

# Build a simple application with Data Access using Spring Boot & Spring Data JPA

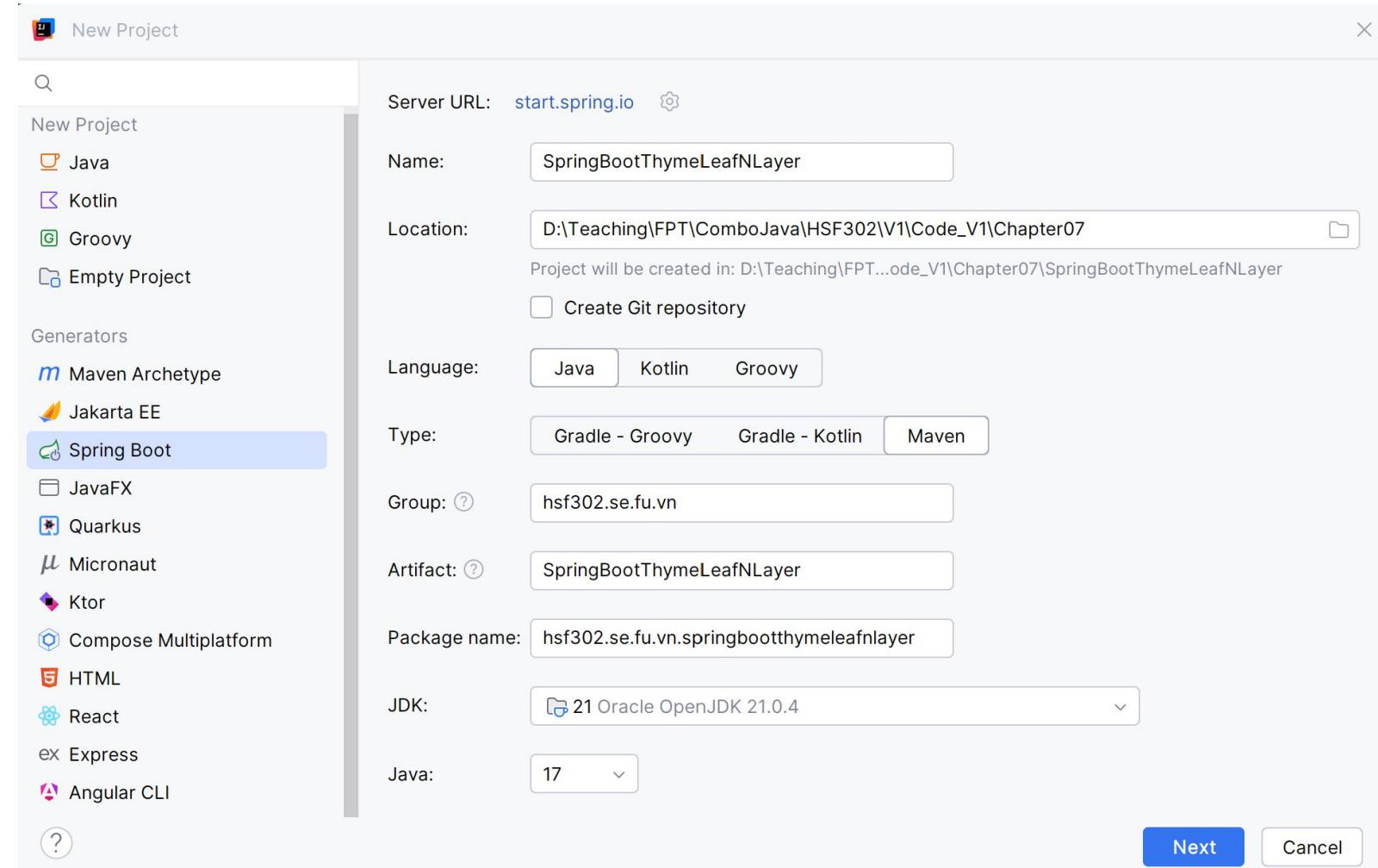
# Objectives

Build application with Data access

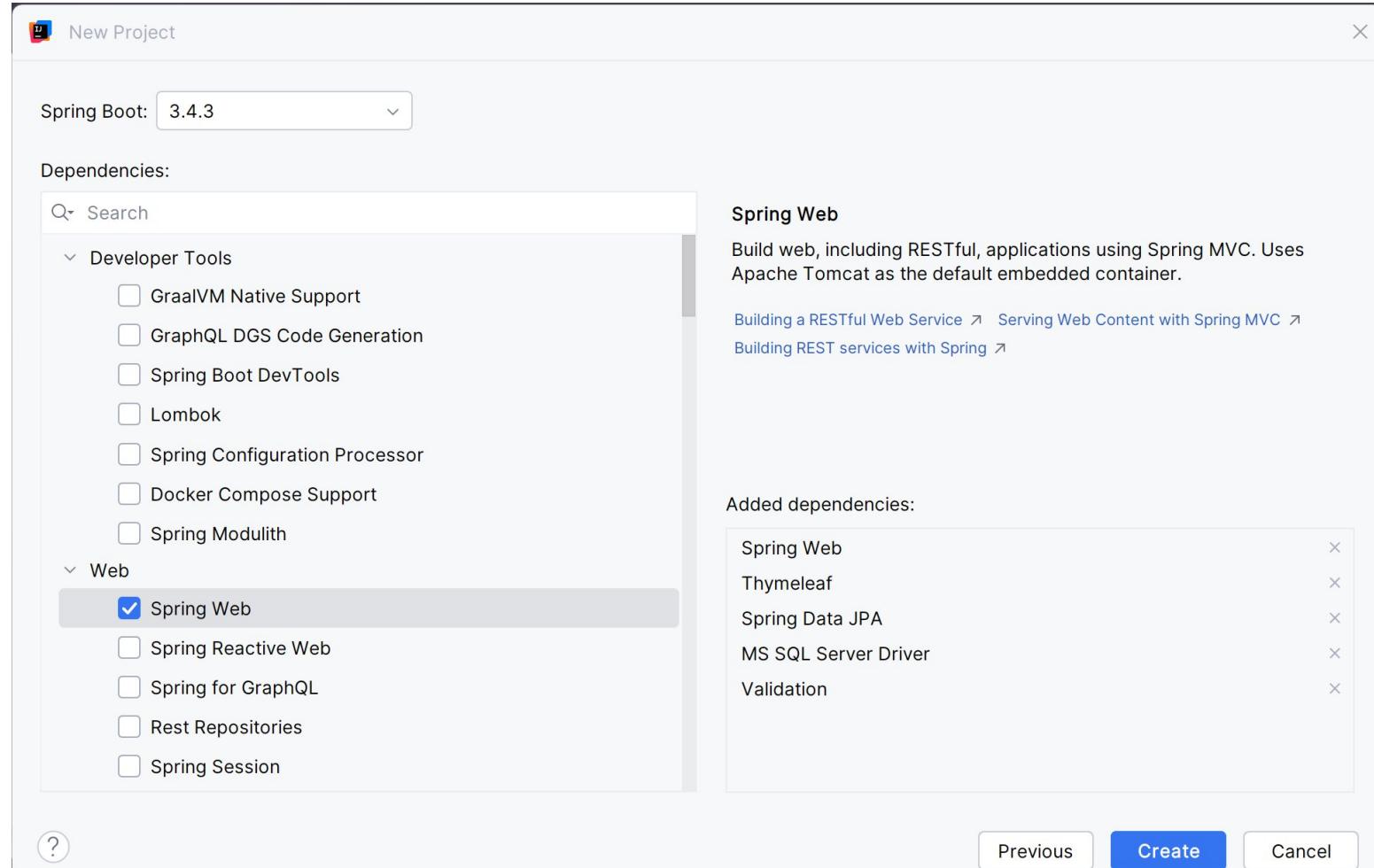
- ◆ Create a new Maven project in IntelliJ IDE
- ◆ Add the necessary dependencies for Spring MVC and data access
- ◆ Create the Model - Define the entity classes that represent domain objects
- ◆ Define the data access object (DAO) interfaces and their implementations for interacting with the database.
- ◆ Create the Controller
- ◆ Set Up the Views
- ◆ Create the views using JSP, Thymeleaf, or another templating engine
- ◆ Implement the Business Logic
- ◆ Testing
- ◆ Run the Application

# Demo Spring Boot with Thymeleaf Engine with Repository Pattern

# Create a simple project using Thymeleaf



# Add dependencies



New Project

Spring Boot: 3.4.3

Dependencies:

- Developer Tools
  - GraalVM Native Support
  - GraphQL DGS Code Generation
  - Spring Boot DevTools
  - Lombok
  - Spring Configuration Processor
  - Docker Compose Support
  - Spring Modulith
- Web
  - Spring Web
  - Spring Reactive Web
  - Spring for GraphQL
  - Rest Repositories
  - Spring Session

Spring Web

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Building a RESTful Web Service ➔ Serving Web Content with Spring MVC ➔ Building REST services with Spring ➔

