FPT Education
FPT UNIVERSITY

Java

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 Freemaker.
- 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.

FPT Education
FPT UNIVERSITY

Java

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 ,…

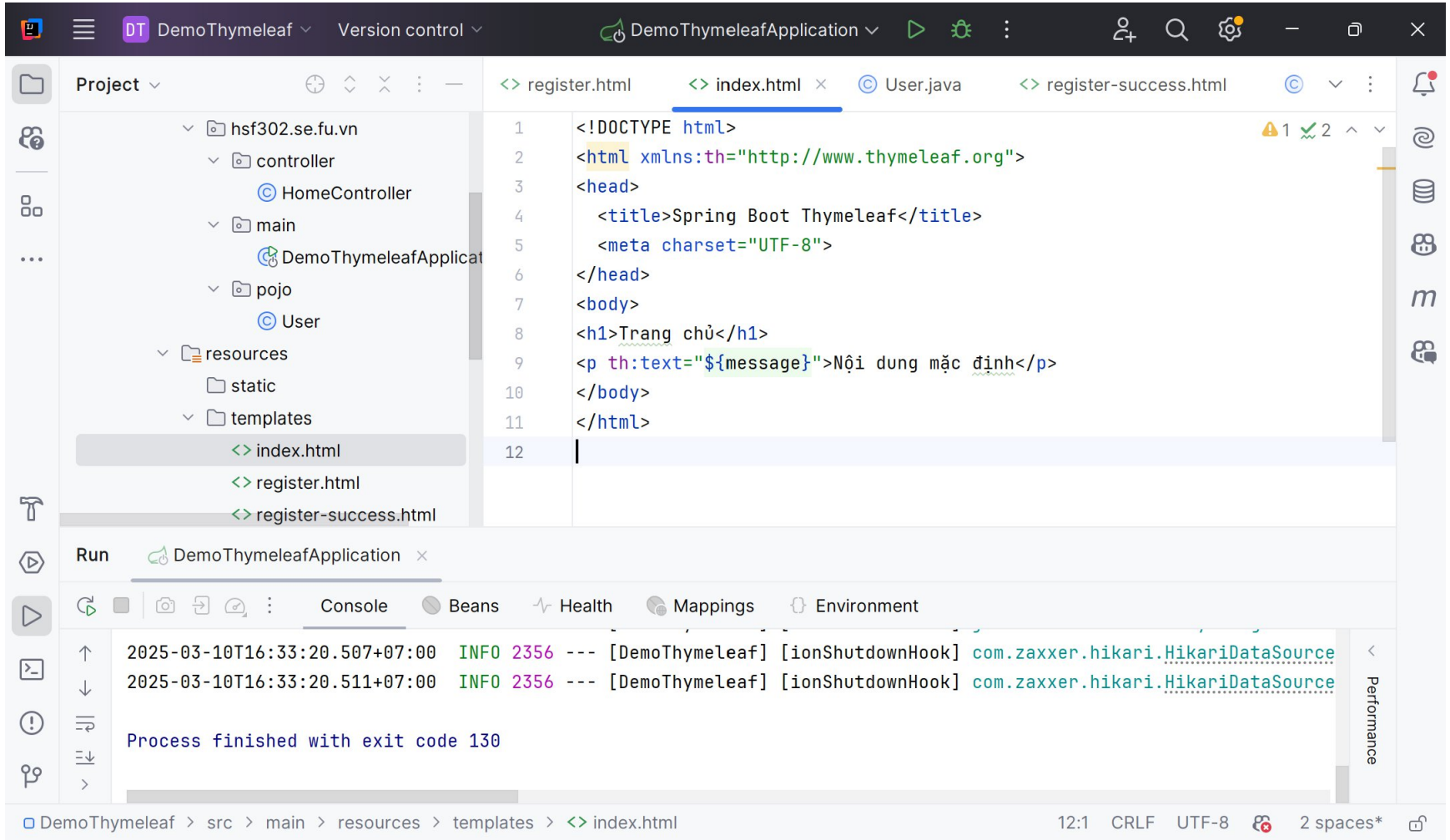
Standard Expression Syntax

- 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



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="${user}" method="post">
                        <div class="mb-3">
                            <label for="firstName" class="form-label">FirstName</label>
                            <input type="text" th:field="*{firstName}" id="firstName" class="form-control" placeholder="First Name">
                            <span th:if="${#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>
```

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

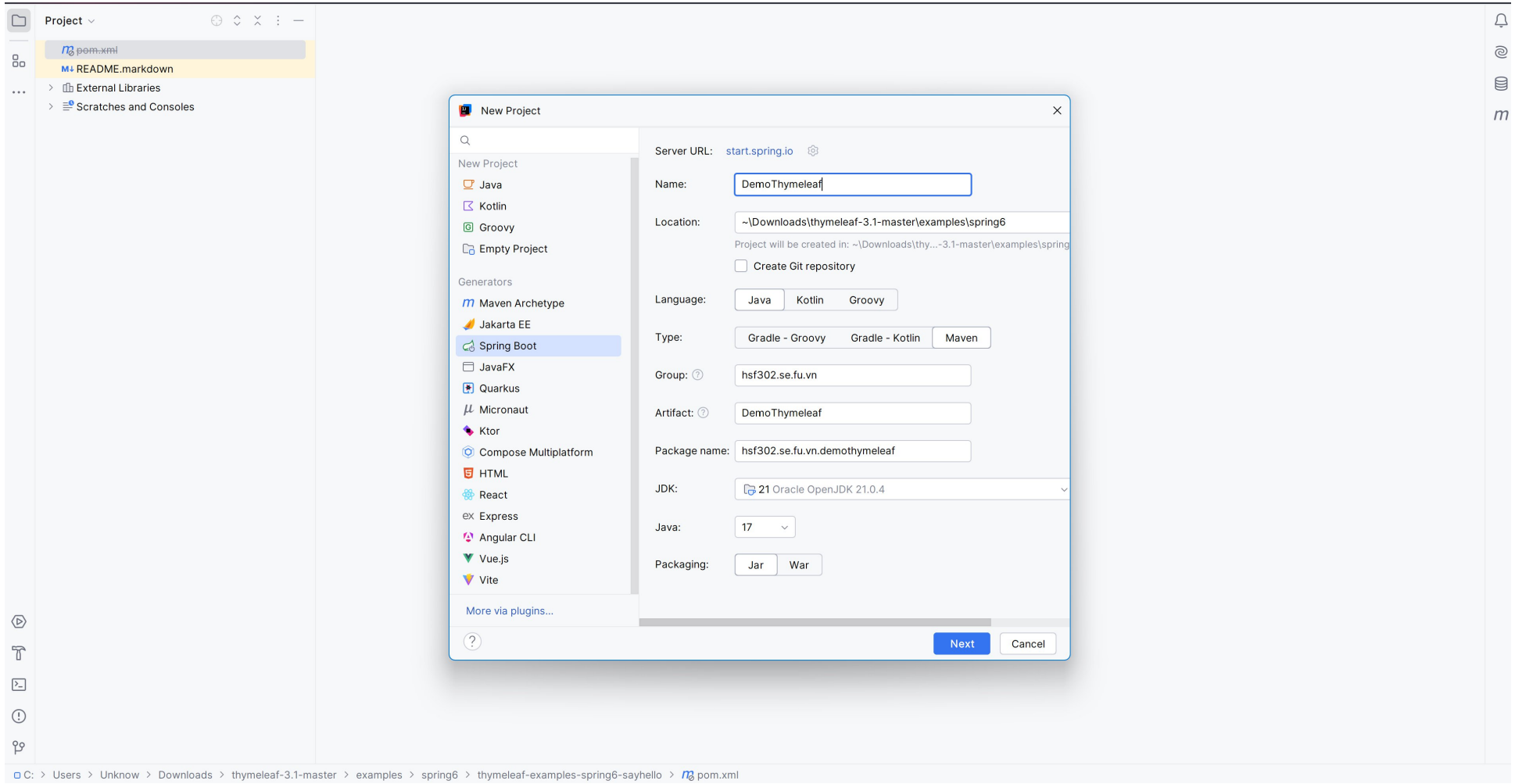
```
<table>
  <thead>
  <tr>
    <th>Province/City</th>
    <th>Date of Birth</th>
    <th>Action</th>
  </tr>
  </thead>
  <tbody>

