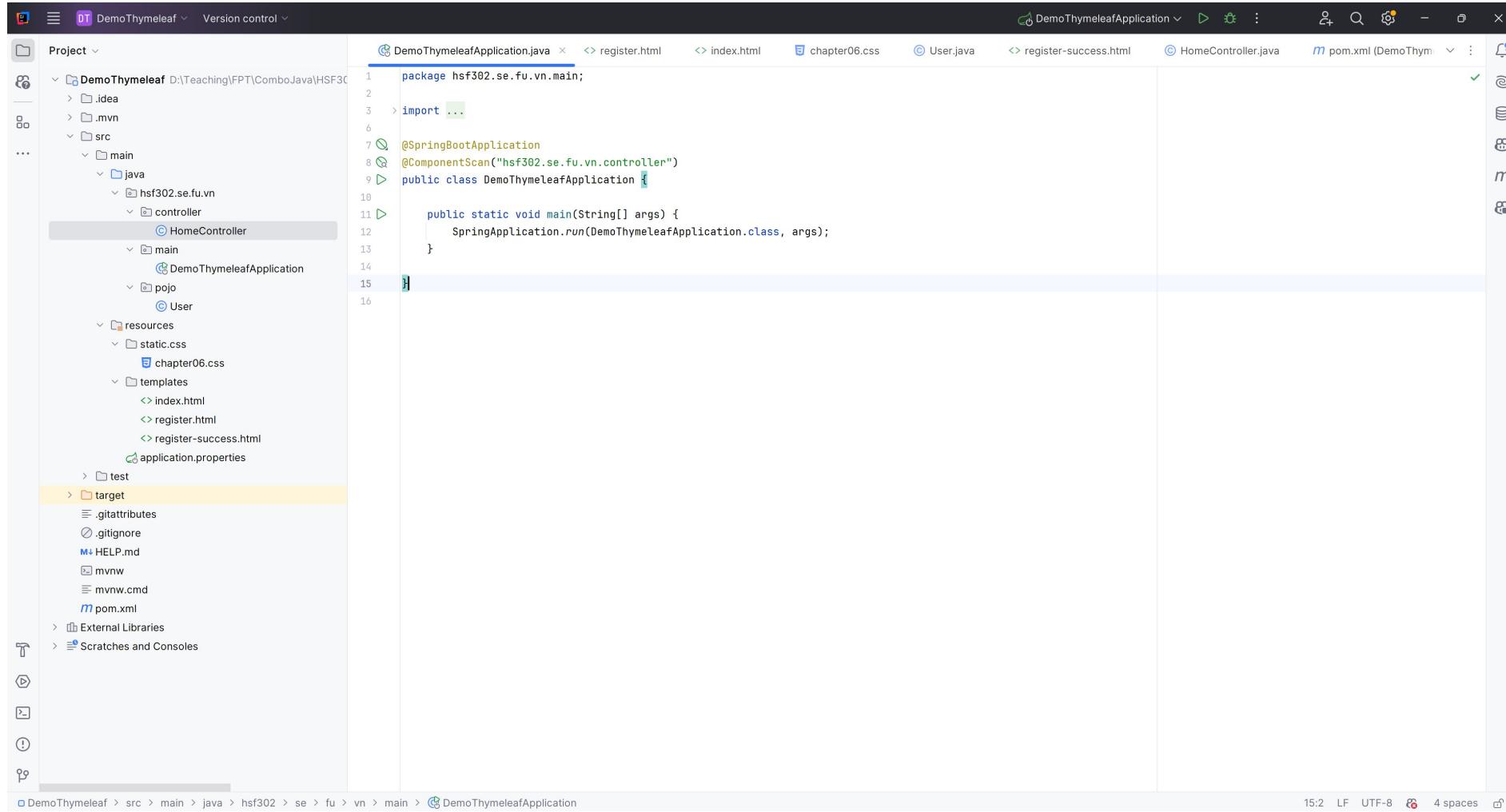
The code editor has a dark theme with syntax highlighting for Java and properties files. The sidebar shows the project structure with a focus on the target folder.

Update pom.xml



```
<dependencies> ⚡ Add Starters...
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</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.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
```