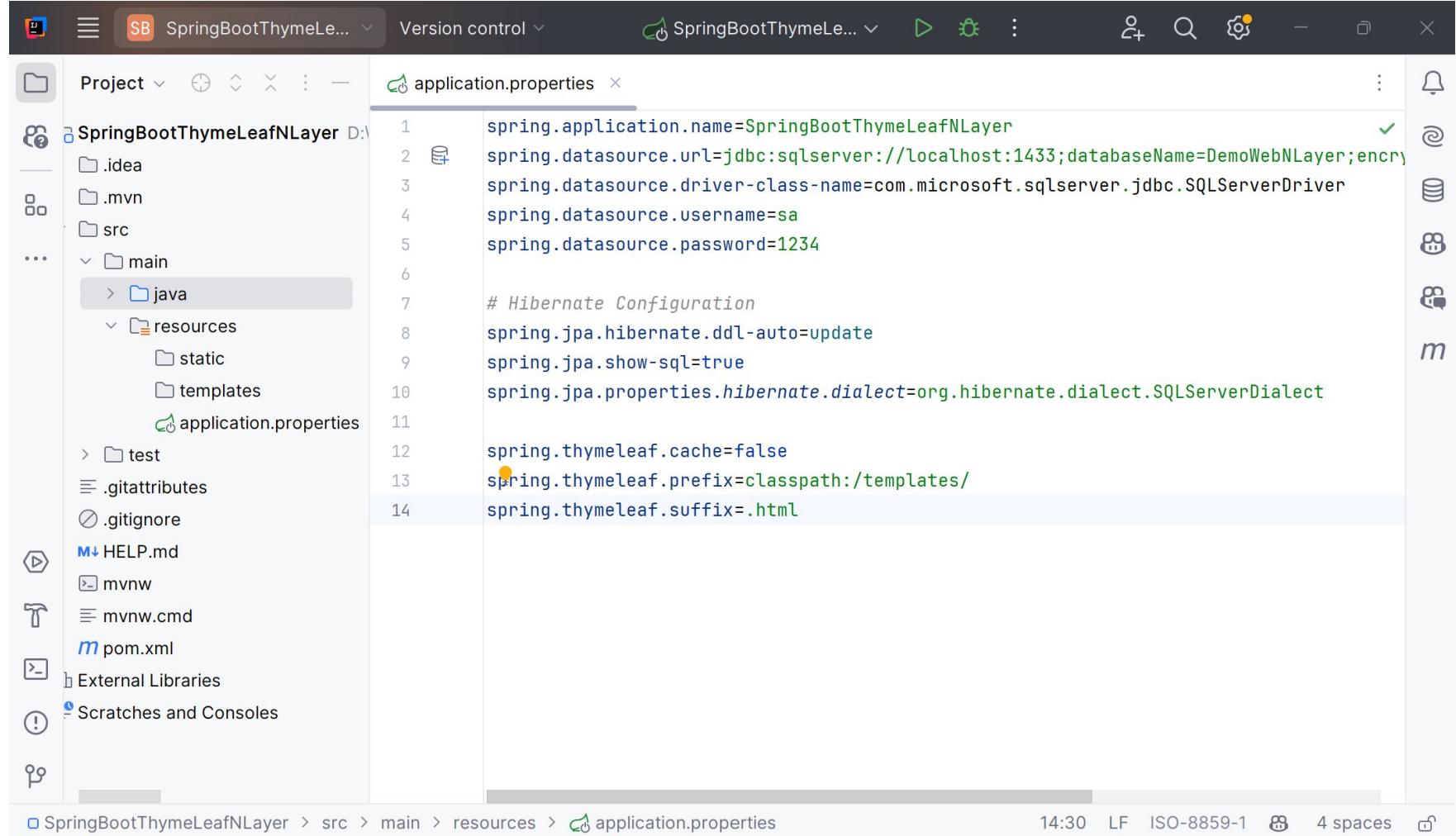
Added dependencies:

- Spring Web
- Thymeleaf
- Spring Data JPA
- MS SQL Server Driver
- Validation

?

Previous Create Cancel

# Update the application.properties



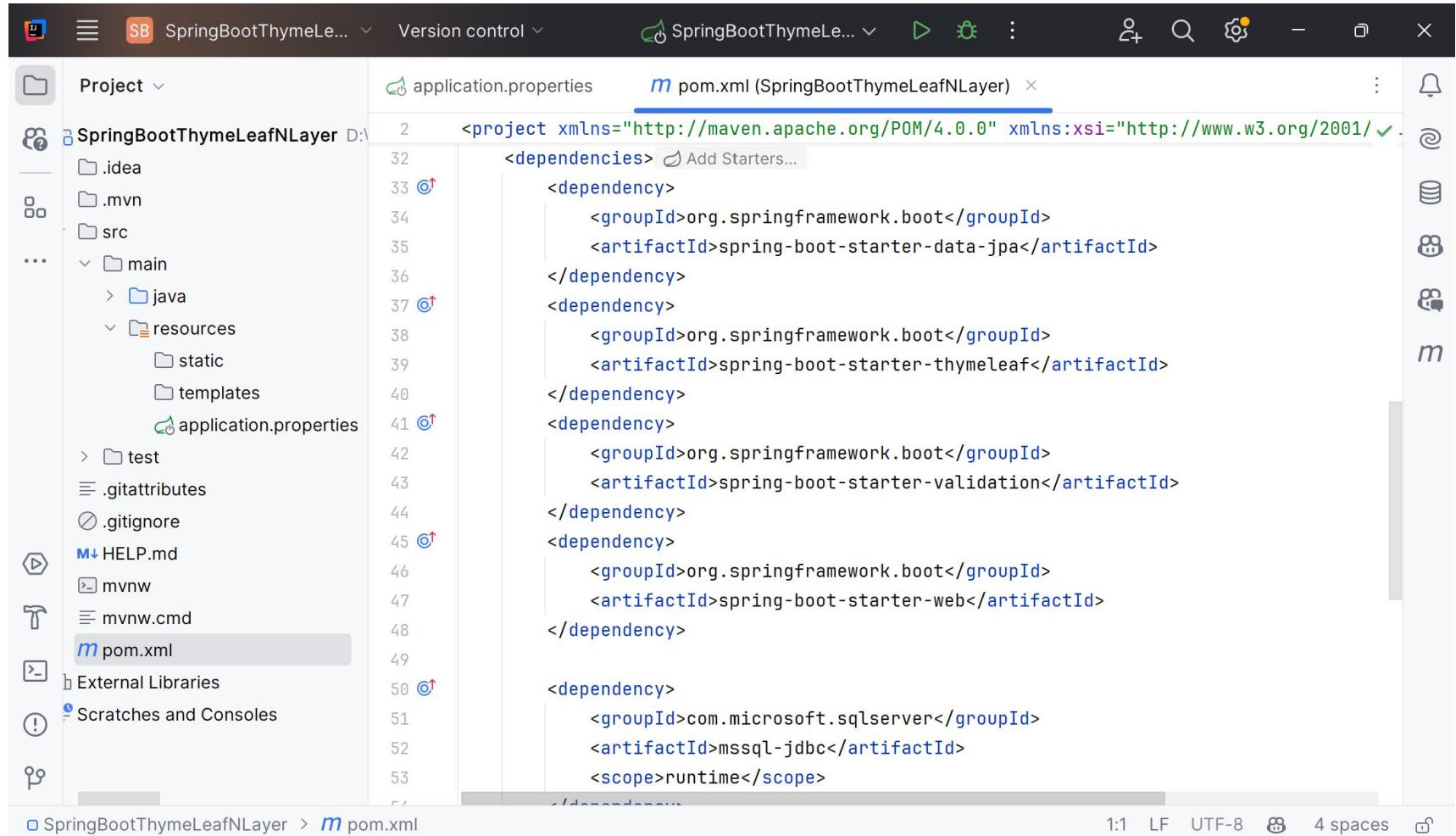
The screenshot shows a code editor interface with the following details:

- Title Bar:** SB SpringBootThymeLe... Version control SpringBootThymeLe... (with icons for file operations like save, copy, paste, etc.)
- Left Sidebar (Project Explorer):** Shows the project structure:
  - Project: SpringBootThymeLeafNLayer
  - Subfolders: .idea, .mvn, src, main, test, .gitattributes, .gitignore, HELP.md, mvnw, mvnw.cmd, pom.xml, External Libraries, Scratches and Consoles.
- Central Area:** The file "application.properties" is open, showing the following configuration:

```
1 spring.application.name=SpringBootThymeLeafNLayer
2 spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=DemoWebNLayer;encry...
3 spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver
4 spring.datasource.username=sa
5 spring.datasource.password=1234
6
7 # Hibernate Configuration
8 spring.jpa.hibernate.ddl-auto=update
9 spring.jpa.show-sql=true
10 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect
11
12 spring.thymeleaf.cache=false
13 spring.thymeleaf.prefix=classpath:/templates/
14 spring.thymeleaf.suffix=.html
```

- Right Sidebar:** Various icons for project management, such as a bell, a spiral, a clipboard, a user, and a search icon labeled 'm'.

# Check the pom.xml



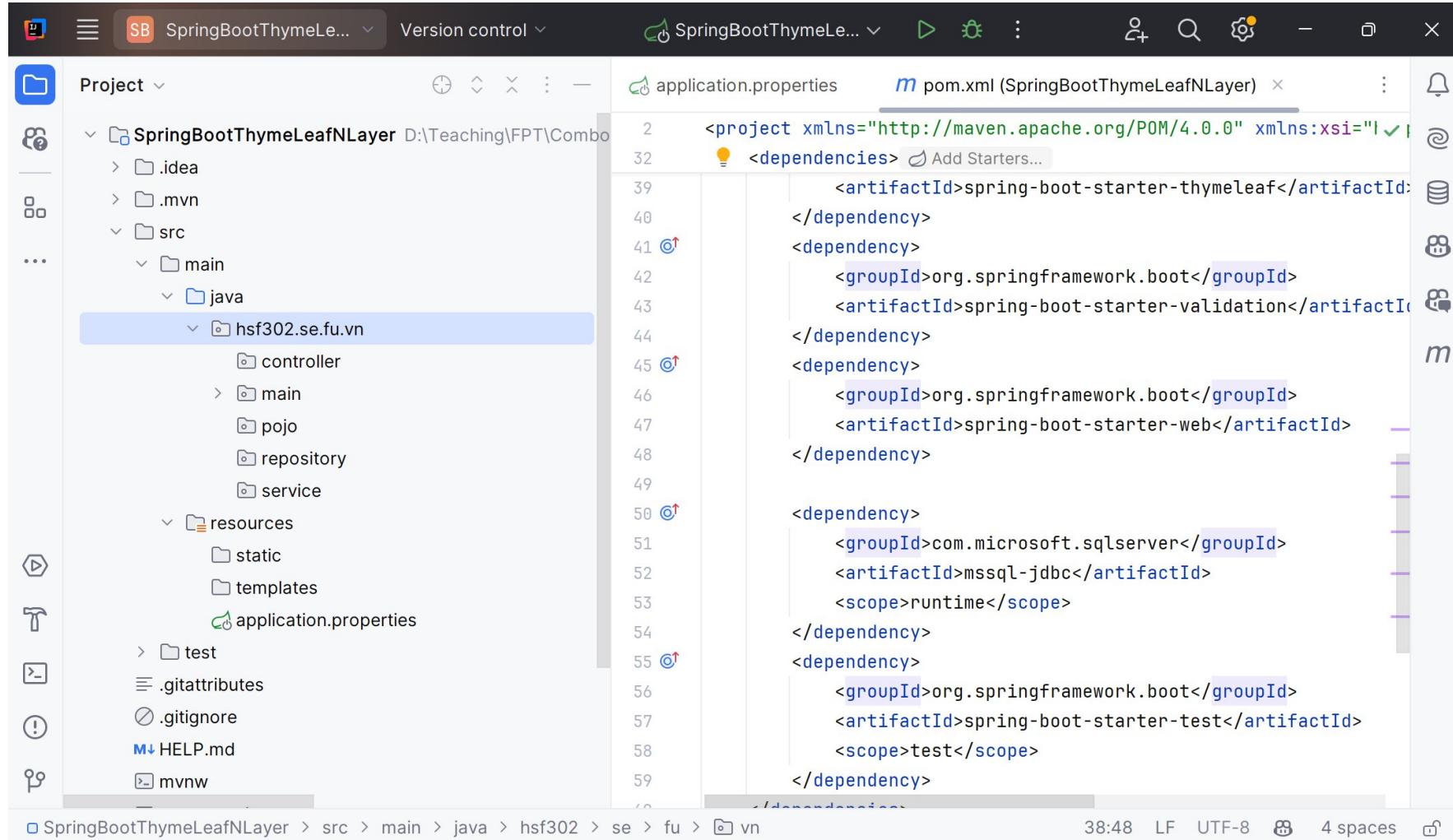
The screenshot shows the IntelliJ IDEA interface with the project "SpringBootThymeLeafNLayer" open. The left sidebar displays the project structure, including .idea, .mvn, src (with main and test), resources (static and templates), application.properties, .gitattributes, .gitignore, HELP.md, mvnw, mvnw.cmd, pom.xml (which is selected and highlighted in blue), External Libraries, Scratches and Consoles, and a GitHub integration icon.