  <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>
```

FPT Education
FPT UNIVERSITY

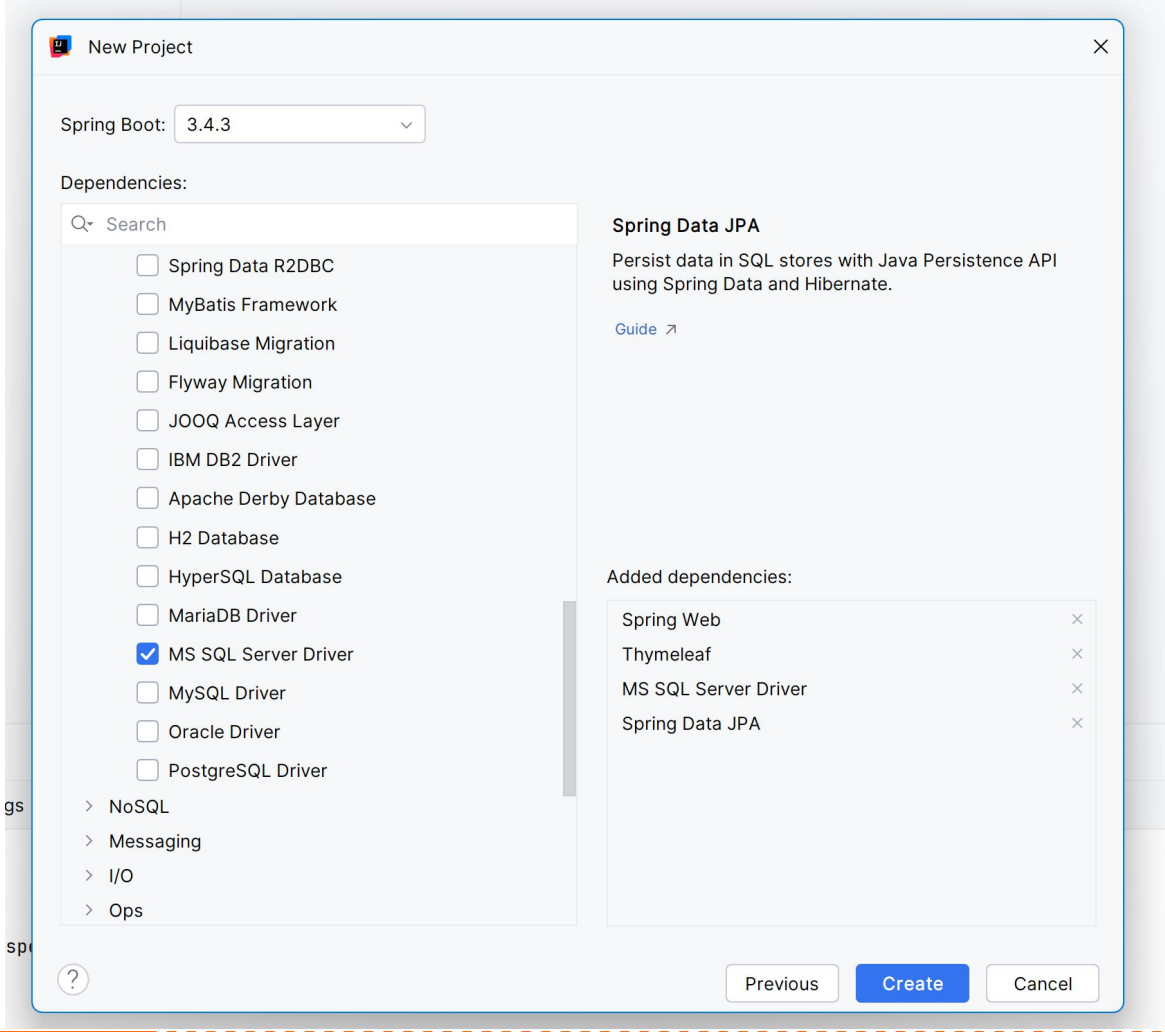
Java

Demo Spring Boot with Thymeleaf Engine

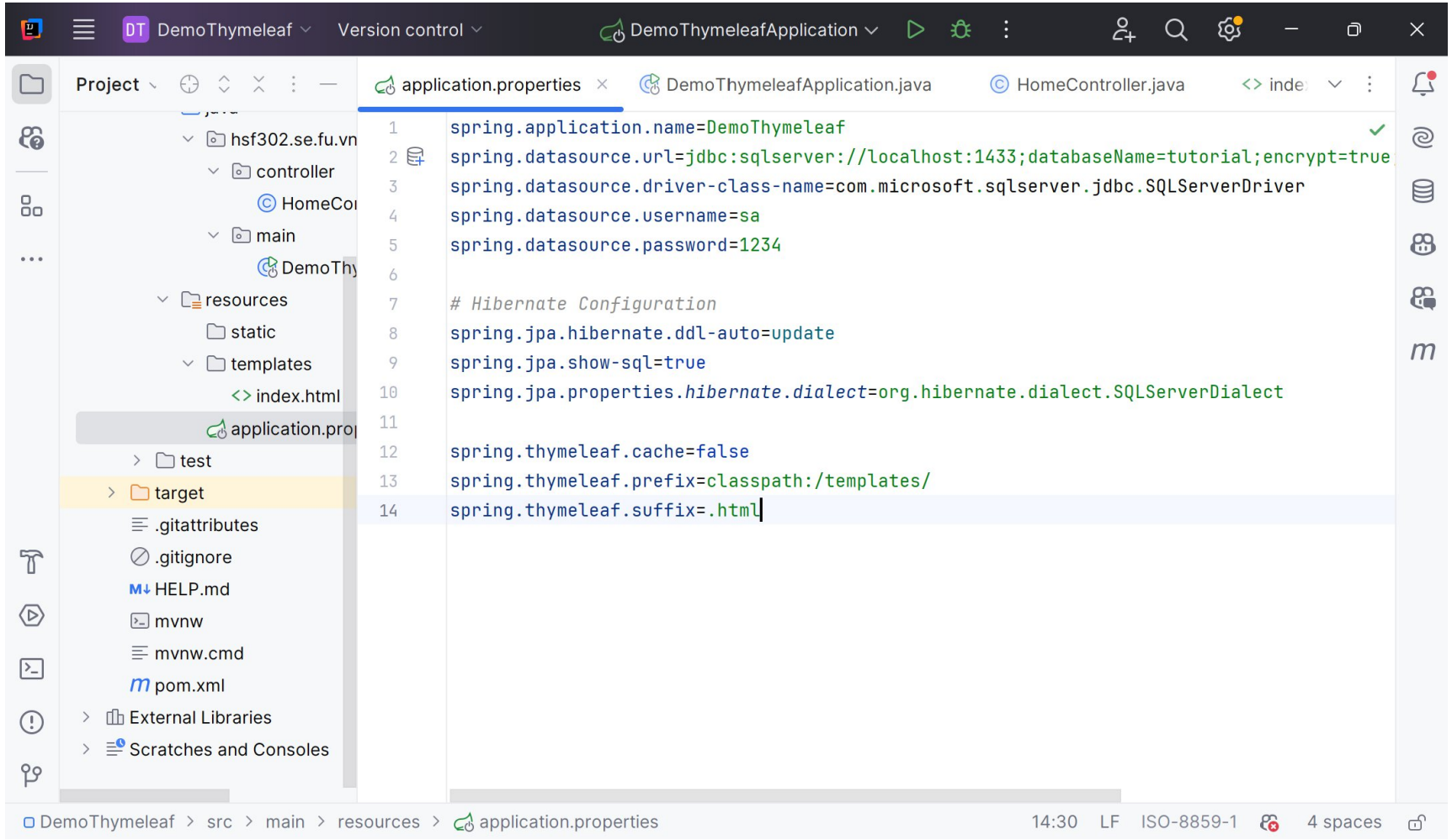