Create the Structure Project

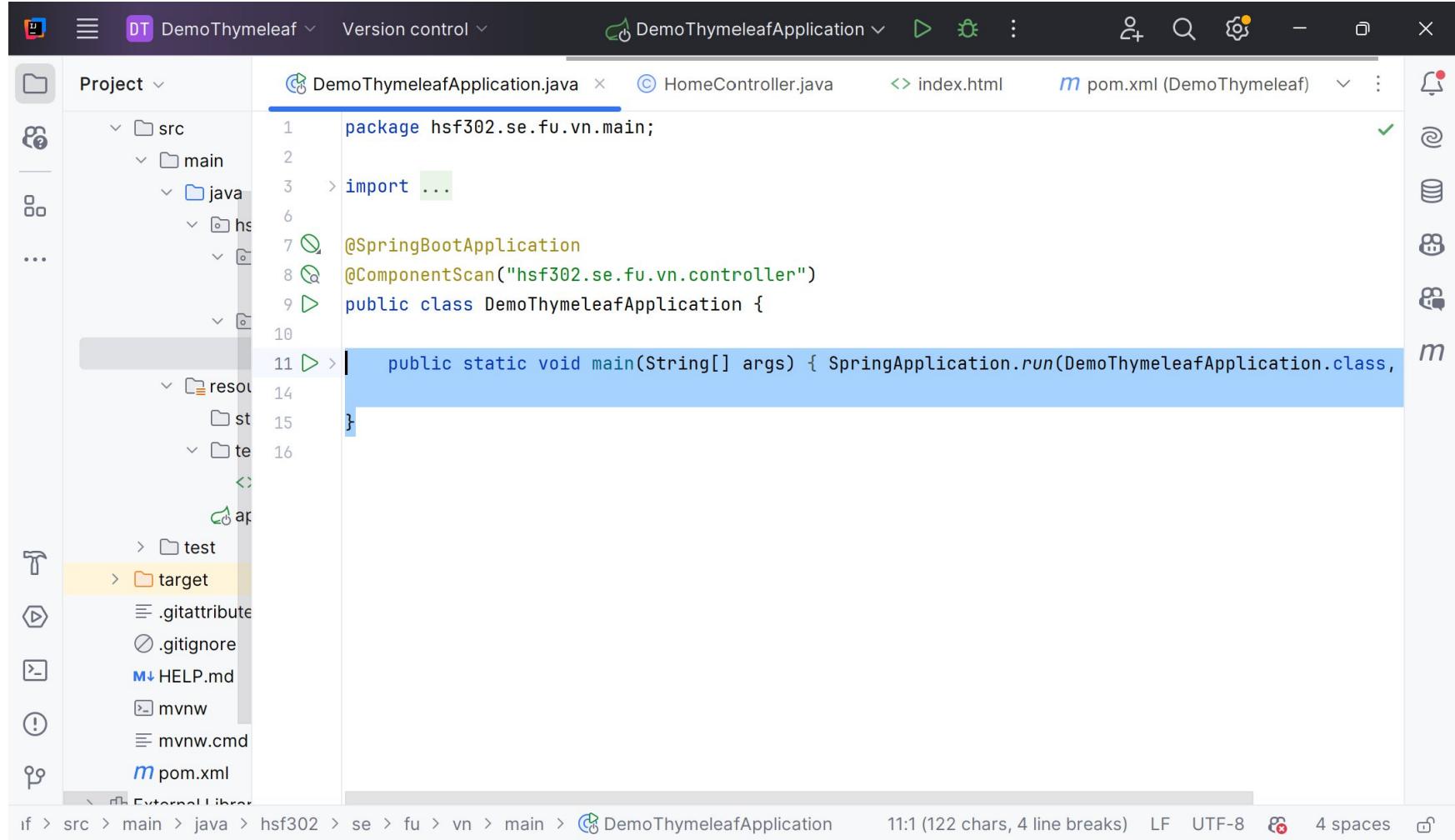


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure is displayed under "DemoThymeleaf". It includes the following packages:
 - src:** Contains main, resources, and test.
 - main:** Contains java, pojo, and templates.
 - java:** Contains hsf302.se.fu.vn.
 - controller:** Contains HomeController.
 - main:** Contains DemoThymeleafApplication.
 - pojo:** Contains User.
 - resources:** Contains static.css, chapter06.css, and application.properties.
 - templates:** Contains index.html, register.html, and register-success.html.
 - Code Editor:** The central window shows the `DemoThymeleafApplication.java` file with the following code:

```
1 package hsf302.se.fu.vn.main;
2
3 import ...
4
5 @SpringBootApplication
6 @ComponentScan("hsf302.se.fu.vn.controller")
7 public class DemoThymeleafApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DemoThymeleafApplication.class, args);
11     }
12 }
```
 - Toolbars and Status Bar:** The top bar shows tabs for other files like register.html, index.html, chapter06.css, etc. The status bar at the bottom right shows "15:2 LF UTF-8 4 spaces".

Update main function



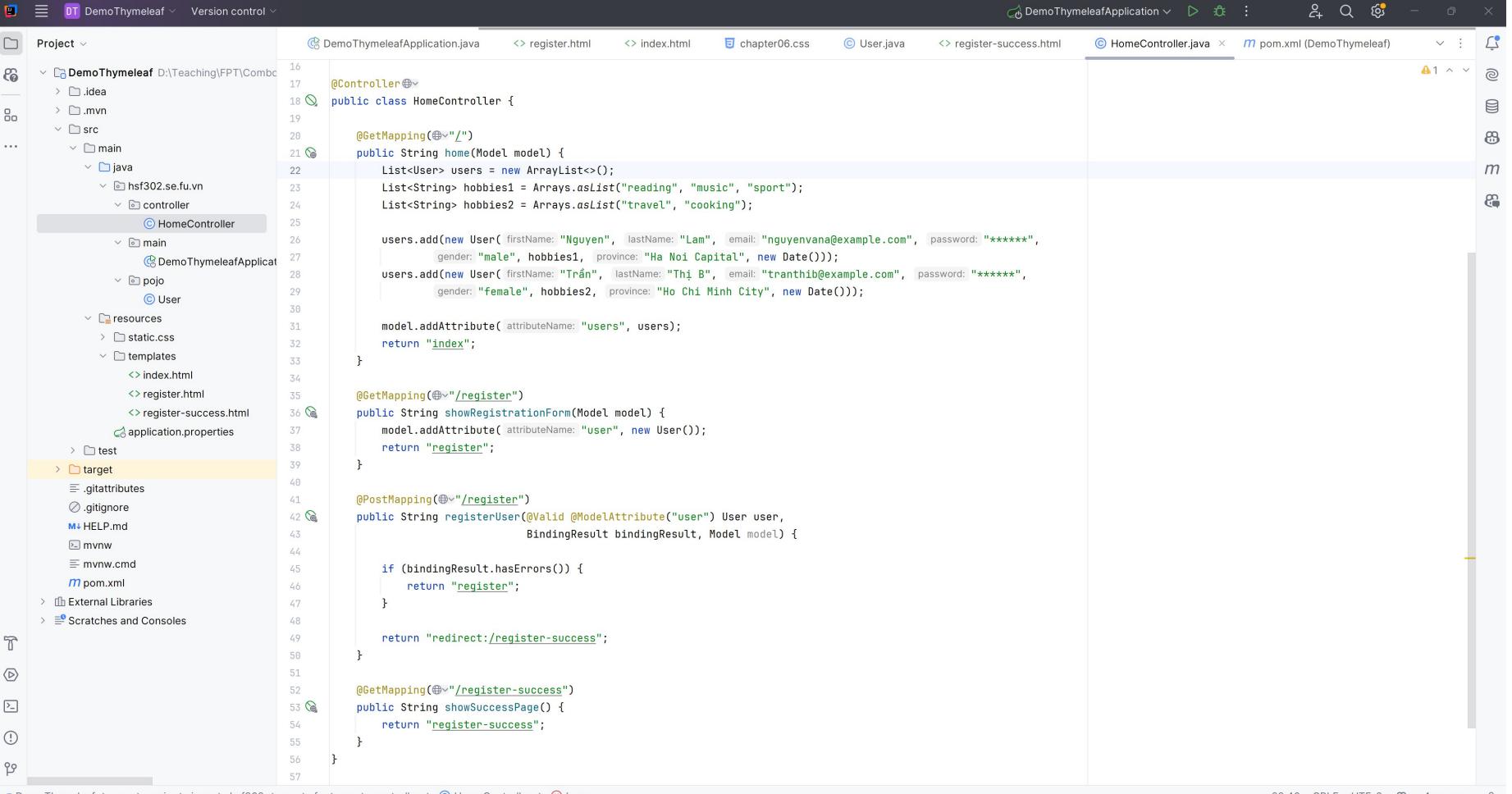
```
package hsf302.se.fu.vn.main;
import ...
@SpringBootApplication
@ComponentScan("hsf302.se.fu.vn.controller")
public class DemoThymeleafApplication {
    public static void main(String[] args) { SpringApplication.run(DemoThymeleafApplication.class,
    }
}
```