The right pane shows the contents of the pom.xml file:

```
<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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.fpt.edu</groupId>
    <artifactId>SpringBootThymeLeafNLayer</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.2</version>
        <relativePath/>
    </parent>
    <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-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

```

# Create the Structure Project



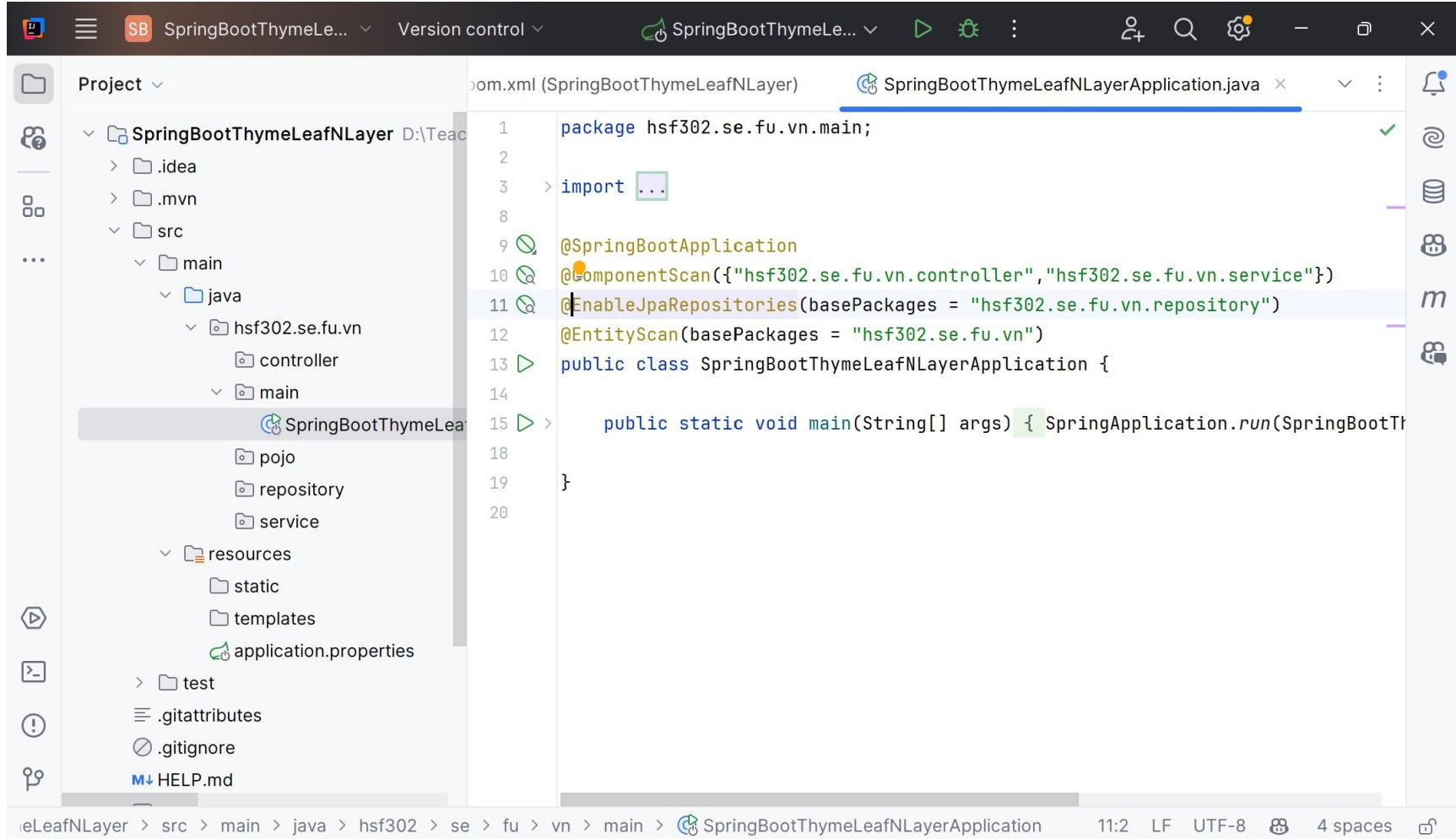
The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "SpringBootThymeLeafNLayer". It contains a .idea folder, a .mvn folder, a src folder with main and test subfolders, and a resources folder with static, templates, and application.properties files.
- pom.xml Content:** The pom.xml file defines the project's dependencies. It includes Spring Boot starters for Thymeleaf, Validation, Web, and Microsoft SQL Server JDBC driver, along with a test dependency for Spring Boot Test.

```
<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 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
  <version>2.5.2</version>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</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>

```

# Update main function



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

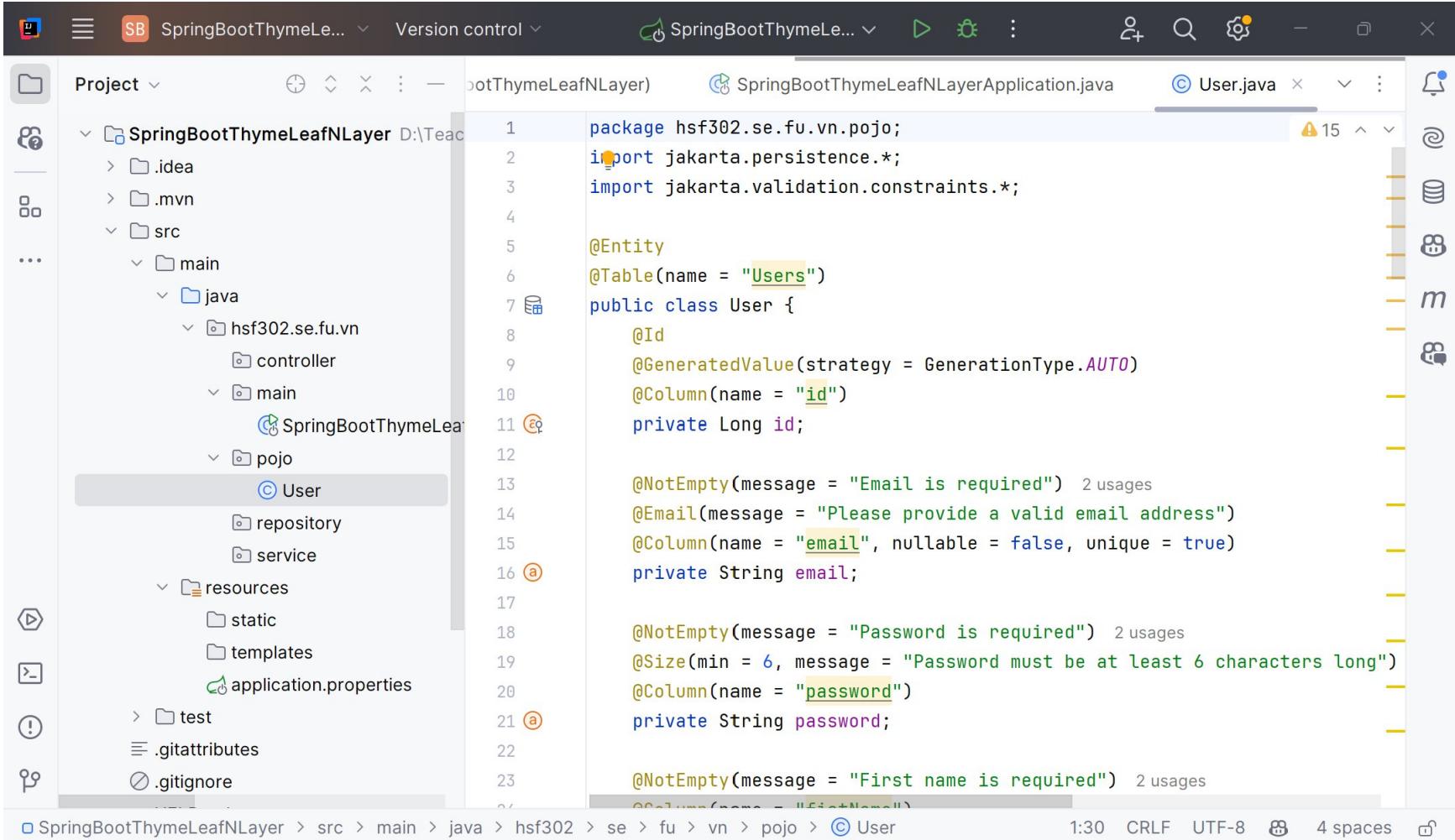
- Title Bar:** SB SpringBootThymeLe... Version control
- Project Structure:** Project > SpringBootThymeLeafNLayer > src > main > java > hsf302 > se > fu > vn > main > SpringBootThymeLeafNLayerApplication.java
- Code Editor:** The current file is SpringBootThymeLeafNLayerApplication.java. The code is as follows:

```
package hsf302.se.fu.vn.main;
import ...
@SpringBootApplication
@ComponentScan({"hsf302.se.fu.vn.controller", "hsf302.se.fu.vn.service"})
@EnableJpaRepositories(basePackages = "hsf302.se.fu.vn.repository")
@EntityScan(basePackages = "hsf302.se.fu.vn")
public class SpringBootThymeLeafNLayerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootThymeLeafNLayerApplication.class, args);
    }
}
```

- Toolbars and Status Bar:** The status bar at the bottom shows the file path, line count (112), character count (LF), encoding (UTF-8), and other settings.