Create a simple project using Thymeleaf



Add dependencies



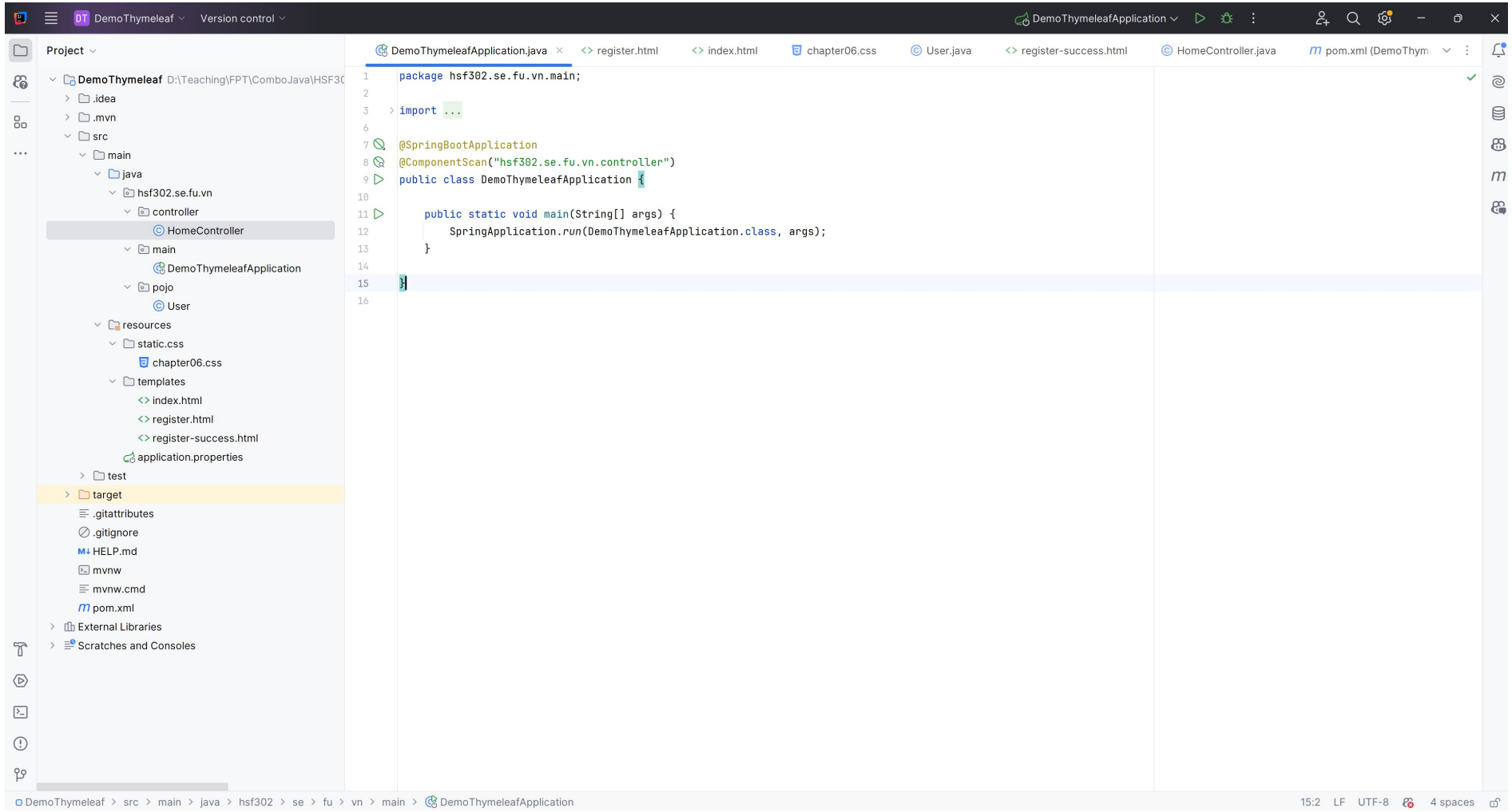
Update the application.properties



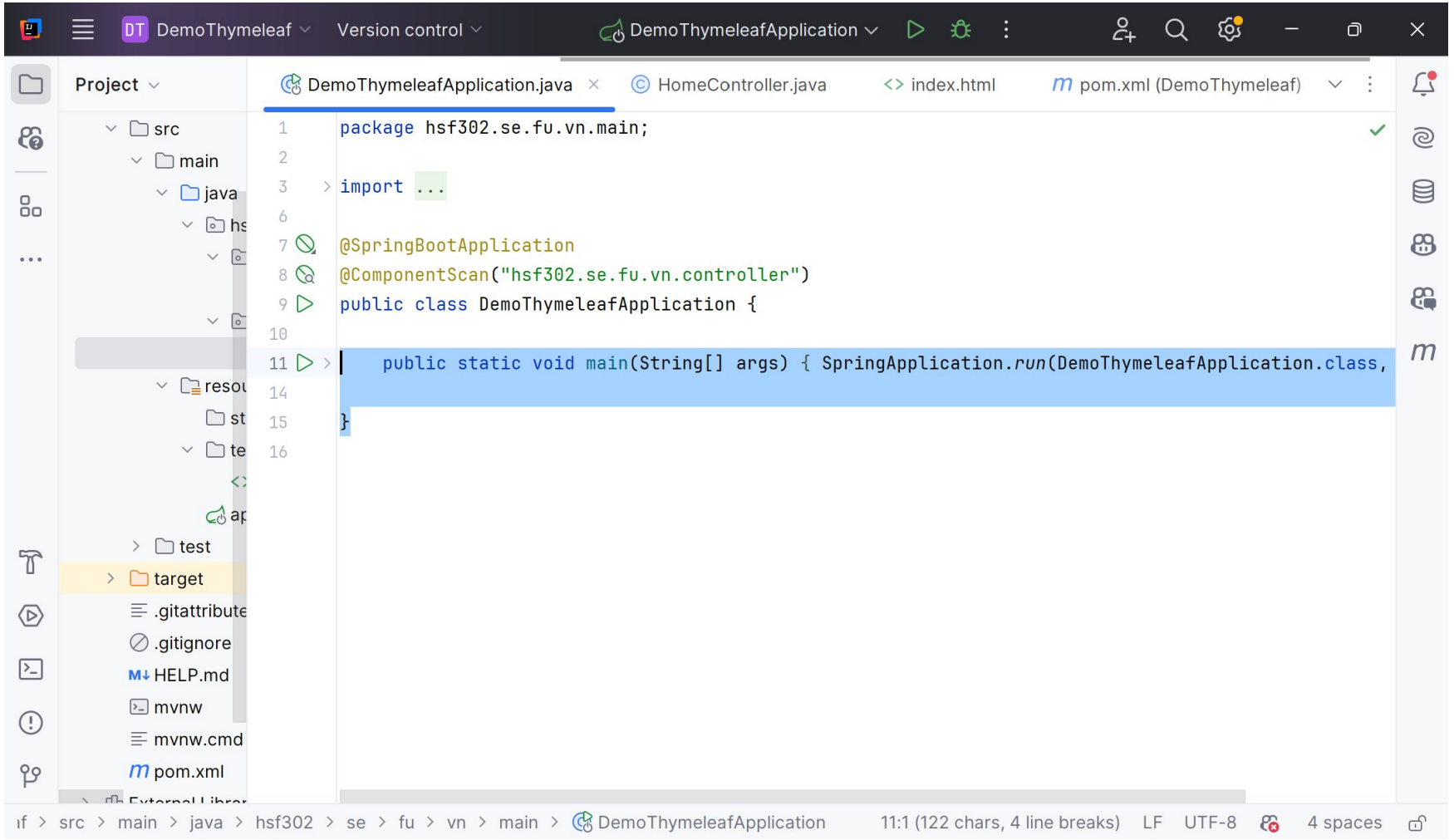
Update pom.xml