The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "DemoThymeleaf". The source code is located in the "src/main/java" directory, specifically under the package "hsf302.se.fu.vn.main".
- Files:** The current file is "DemoThymeleafApplication.java", which contains the main application class definition.
- Code View:** The code editor displays the main function definition:

```
public static void main(String[] args) { SpringApplication.run(DemoThymeleafApplication.class,
```
- IDE Interface:** The interface includes standard tabs for "DemoThymeleafApplication.java", "HomeController.java", "index.html", and "pom.xml". The status bar at the bottom shows the file path, character count (11:1 (122 chars, 4 line breaks)), encoding (LF), and indentation settings (4 spaces).

Controller

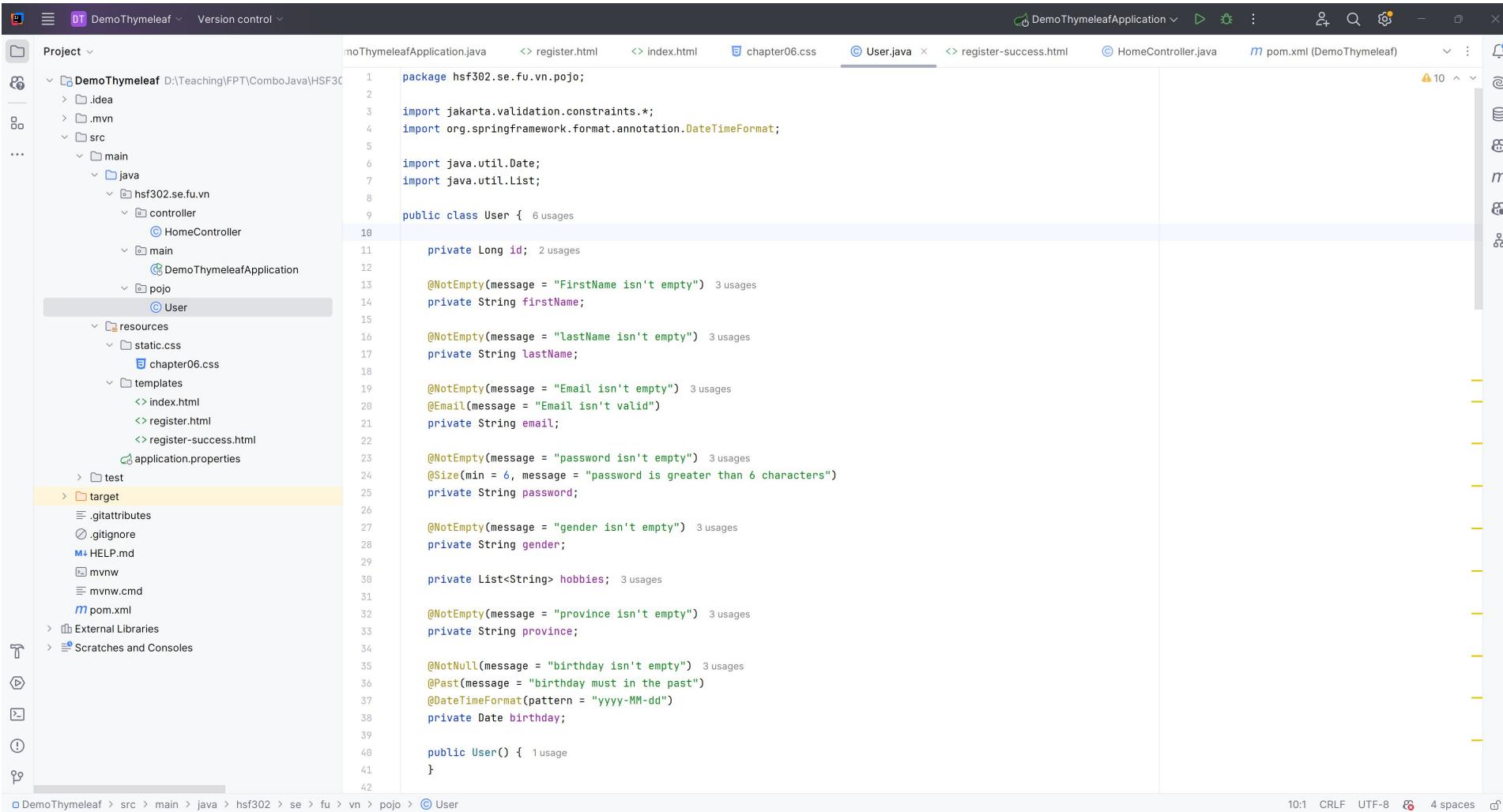


The screenshot shows the IntelliJ IDEA interface with the HomeController.java file open in the central editor window. The code implements a Spring MVC controller with methods for home, register, registerUser, and registerSuccess.

```
16  @Controller
17  public class HomeController {
18
19      @GetMapping("/")
20      public String home(Model model) {
21          List<User> users = new ArrayList<>();
22          List<String> hobbies1 = Arrays.asList("reading", "music", "sport");
23          List<String> hobbies2 = Arrays.asList("travel", "cooking");
24
25          users.add(new User( firstName: "Nguyen", lastName: "Lam", email: "nguyenvana@example.com", password: "*****",
26          | gender: "male", hobbies1, province: "Ha Noi Capital", new Date());
27          users.add(new User( firstName: "Trần", lastName: "Thị B", email: "tranthib@example.com", password: "*****",
28          | gender: "female", hobbies2, province: "Ho Chi Minh City", new Date()));
29
30          model.addAttribute( attributeName: "users", users);
31          return "index";
32      }
33
34
35      @GetMapping("/register")
36      public String showRegistrationForm(Model model) {
37          model.addAttribute( attributeName: "user", new User());
38          return "register";
39      }
34
35
36      @PostMapping("/register")
37      public String registerUser(@Valid @ModelAttribute("user") User user,
38          BindingResult bindingResult, Model model) {
39
40          if (bindingResult.hasErrors()) {
41              return "register";
42          }
43
44          return "redirect:/register-success";
45      }
46
47
48      @GetMapping("/register-success")
49      public String showSuccessPage() {
50          return "register-success";
51      }
52
53
54
55
56
57 }
```

The left sidebar shows the project structure with files like DemoThymeleafApplication.java, User.java, and various HTML templates. The bottom status bar indicates the file path as DemoThymeleaf > src > main > java > hsf302 > se > fu > vn > controller > HomeController > home, and the file is saved at 22:46 CRLF UTF-8 4 spaces.