# Pojo and validation



The screenshot shows the IntelliJ IDEA interface with the code editor open to the `User.java` file. The code defines a POJO (Plain Old Java Object) named `User` with validation annotations from the `jakarta.validation.constraints` package.

```
package hsf302.se.fu.vn.pojo;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;

@Entity
@Table(name = "Users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;

    @NotEmpty(message = "Email is required") 2 usages
    @Email(message = "Please provide a valid email address")
    @Column(name = "email", nullable = false, unique = true)
    private String email;

    @NotEmpty(message = "Password is required") 2 usages
    @Size(min = 6, message = "Password must be at least 6 characters long")
    @Column(name = "password")
    private String password;

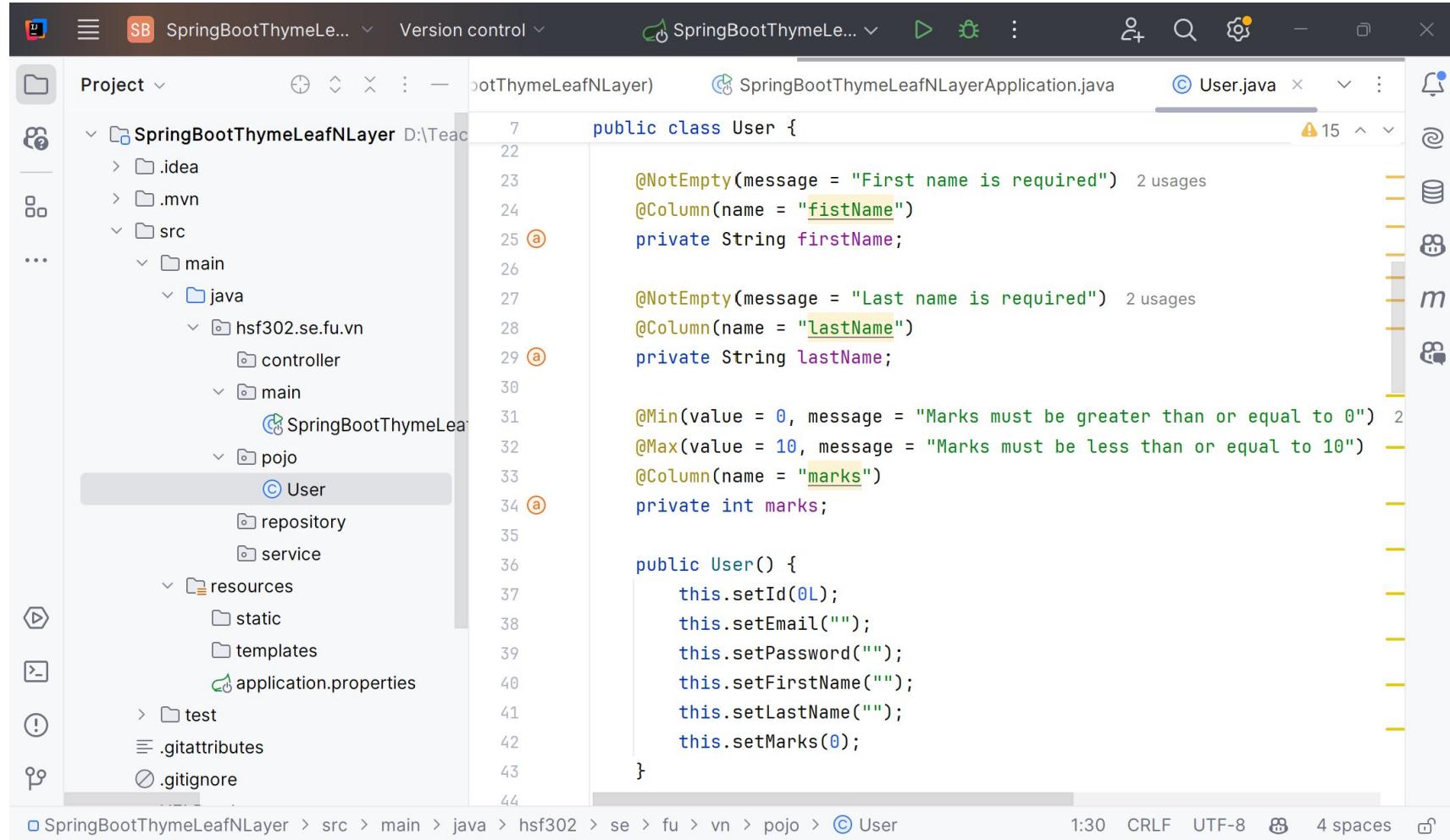
    @NotEmpty(message = "First name is required") 2 usages
    @Column(name = "first_name")
}
```

The code editor shows several validation annotations:

- `@Id`, `@GeneratedValue(strategy = GenerationType.AUTO)`, `@Column(name = "id")`: Primary key definition.
- `@NotEmpty(message = "Email is required")`, `@Email(message = "Please provide a valid email address")`, `@Column(name = "email", nullable = false, unique = true)`: Email validation.
- `@NotEmpty(message = "Password is required")`, `@Size(min = 6, message = "Password must be at least 6 characters long")`, `@Column(name = "password")`: Password validation.
- `@NotEmpty(message = "First name is required")`, `@Column(name = "first_name")`: First name validation.

The project structure on the left shows the `SpringBootThymeLeafNLayer` directory containing `.idea`, `.mvn`, `src`, `test`, `.gitattributes`, and `.gitignore`. The `src/main/java` directory contains `hsf302.se.fu.vn`, `controller`, `main`, `pojo`, `repository`, `service`, and `resources`. The `pojo` directory contains the `User` class.

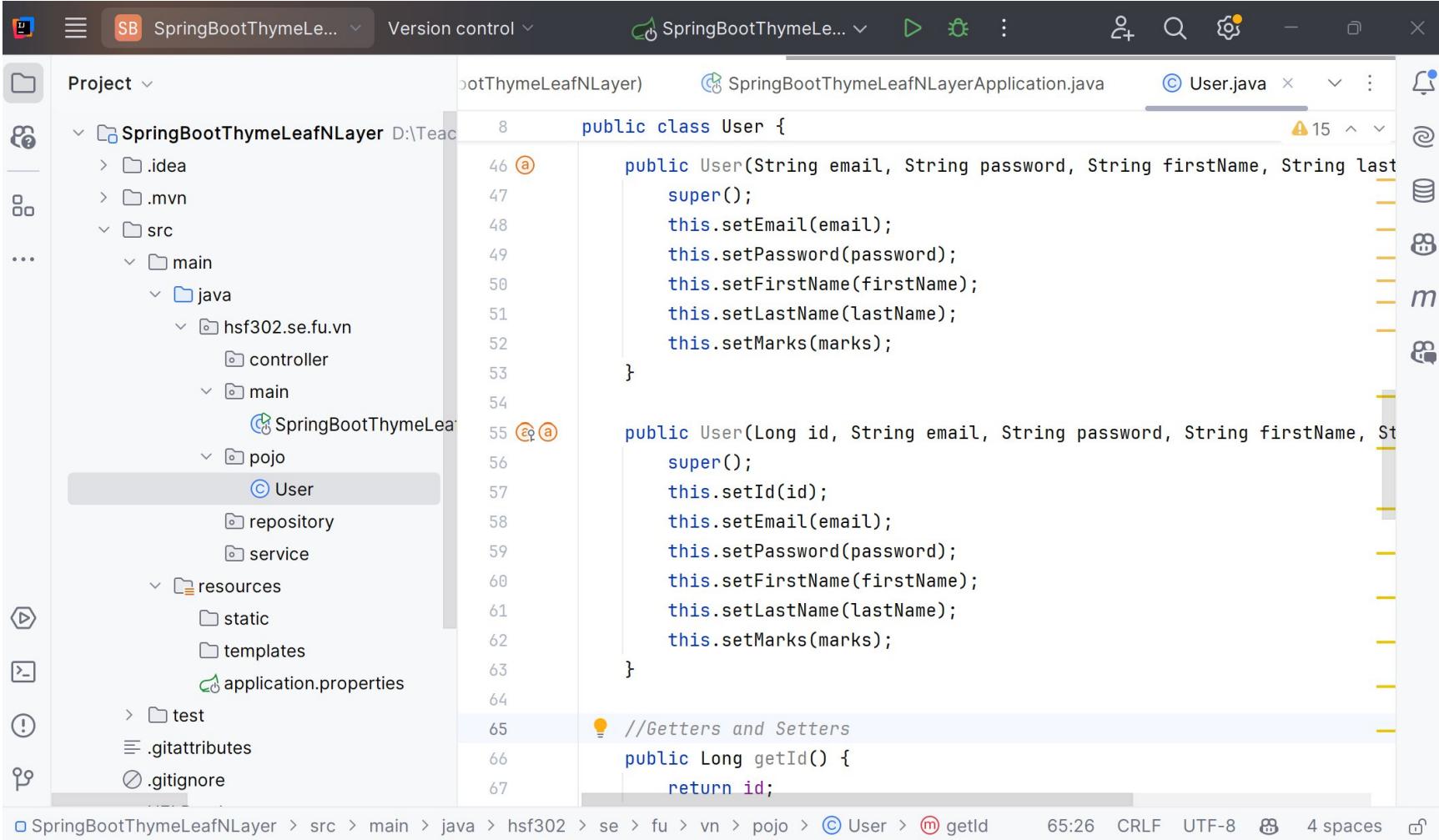
# Pojo and validation



The screenshot shows the IntelliJ IDEA interface with the code editor open to the `User.java` file. The code defines a `User` class with three fields: `firstName`, `lastName`, and `marks`. Each field has a `@Column` annotation with a specific name. The `firstName` and `lastName` fields also have `@NotEmpty` annotations with messages indicating they are required. The `marks` field has `@Min` and `@Max` annotations specifying its range.

```
public class User {  
    @NotEmpty(message = "First name is required") 2 usages  
    @Column(name = "firstName")  
    private String firstName;  
  
    @NotEmpty(message = "Last name is required") 2 usages  
    @Column(name = "lastName")  
    private String lastName;  
  
    @Min(value = 0, message = "Marks must be greater than or equal to 0") 2  
    @Max(value = 10, message = "Marks must be less than or equal to 10")  
    @Column(name = "marks")  
    private int marks;  
  
    public User() {  
        this.setId(0L);  
        this.setEmail("");  
        this.setPassword("");  
        this.setFirstName("");  
        this.setLastName("");  
        this.setMarks(0);  
    }  
}
```