```
<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

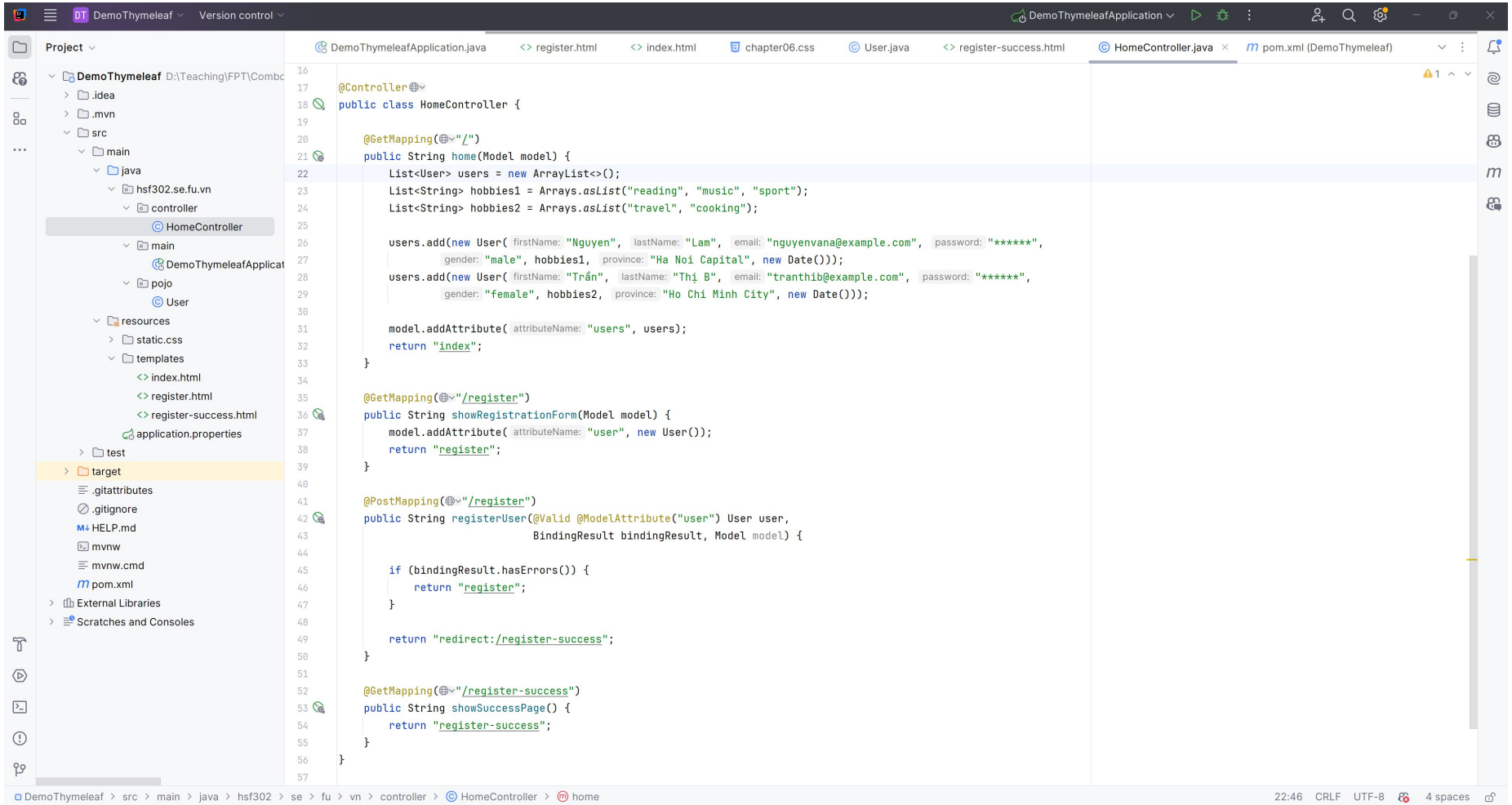


Update main function



FPT Education
FPT UNIVERSITY

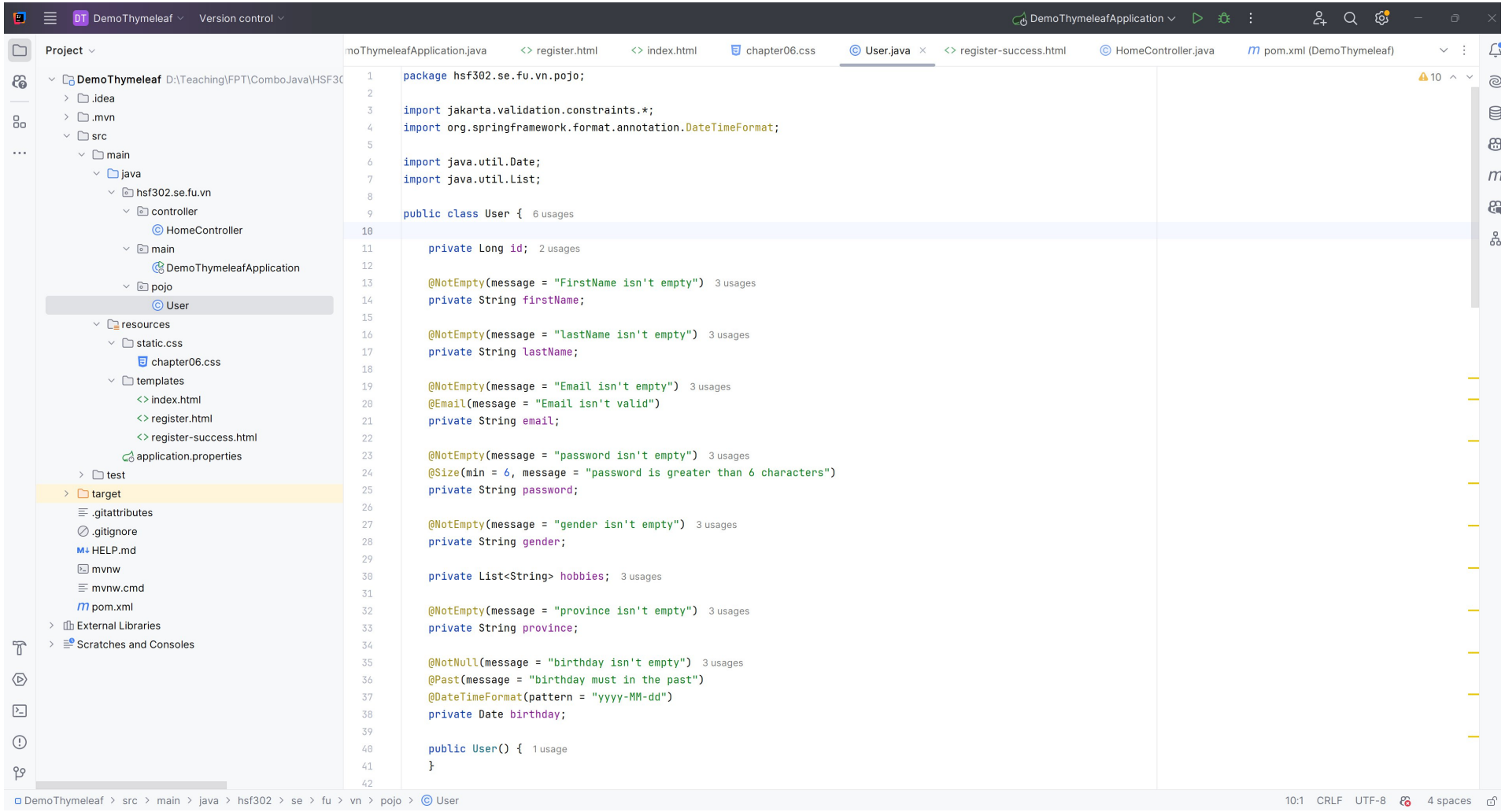
Controller



FPT Education
FPT UNIVERSITY