Pojo and validation

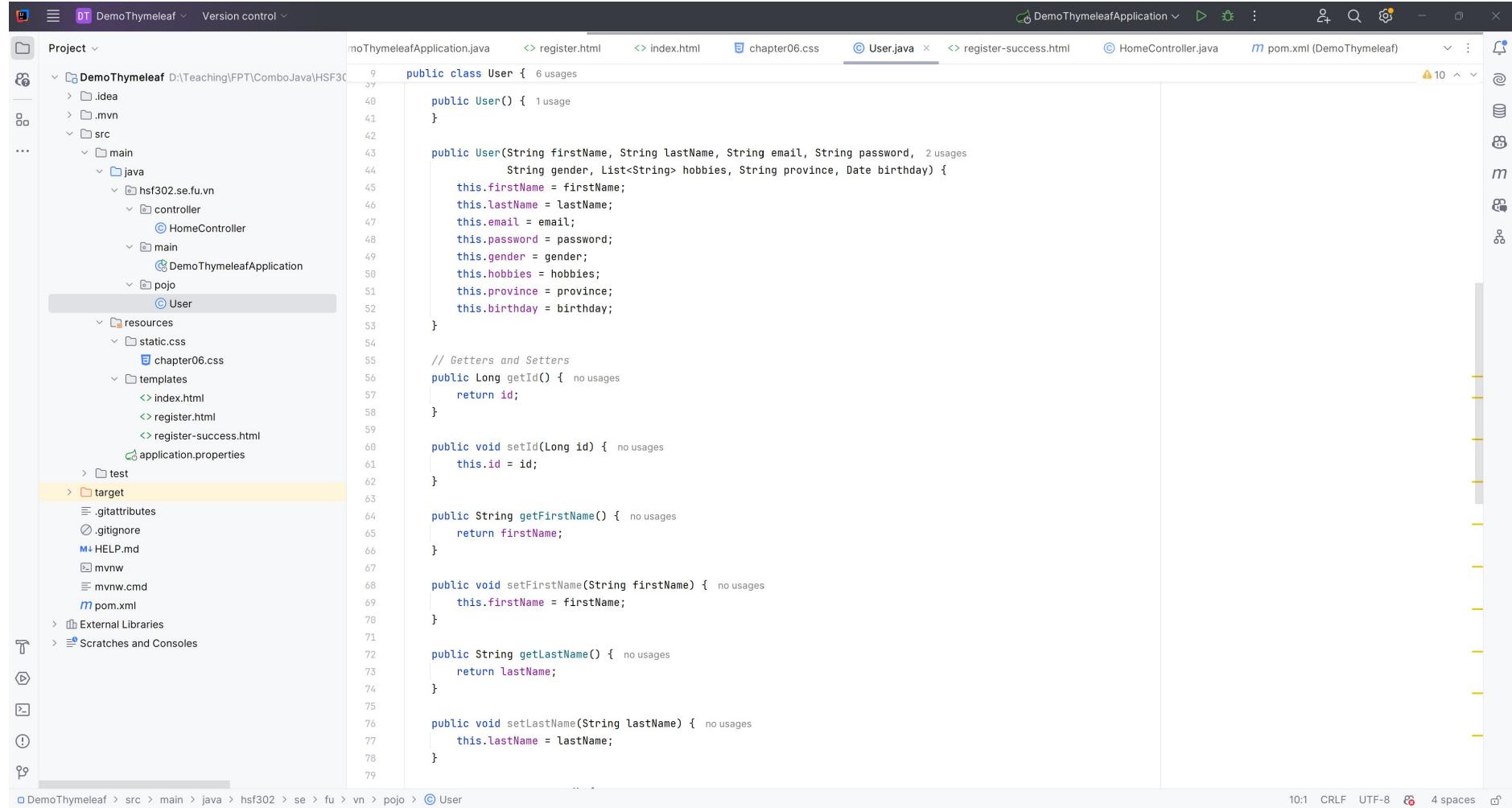


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "DemoThymeleaf". It contains a "src" directory with "main" and "test" packages. The "main" package has a "java" folder containing "hsf302.se.fu.vn" and "DemoThymeleafApplication". "hsf302.se.fu.vn" contains a "controller" folder with "HomeController" and a "pojo" folder with "User". The "pojo" folder is selected.
- User.java Content:**

```
1 package hsf302.se.fu.vn.pojo;
2
3 import jakarta.validation.constraints.*;
4 import org.springframework.format.annotation.DateTimeFormat;
5
6 import java.util.Date;
7 import java.util.List;
8
9 public class User { 6 usages
10     private Long id; 2 usages
11
12     @NotEmpty(message = "FirstName isn't empty") 3 usages
13     private String firstName;
14
15     @NotEmpty(message = "lastName isn't empty") 3 usages
16     private String lastName;
17
18     @NotEmpty(message = "Email isn't empty") 3 usages
19     @Email(message = "Email isn't valid")
20     private String email;
21
22     @NotEmpty(message = "password isn't empty") 3 usages
23     @Size(min = 6, message = "password is greater than 6 characters")
24     private String password;
25
26     @NotEmpty(message = "gender isn't empty") 3 usages
27     private String gender;
28
29     private List<String> hobbies; 3 usages
30
31     @NotEmpty(message = "province isn't empty") 3 usages
32     private String province;
33
34     @NotNull(message = "birthday isn't empty") 3 usages
35     @Past(message = "birthday must in the past")
36     @DateTimeFormat(pattern = "yyyy-MM-dd")
37     private Date birthday;
38
39     public User() { 1 usage
40     }
41
42 }
```
- Toolbars and Status Bar:** The status bar at the bottom shows "10:1 CRLF UTF-8 4 spaces".