# Pojo and validation



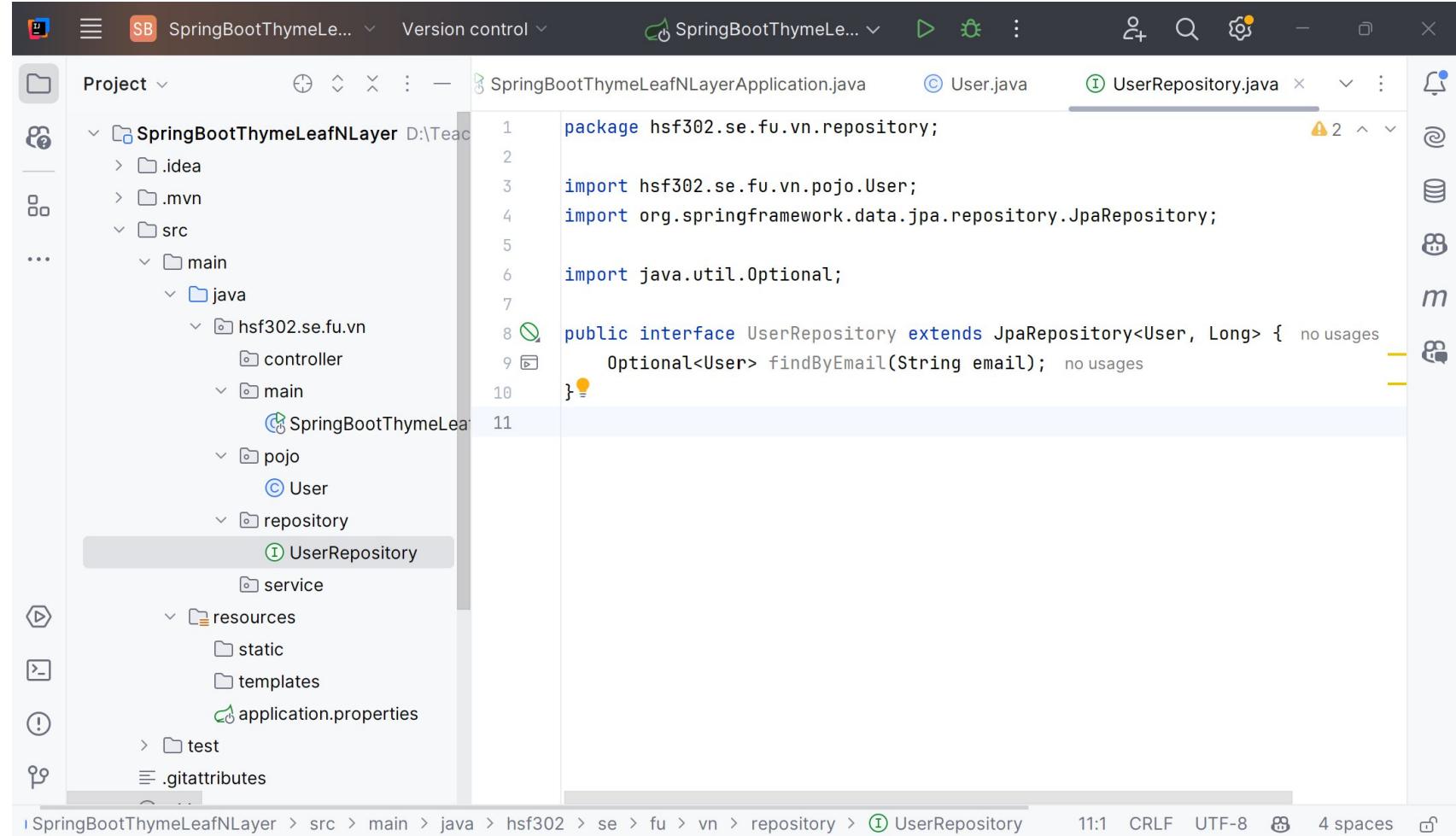
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "SpringBootThymeLeafNLayer". The structure includes .idea, .mvn, src (with main/java and main/resources), test, .gitattributes, and .gitignore.
- User.java Content:** The code defines a User class with two constructors, getters and setters, and a comment indicating the presence of Getters and Setters.

```
public class User {  
    public User(String email, String password, String firstName, String lastName) {  
        super();  
        this.setEmail(email);  
        this.setPassword(password);  
        this.setFirstName(firstName);  
        this.setLastName(lastName);  
        this.setMarks(marks);  
    }  
  
    public User(Long id, String email, String password, String firstName, String lastName) {  
        super();  
        this.setId(id);  
        this.setEmail(email);  
        this.setPassword(password);  
        this.setFirstName(firstName);  
        this.setLastName(lastName);  
        this.setMarks(marks);  
    }  
  
    //Getters and Setters  
    public Long getId() {  
        return id;  
    }  
}
```

- Toolbars and Status Bar:** The top bar shows tabs for "SpringBootThymeLeafNLayer" and "SpringBootThymeLeafNLayerApplication.java". The status bar at the bottom shows file path, line count (65:26), encoding (CRLF), and character set (UTF-8).

# Create the UserRepository.java

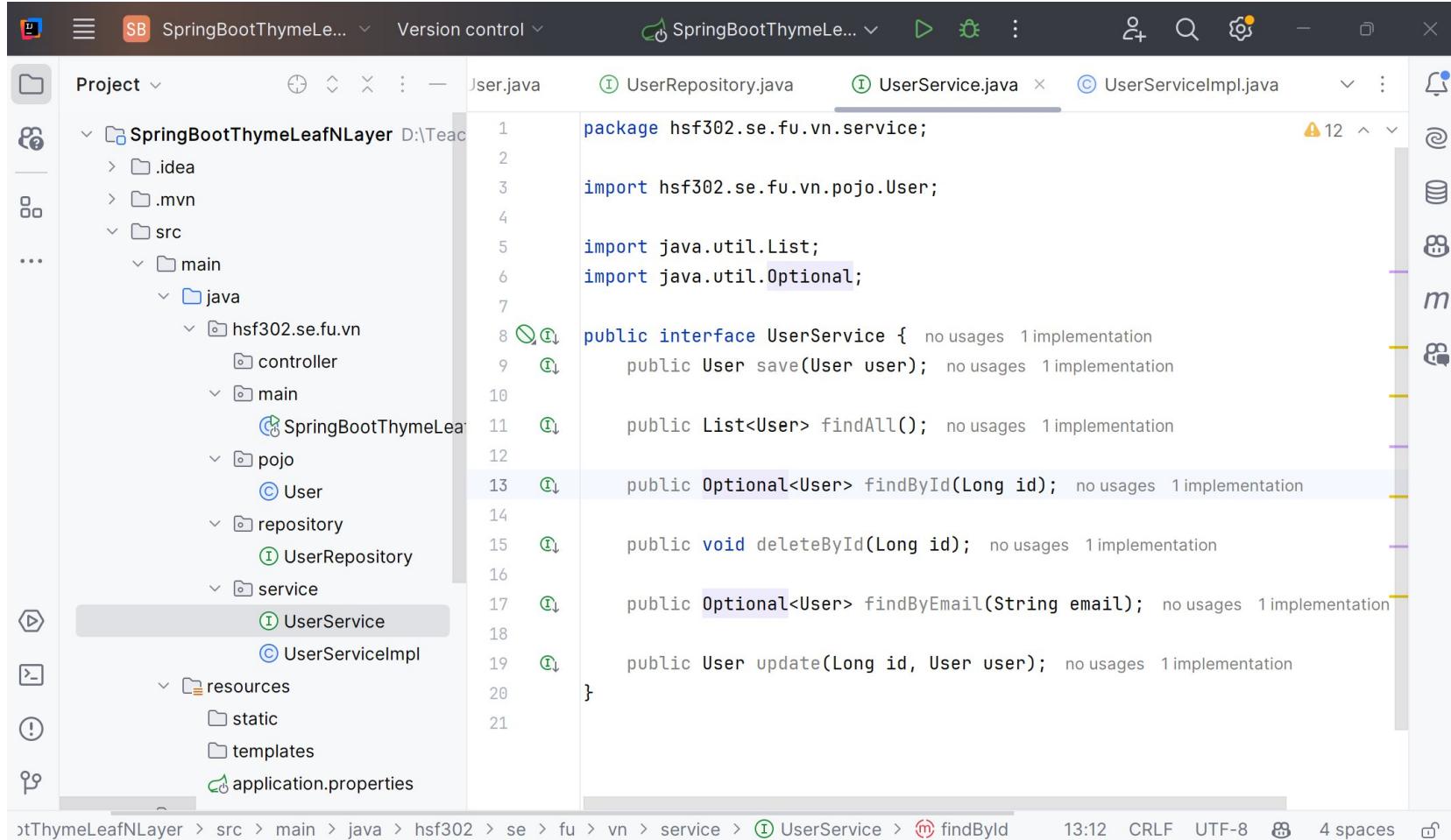


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "SpringBootThymeLeafNLayer". It contains a ".idea" folder, a ".mvn" folder, and a "src" folder. The "src" folder has a "main" directory, which contains a "java" directory. Inside the "java" directory, there is a package named "hsf302.se.fu.vn". This package contains "controller", "main", "pojo", "User", "repository", and "service" sub-directories. The "repository" directory is currently selected.
- Code Editor:** The file "UserRepository.java" is open in the editor. The code is as follows:

```
1 package hsf302.se.fu.vn.repository;
2
3 import hsf302.se.fu.vn.pojo.User;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.Optional;
7
8 public interface UserRepository extends JpaRepository<User, Long> {
9     Optional<User> findByEmail(String email);
10 }
```