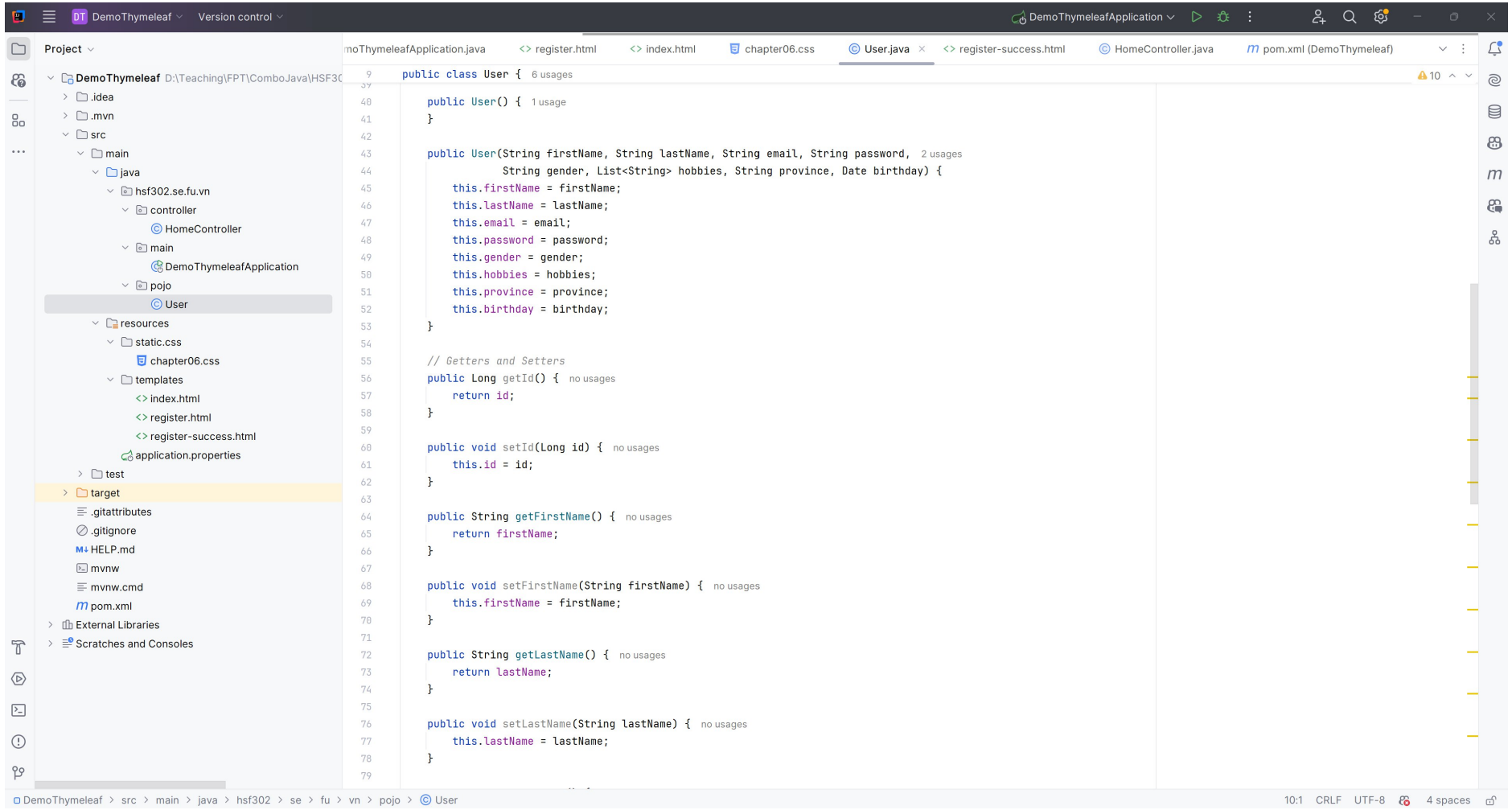
Pojo and validation

Java

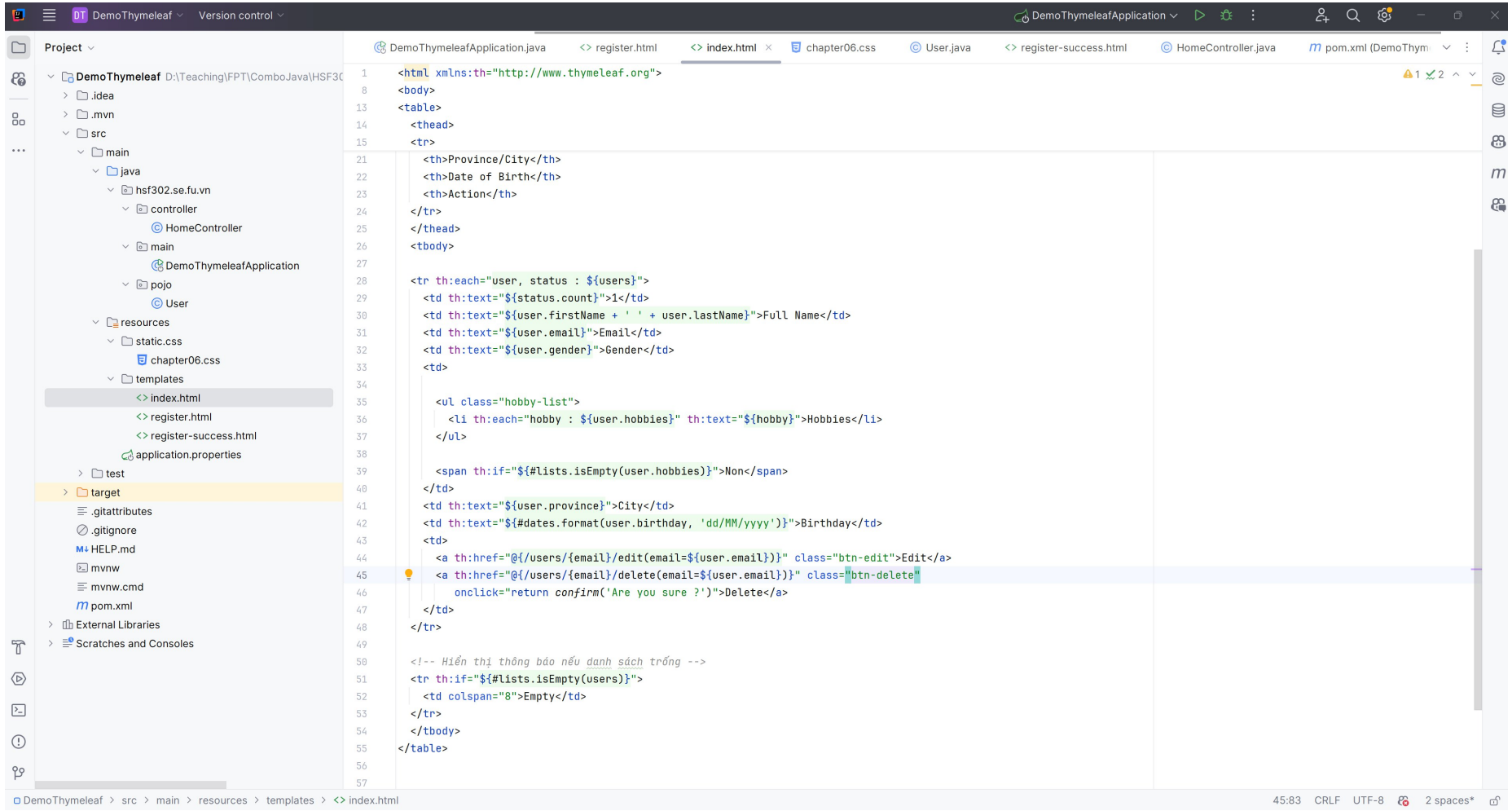


FPT Education
FPT UNIVERSITY

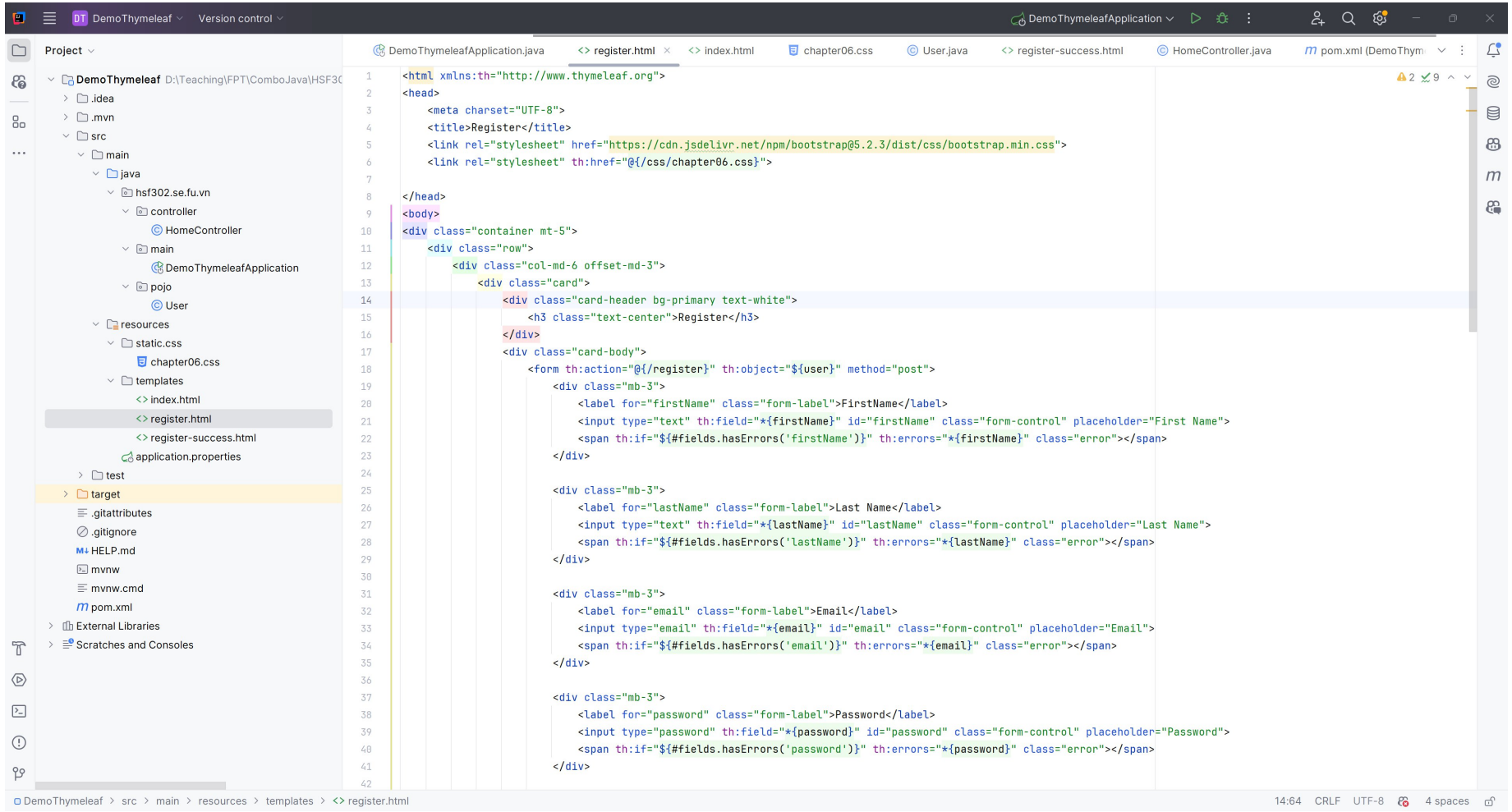
Pojo and validation



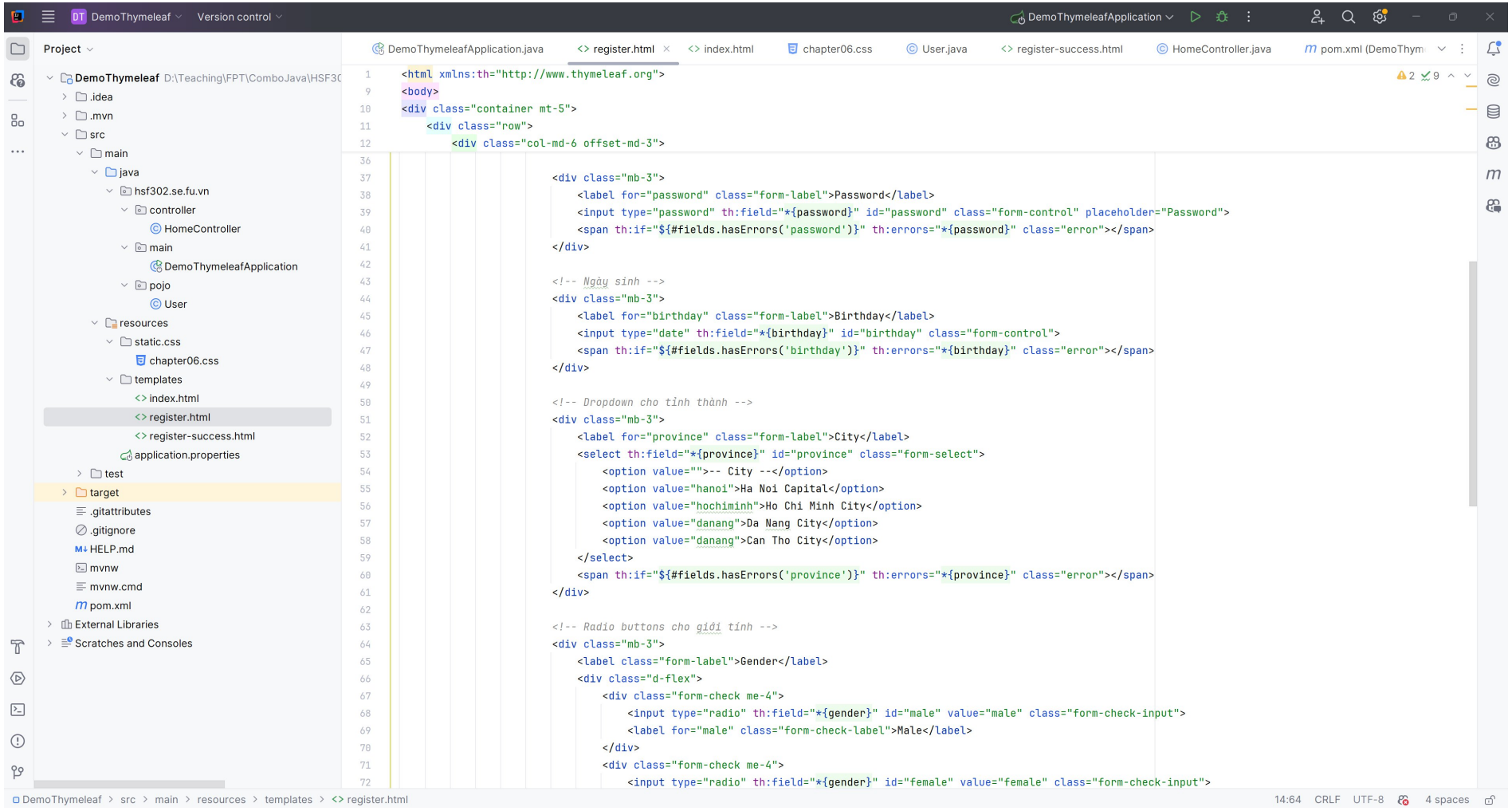