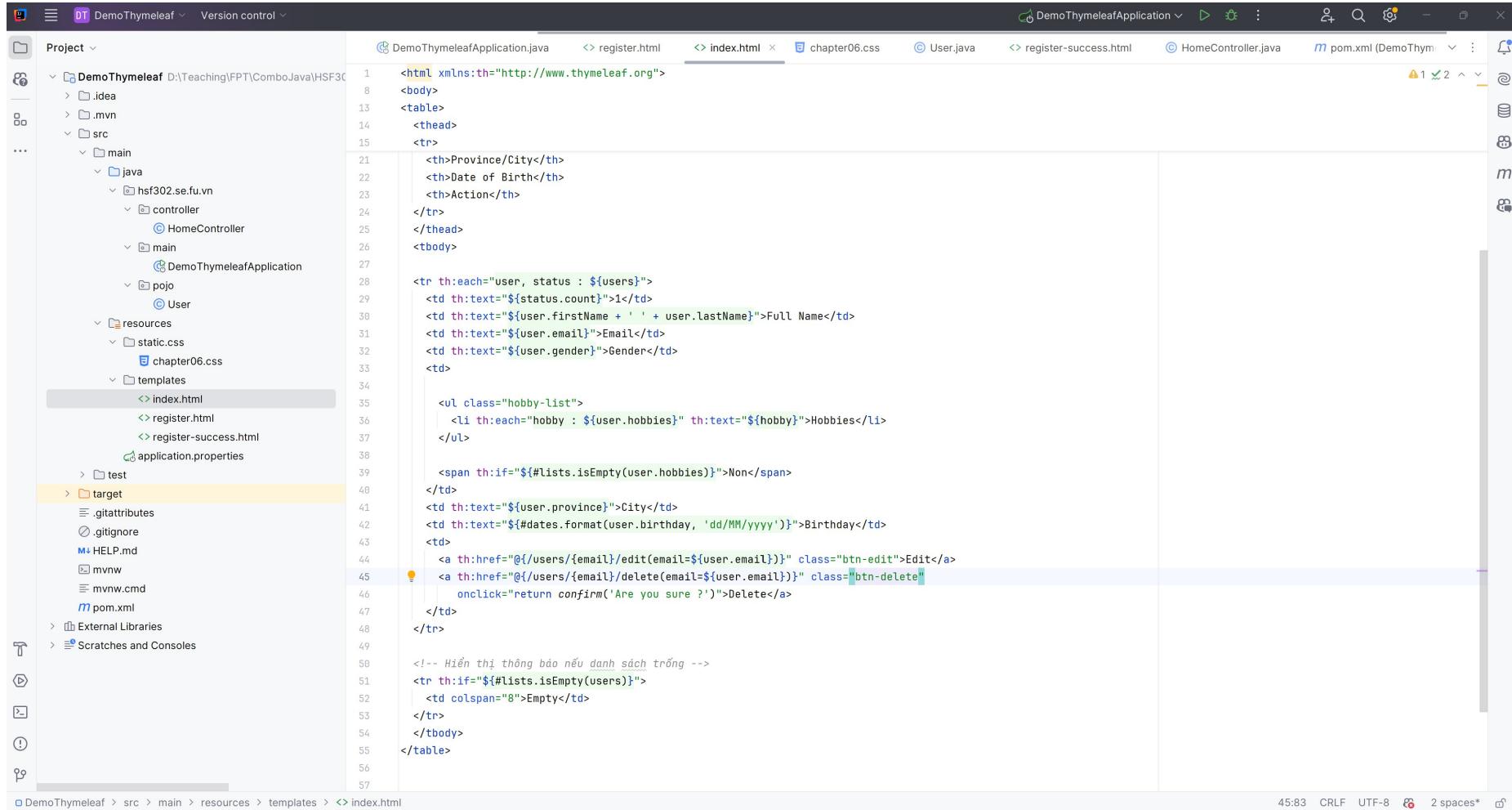
Pojo and validation



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "DemoThymeleaf". The "User.java" file is selected in the "src/main/java" directory.
- User.java Code:** The main content area displays the code for the User class. The code includes a constructor that takes firstName, lastName, email, password, gender, hobbies, province, and birthday as parameters and initializes them. It also includes standard getters and setters for each field.
- Toolbars and Status Bar:** The top bar includes tabs for other files like "moThymeleafApplication.java", "register.html", "index.html", "chapter06.css", "HomeController.java", and "pom.xml". The status bar at the bottom shows "10:1 CRLF UTF-8 4 spaces".