# Create the UserService.java



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

- Project Structure:** The project is named "SpringBootThymeLeafNLayer". It contains a .idea folder, a .mvn folder, and a src folder. The src folder has main and java subfolders. The java folder contains hsf302.se.fu.vn, controller, main, pojo, repository, and service subfolders. The service folder contains UserRepository.java, UserService.java (selected), and UserServiceImpl.java.
- Code Editor:** The code editor displays the UserService.java file. The code is as follows:

```
package hsf302.se.fu.vn.service;

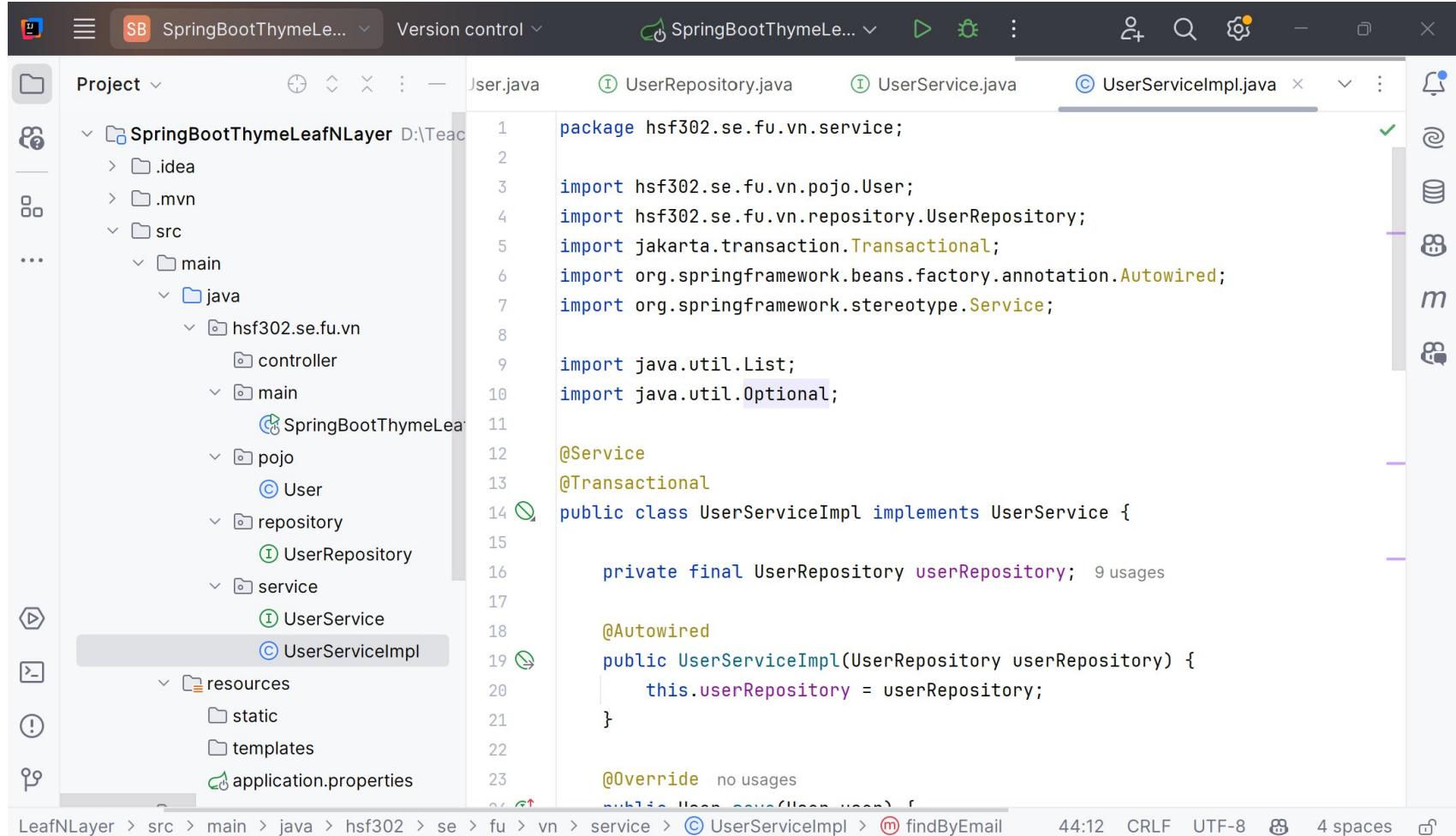
import hsf302.se.fu.vn.pojo.User;

import java.util.List;
import java.util.Optional;

public interface UserService {
    public User save(User user);
    public List<User> findAll();
    public Optional<User> findById(Long id);
    public void deleteById(Long id);
    public Optional<User> findByEmail(String email);
    public User update(Long id, User user);
}
```

- Toolbars and Status Bar:** The top bar includes tabs for Project, Version control, and other files. The bottom status bar shows the file path (src/main/java/hsf302/se/fu/vn/service/UserService.java), the current line (13:12), encoding (CRLF), character set (UTF-8), and code style (4 spaces).

# Create the UserServiceImpl.java



The screenshot shows the IntelliJ IDEA interface with the code editor open to the `UserServiceImpl.java` file. The code implements the `UserService` interface, utilizing `UserRepository` and `java.util.List`. Annotations include `@Service`, `@Transactional`, and `@Autowired`. The code editor displays line numbers and syntax highlighting.

```
package hsf302.se.fu.vn.service;

import hsf302.se.fu.vn.pojo.User;
import hsf302.se.fu.vn.repository.UserRepository;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

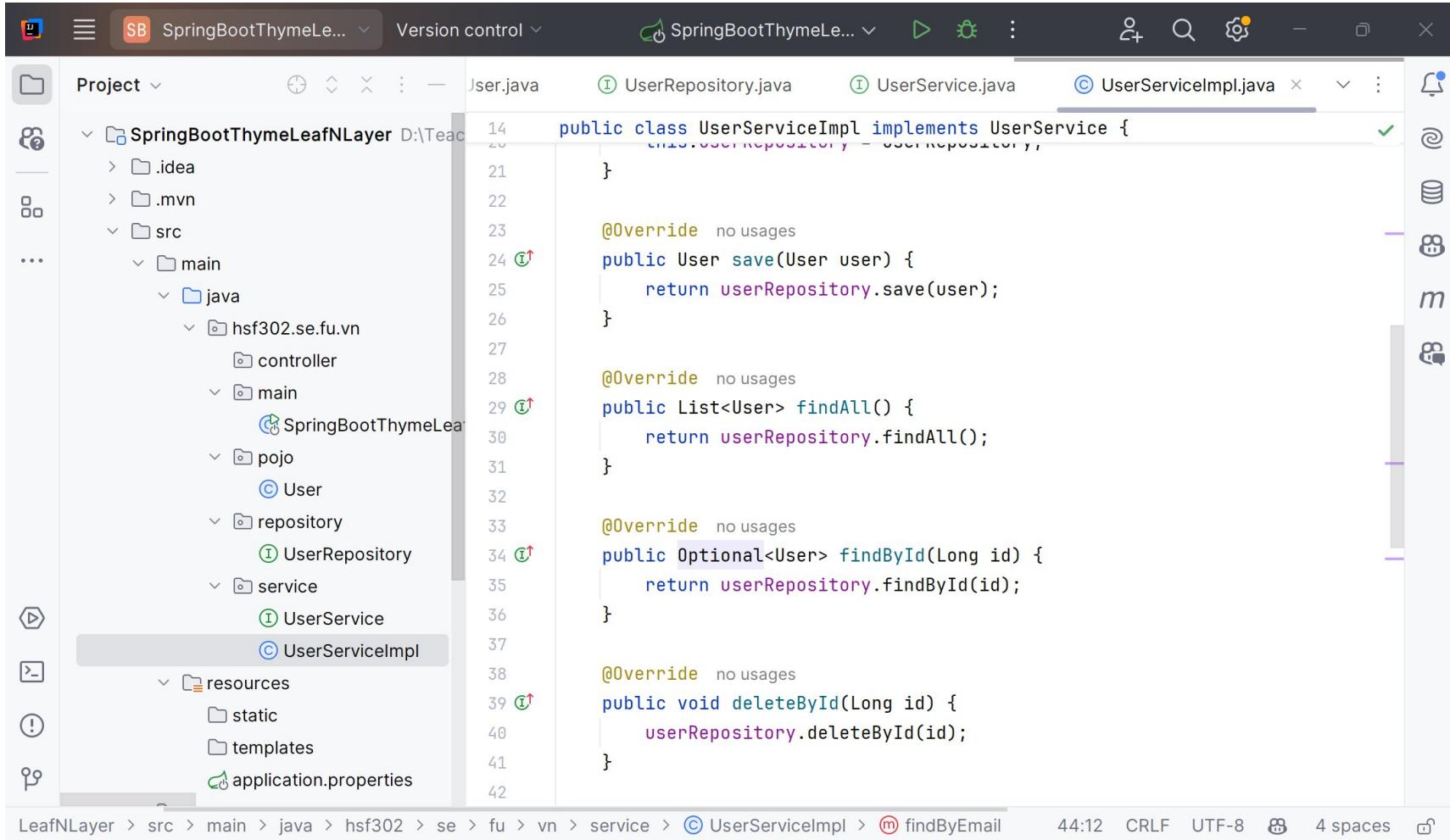
@Service
@Transactional
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;  9 usages

    @Autowired
    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override  no usages
    public User findByEmail(String email) {
        return userRepository.findByEmail(email);
    }
}
```