Create index.html



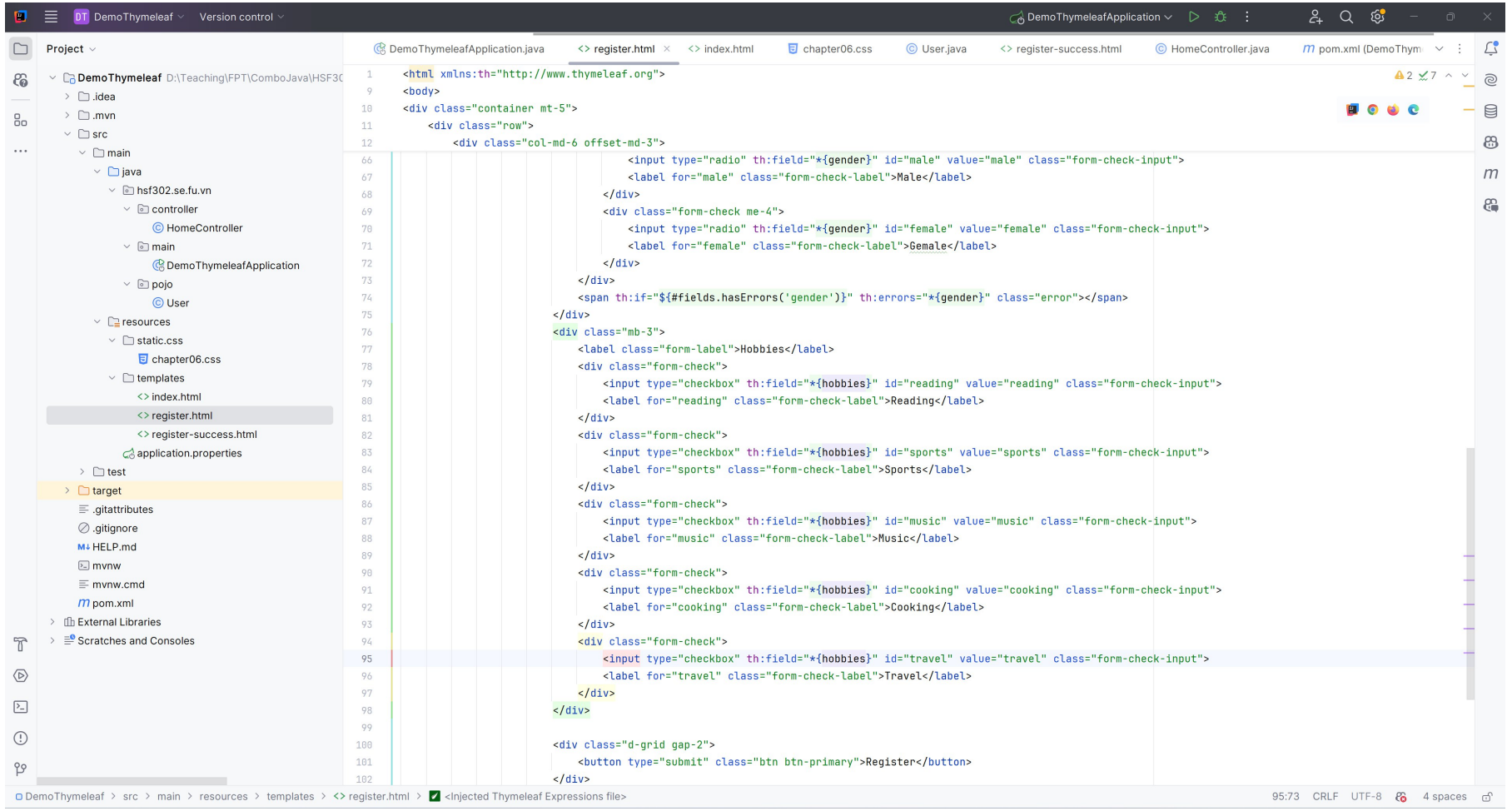
Create register.html



Create register.html



Create register.html



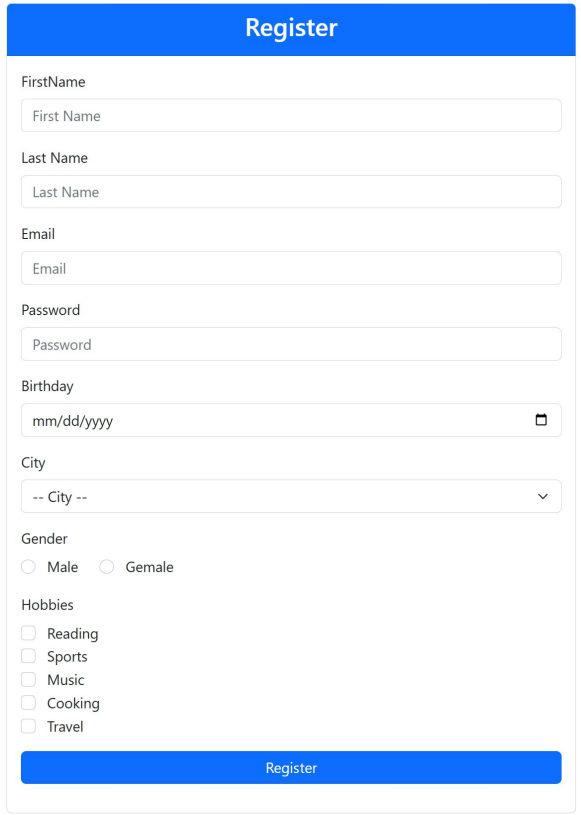
Result

User List

Add user

ID	Full Name	Email	Gender	Hobbies	Province/City	Date of Birth	Action
1	Nguyen Lam	nguyenvana@example.com	male	• reading • music • sport	Ha Noi Capital	11/03/2025	Edit Delete
2	Trần Thị B	tranthib@example.com	female	• travel • cooking	Ho Chi Minh City	11/03/2025	Edit Delete

Result


Register
FirstName
First Name
Last Name
Last Name
Email
Email
Password
Password
Birthday
mm/dd/yyyy
City
-- City --
Gender
Male
Gemale
Hobbies
Reading
Sports
Music
Cooking
Travel
Register

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)