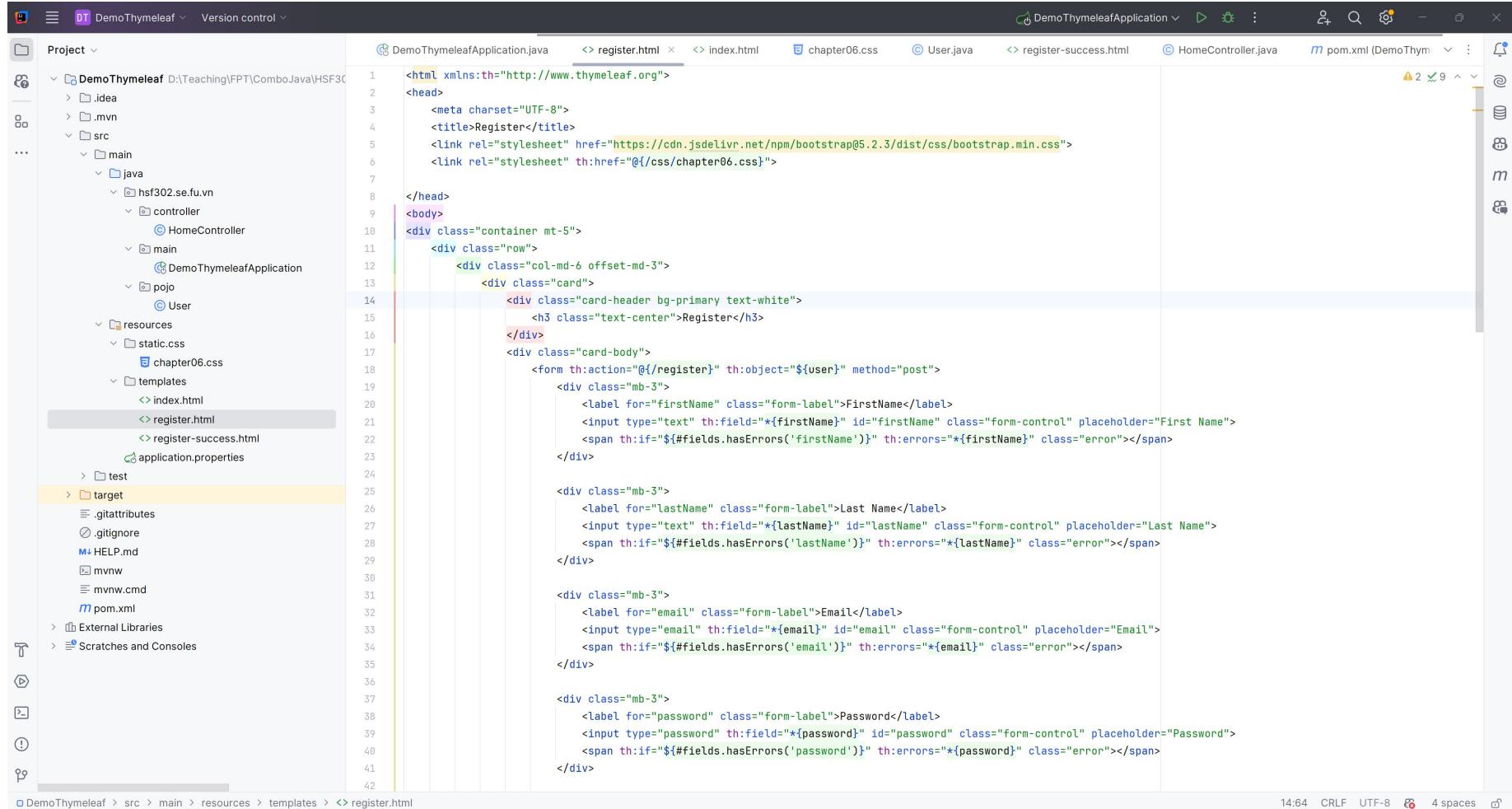
Create index.html



The screenshot shows the IntelliJ IDEA interface with the project navigation bar at the top. The left sidebar displays the project structure under 'Project'. The main editor area shows the 'index.html' file, which contains Thymeleaf template code. The code includes a table with columns for Province/City, Date of Birth, and Action. It iterates over a list of users, displaying their full name, email, gender, and hobbies. It also handles the deletion of users by providing edit and delete links with confirmation dialogs. A note at the bottom indicates that it will show a message if the list is empty.