# Create the UserServiceImpl.java

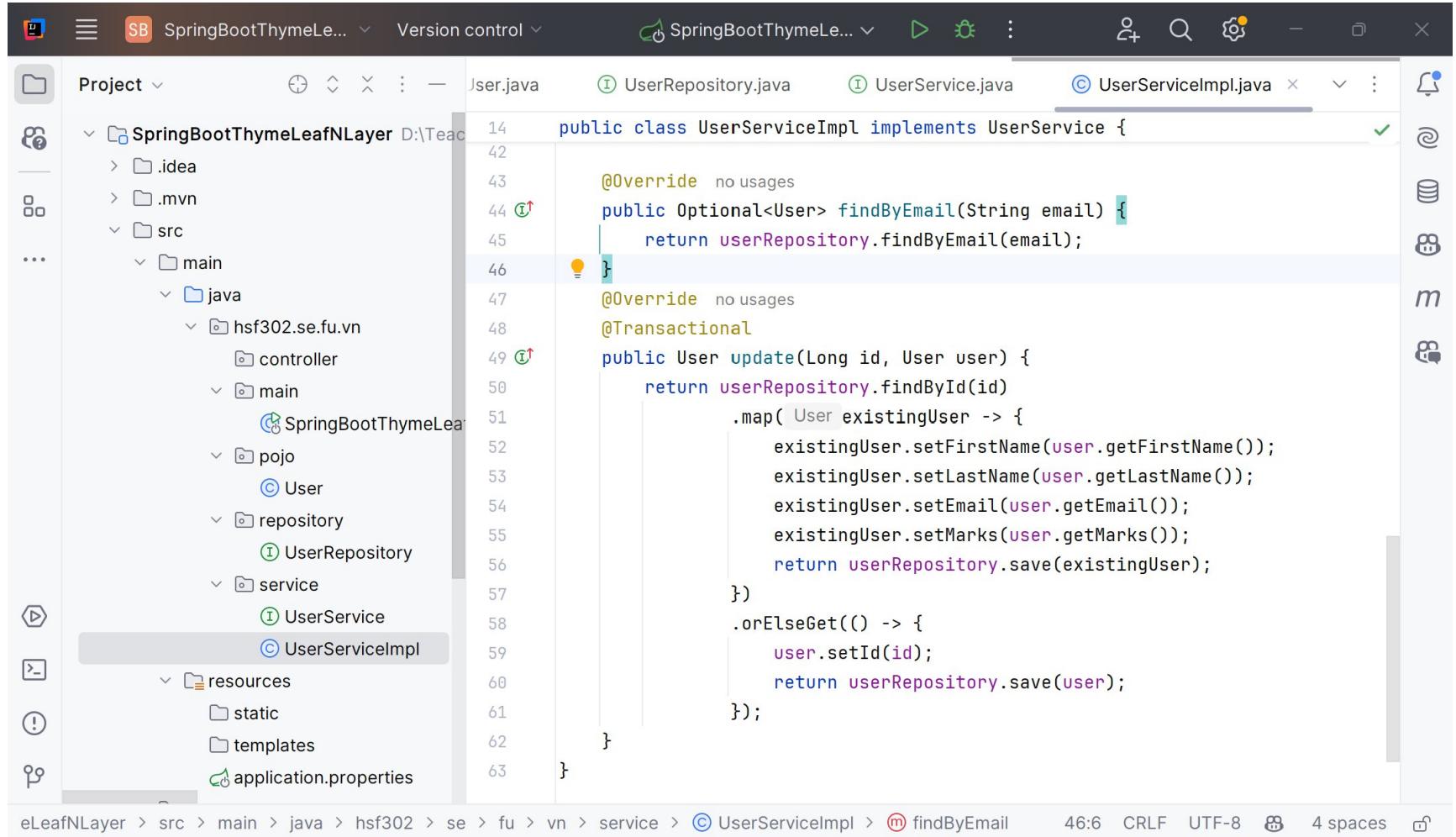


The screenshot shows the IntelliJ IDEA interface with the code editor open to the `UserServiceImpl.java` file. The project structure on the left shows a package named `SpringBootThymeLeafNLayer` containing several sub-packages like `.idea`, `.mvn`, `src`, `main`, `java`, `hsf302.se.fu.vn`, `controller`, `main`, `pojo`, `User`, `repository`, `UserRepository`, `service`, `UserService`, and `UserServiceImpl`. The `UserServiceImpl` class is selected in the project tree.

```
public class UserServiceImpl implements UserService {
    private UserRepository userRepository;
    ...
    @Override no usages
    public User save(User user) {
        return userRepository.save(user);
    }
    ...
    @Override no usages
    public List<User> findAll() {
        return userRepository.findAll();
    }
    ...
    @Override no usages
    public Optional<User> findById(Long id) {
        return userRepository.findById(id);
    }
    ...
    @Override no usages
    public void deleteById(Long id) {
        userRepository.deleteById(id);
    }
}
```

The status bar at the bottom indicates the file path as `LeafNLayer > src > main > java > hsf302 > se > fu > vn > service > C UserServiceImpl > m findByEmail`, the time as `44:12`, and encoding as `CRLF`.

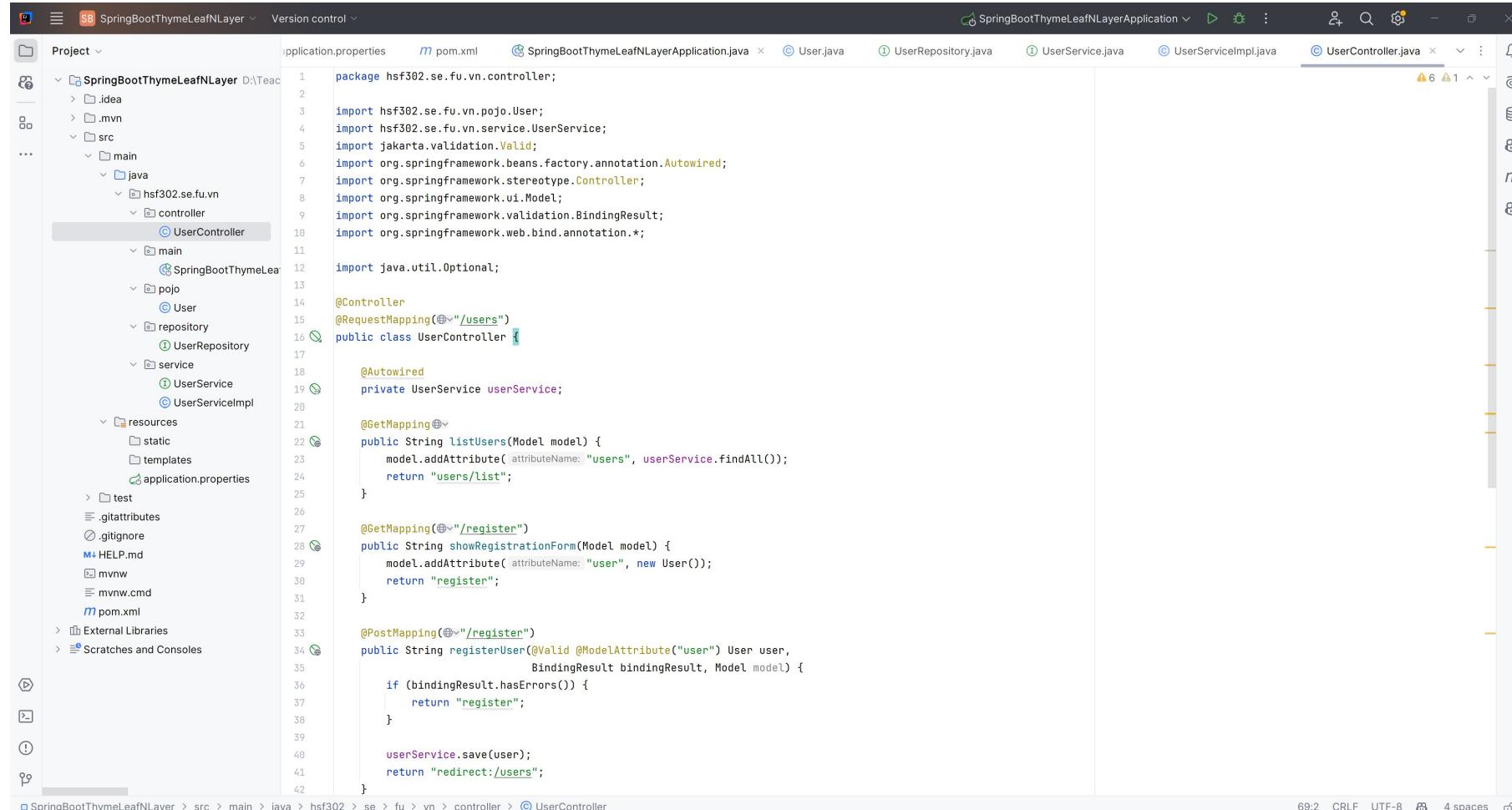
# Create the UserServiceImpl.java



The screenshot shows the IntelliJ IDEA IDE interface. The code editor displays the `UserServiceImpl.java` file, which implements the `UserService` interface. The file contains methods for finding a user by email and updating an existing user's information. The code uses `Optional` and `Transactional` annotations. The project structure on the left shows the `SpringBootThymeLeafNLayer` directory containing `.idea`, `.mvn`, `src`, `hsf302.se.fu.vn`, `controller`, `main`, `pojo`, `repository`, `UserService`, and `UserServiceImpl` files. The `application.properties` file is also visible. The status bar at the bottom provides details about the file: `eLeafNLayer > src > main > java > hsf302 > se > fu > vn > service > C UserServicelmpl > m findByEmail`, `46:6 CRLF UTF-8 4 spaces`.

```
public class UserServiceImpl implements UserService {
    @Override no usages
    public Optional<User> findByEmail(String email) {
        return userRepository.findByEmail(email);
    }
    @Override no usages
    @Transactional
    public User update(Long id, User user) {
        return userRepository.findById(id)
            .map( User existingUser -> {
                existingUser.setFirstName(user.getFirstName());
                existingUser.setLastName(user.getLastName());
                existingUser.setEmail(user.getEmail());
                existingUser.setMarks(user.getMarks());
                return userRepository.save(existingUser);
            })
            .orElseGet(() -> {
                user.setId(id);
                return userRepository.save(user);
            });
    }
}
```

# UserController



The screenshot shows the IntelliJ IDEA interface with the project structure and code editor for a Spring Boot application.