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
<table>
<thead>
<tr>
<th>Province/City</th>
<th>Date of Birth</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr th:each="user, status : ${users}">
<td th:text="${status.count}>1</td>
<td th:text="${user.firstName + ' ' + user.lastName}">Full Name</td>
<td th:text="${user.email}">Email</td>
<td th:text="${user.gender}">Gender</td>
<td>
<ul class="hobby-list">
<li th:each="hobby : ${user.hobbies}" th:text="${hobby}">Hobbies</li>
</ul>
<span th:if="#lists.isEmpty(user.hobbies)">Non</span>
</td>
<td th:text="${user.province}">City</td>
<td th:text="#dates.format(user.birthday, 'dd/MM/yyyy')">Birthday</td>
<td>
<a th:href="@{/users/{email}/edit(email=${user.email})}" class="btn-edit">Edit</a>
<a th:href="@{/users/{email}/delete(email=${user.email})}" class="btn-delete" onclick="return confirm('Are you sure ?')">Delete</a>
</td>
</tr>
<!-- Hiển thị thông báo nếu danh sách trống -->
<tr th:if="#lists.isEmpty(users)">
<td colspan="8">Empty</td>
</tr>
</tbody>
</table>
```

Create register.html



The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right.

Project Structure:

- DemoThymeleaf (Project)
- .idea
- .mvn
- src
 - main
 - java
 - hsf302.se.fu.vn
 - controller
 - HomeController
 - main
 - DemoThymeleafApplication
 - pojo
 - User
 - resources
 - static.css
 - chapter06.css
 - templates
 - index.html
 - register.html
 - register-success.html
 - test
 - target
 - .gitattributes
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - External Libraries
 - Scratches and Consoles

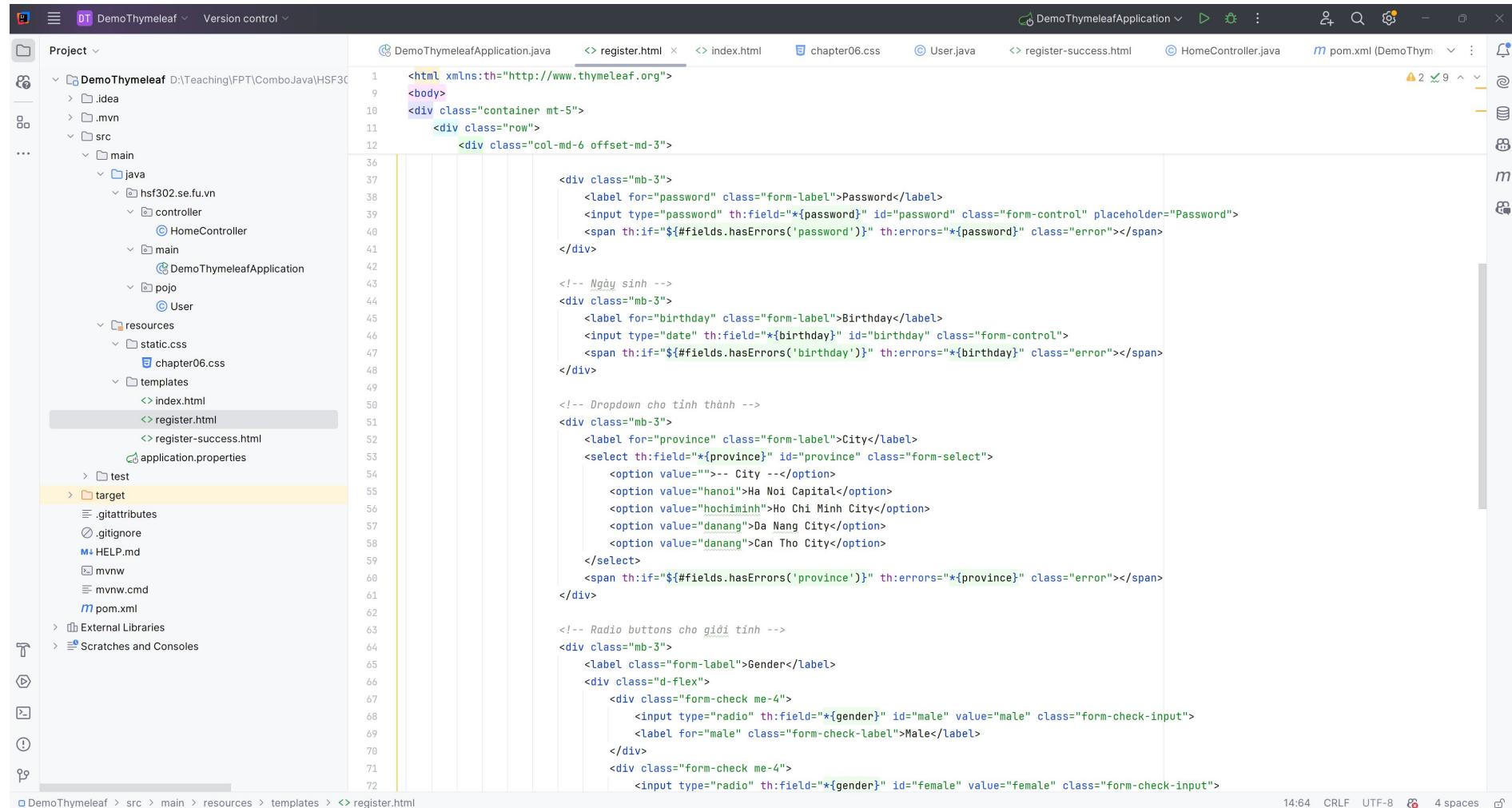
Code Editor (register.html):

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Register</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css">
    <link rel="stylesheet" th:href="@{/css/chapter06.css}">
</head>
<body>
<div class="container mt-5">
    <div class="row">
        <div class="col-md-6 offset-md-3">
            <div class="card">
                <div class="card-header bg-primary text-white">
                    <h3 class="text-center">Register</h3>
                </div>
                <div class="card-body">
                    <form th:action="@{/register}" th:object="${user}" method="post">
                        <div class="mb-3">
                            <label for="firstName" class="form-label">First Name</label>
                            <input type="text" th:field="*{firstName}" id="firstName" class="form-control" placeholder="First Name">
                            <span th:if="#fields.hasErrors('firstName')" th:errors="*{firstName}" class="error"></span>
                        </div>
                        <div class="mb-3">
                            <label for="lastName" class="form-label">Last Name</label>
                            <input type="text" th:field="*{lastName}" id="lastName" class="form-control" placeholder="Last Name">
                            <span th:if="#fields.hasErrors('lastName')" th:errors="*{lastName}" class="error"></span>
                        </div>
                        <div class="mb-3">
                            <label for="email" class="form-label">Email</label>
                            <input type="email" th:field="*{email}" id="email" class="form-control" placeholder="Email">
                            <span th:if="#fields.hasErrors('email')" th:errors="*{email}" class="error"></span>
                        </div>
                        <div class="mb-3">
                            <label for="password" class="form-label">Password</label>
                            <input type="password" th:field="*{password}" id="password" class="form-control" placeholder="Password">
                            <span th:if="#fields.hasErrors('password')" th:errors="*{password}" class="error"></span>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>

```

Status Bar: 14:44 CRLF UTF-8 4 spaces

Create register.html



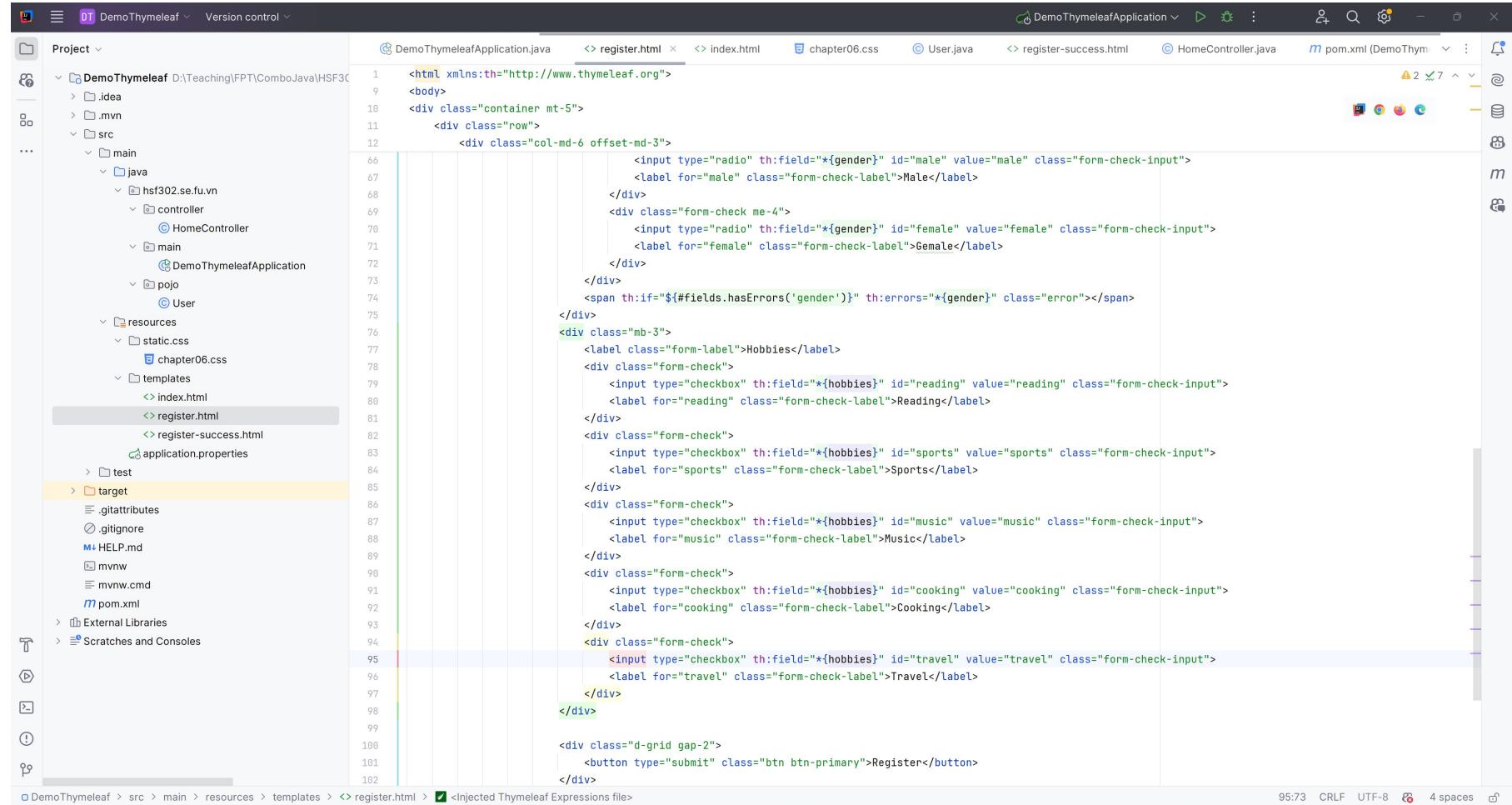
```
<html xmlns:th="http://www.thymeleaf.org">
    <body>
        <div class="container mt-5">
            <div class="row">
                <div class="col-md-6 offset-md-3">
                    <div class="mb-3">
                        <label for="password" class="form-label">Password</label>
                        <input type="password" th:field="#{password}" id="password" class="form-control" placeholder="Password">
                        <span th:if="${#fields.hasErrors('password')}" th:errors="*{password}" class="error"></span>
                    </div>

                    <!-- Ngày sinh -->
                    <div class="mb-3">
                        <label for="birthday" class="form-label">Birthday</label>
                        <input type="date" th:field="#{birthday}" id="birthday" class="form-control">
                        <span th:if="${#fields.hasErrors('birthday')}" th:errors="*{birthday}" class="error"></span>
                    </div>

                    <!-- Dropdown cho tinh thanh -->
                    <div class="mb-3">
                        <label for="province" class="form-label">City</label>
                        <select th:field="#{province}" id="province" class="form-select">
                            <option value="">-- City --</option>
                            <option value="hanoi">Ha Noi Capital</option>
                            <option value="hochiminhh">Ho Chi Minh City</option>
                            <option value="danang">Da Nang City</option>
                            <option value="danang">Can Tho City</option>
                        </select>
                        <span th:if="${#fields.hasErrors('province')}" th:errors="*{province}" class="error"></span>
                    </div>

                    <!-- Radio buttons cho gioi tinh -->
                    <div class="mb-3">
                        <label class="form-label">Gender</label>
                        <div class="d-flex">
                            <div class="form-check me-4">
                                <input type="radio" th:field="#{gender}" id="male" value="male" class="form-check-input">
                                <label for="male" class="form-check-label">Male</label>
                            </div>
                            <div class="form-check me-4">
                                <input type="radio" th:field="#{gender}" id="female" value="female" class="form-check-input">
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>
```

Create register.html



```
<html xmlns:th="http://www.thymeleaf.org">
<body>
<div class="container mt-5">
    <div class="row">
        <div class="col-md-6 offset-md-3">
            <input type="radio" th:field="*{gender}" id="male" value="male" class="form-check-input">
            <label for="male" class="form-check-label">Male</label>
        </div>
        <div class="form-check me-4">
            <input type="radio" th:field="*{gender}" id="female" value="female" class="form-check-input">
            <label for="female" class="form-check-label">Female</label>
        </div>
        <span th:if="${#fields.hasErrors('gender')}" th:errors="*{gender}" class="error"></span>
    </div>
    <div class="mb-3">
        <label class="form-label">Hobbies</label>
        <div class="form-check">
            <input type="checkbox" th:field="*{hobbies}" id="reading" value="reading" class="form-check-input">
            <label for="reading" class="form-check-label">Reading</label>
        </div>
        <div class="form-check">
            <input type="checkbox" th:field="*{hobbies}" id="sports" value="sports" class="form-check-input">
            <label for="sports" class="form-check-label">Sports</label>
        </div>
        <div class="form-check">
            <input type="checkbox" th:field="*{hobbies}" id="music" value="music" class="form-check-input">
            <label for="music" class="form-check-label">Music</label>
        </div>
        <div class="form-check">
            <input type="checkbox" th:field="*{hobbies}" id="cooking" value="cooking" class="form-check-input">
            <label for="cooking" class="form-check-label">Cooking</label>
        </div>
        <div class="form-check">
            <input type="checkbox" th:field="*{hobbies}" id="travel" value="travel" class="form-check-input">
            <label for="travel" class="form-check-label">Travel</label>
        </div>
    </div>
    <div class="d-grid gap-2">
        <button type="submit" class="btn btn-primary">Register</button>
    </div>
</div>
```

Result



User List

Add user							
ID	Full Name	Email	Gender	Hobbies	Province/City	Date of Birth	Action
1	Nguyen Lam	nguyenvana@example.com	male	<ul style="list-style-type: none">readingmusicsport	Ha Noi Capital	11/03/2025	<button>Edit</button> <button>Delete</button>
2	Trần Thị B	tranthib@example.com	female	<ul style="list-style-type: none">travelcooking	Ho Chi Minh City	11/03/2025	<button>Edit</button> <button>Delete</button>

Result



Register

FirstName

Last Name

Email

Password

Birthday
 □

City
 ▼

Gender
 Male Female

Hobbies
 Reading
 Sports
 Music
 Cooking
 Travel

Summary

Concepts were introduced:

- ◆ Building web applications using the Spring Framework.
 - Spring MVC (Model-View-Controller)
 - Spring WebFlux
 - Spring Boot
 - ThymeLeaf with Spring MVC and Spring Boot
- ◆ A modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text (Thymeleaf)