**Project Structure:**

- Project: SpringBootThymeLeafNLayer
- src/main/java/hsf302/se/fu/vn/controller/UserController.java (selected)
- src/main/java/hsf302/se/fu/vn/service/UserService.java
- src/main/java/hsf302/se/fu/vn/repository/UserRepository.java
- src/main/java/hsf302/se/fu/vn/service/UserServiceImpl.java
- src/main/resources/application.properties
- test
- .gitattributes
- .gitignore
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml
- External Libraries
- Scratches and Consoles

**UserController.java Code:**

```
package hsf302.se.fu.vn.controller;

import hsf302.se.fu.vn.pojo.User;
import hsf302.se.fu.vn.service.UserService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

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

    @Autowired
    private UserService userService;

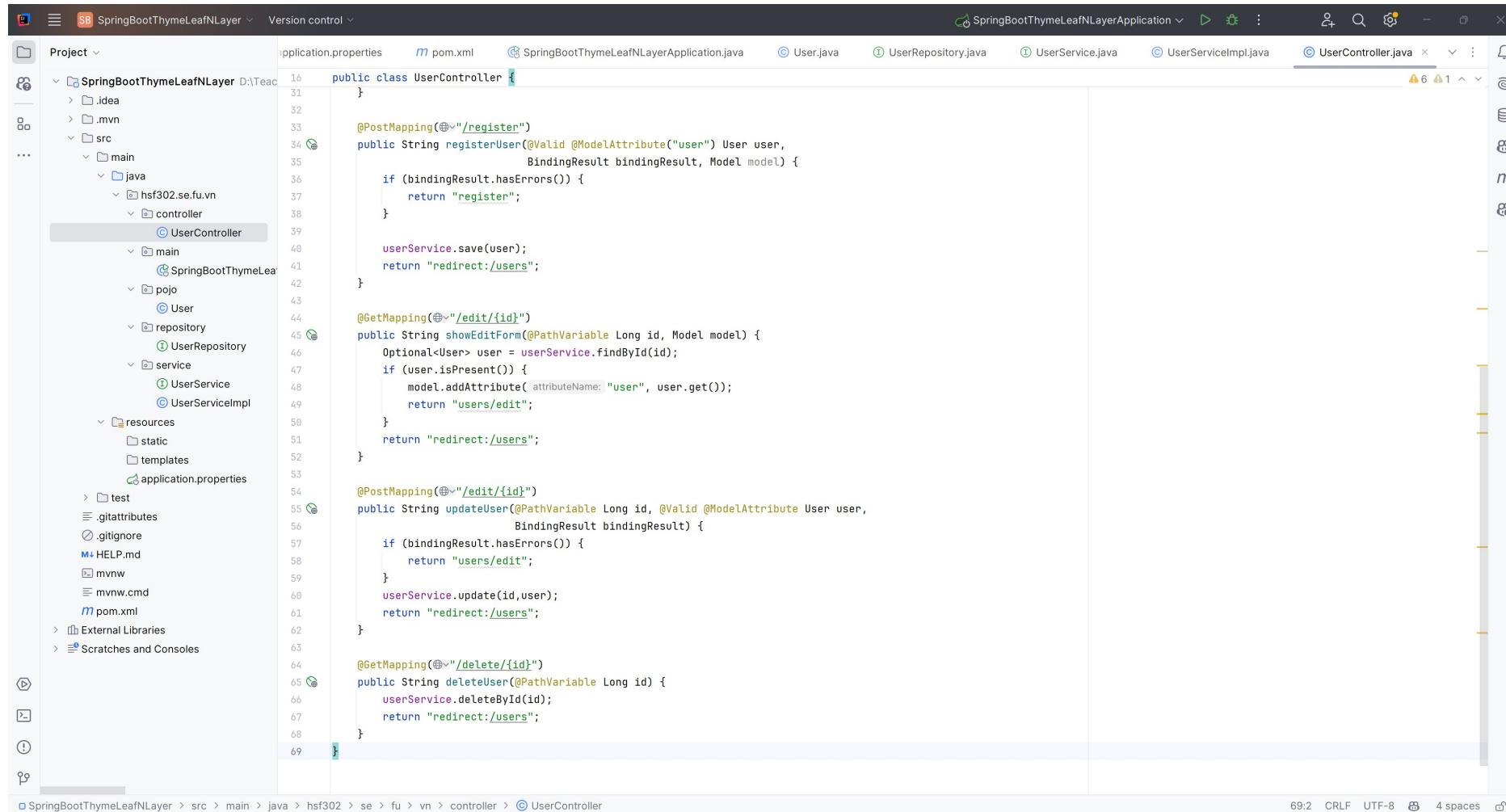
    @GetMapping("")
    public String listUsers(Model model) {
        model.addAttribute("users", userService.findAll());
        return "users/list";
    }

    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }

    @PostMapping("/register")
    public String registerUser(@Valid @ModelAttribute("user") User user,
                               BindingResult bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
            return "register";
        }

        userService.save(user);
        return "redirect:/users";
    }
}
```

# UserController



The screenshot shows the IntelliJ IDEA interface with the project 'SpringBootThymeLeafNLayer' open. The 'UserController.java' file is selected in the left navigation bar and is displayed in the main editor window. The code implements a REST controller for user management, utilizing Spring's @Controller, @GetMapping, and @PostMapping annotations along with validation annotations like @Valid and @ModelAttribute. It interacts with a UserRepository and UserService to handle register, edit, update, and delete operations.

```
public class UserController {

    @PostMapping("/register")
    public String registerUser(@Valid @ModelAttribute("user") User user,
                               BindingResult bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
            return "register";
        }

        userService.save(user);
        return "redirect:/users";
    }

    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable Long id, Model model) {
        Optional<User> user = userService.findById(id);
        if (user.isPresent()) {
            model.addAttribute("user", user.get());
            return "users/edit";
        }
        return "redirect:/users";
    }

    @PostMapping("/edit/{id}")
    public String updateUser(@PathVariable Long id, @Valid @ModelAttribute User user,
                           BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            return "users/edit";
        }
        userService.update(id, user);
        return "redirect:/users";
    }

    @GetMapping("/delete/{id}")
    public String deleteUser(@PathVariable Long id) {
        userService.deleteById(id);
        return "redirect:/users";
    }
}
```

# Create the css



The screenshot shows an IDE interface with the project structure on the left and the code editor on the right. The code editor displays a CSS file named 'gof.css' containing several button styles:

```
.error {
    color: red;
    font-size: 0.9em;
}

.form-check-label {
    font-weight: normal;
    margin-left: 5px;
}

.btn-create {
    display: inline-block;
    padding: 8px 16px;
    background-color: #2196F3;
    color: white;
    text-decoration: none;
    border-radius: 4px;
    font-weight: bold;
    margin-top: 15px;
}

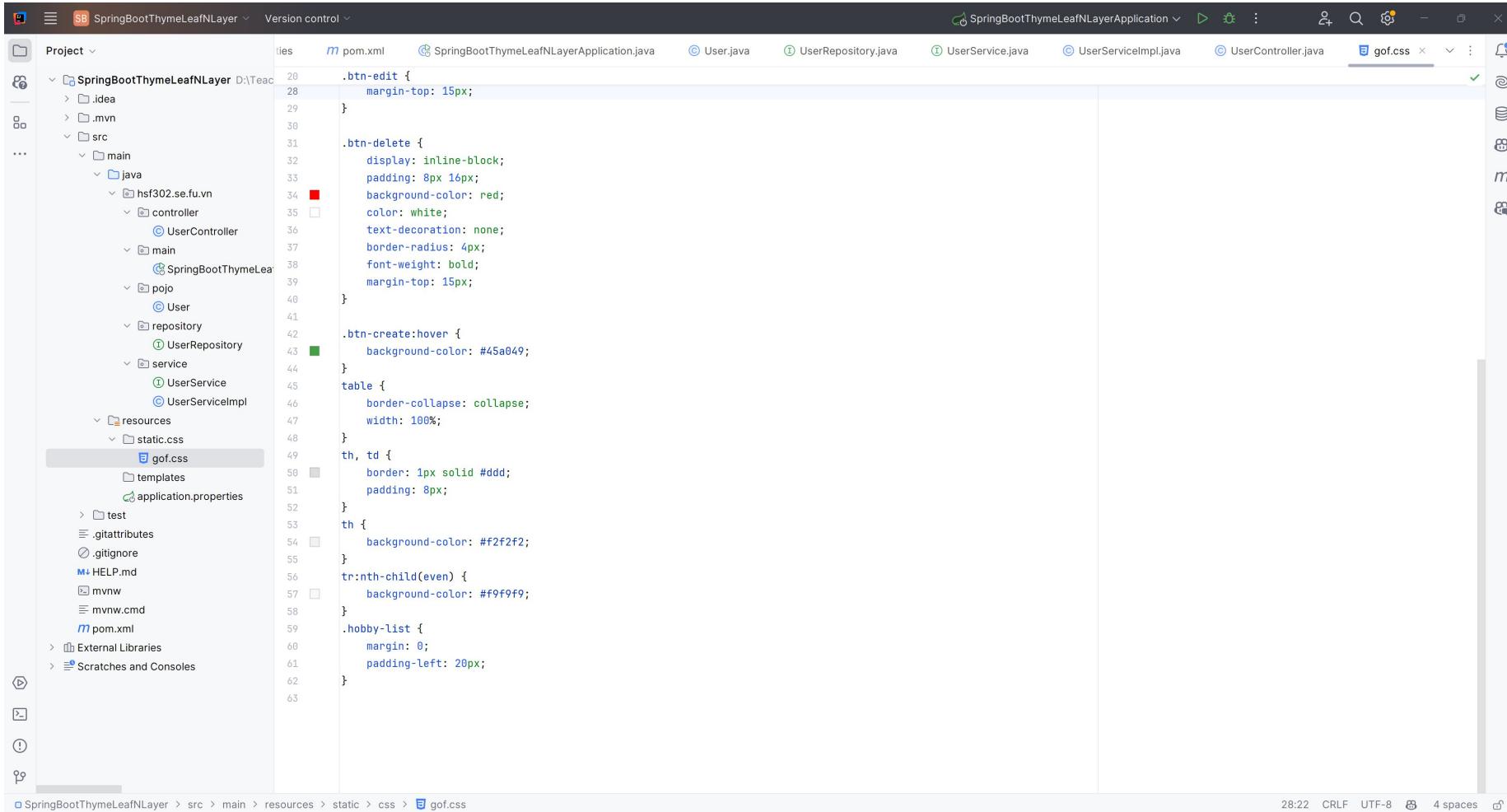
.btn-edit {
    display: inline-block;
    padding: 8px 16px;
    background-color: gold;
    color: white;
    text-decoration: none;
    border-radius: 4px;
    font-weight: bold;
    margin-top: 15px;
}

.btn-delete {
    display: inline-block;
    padding: 8px 16px;
    background-color: red;
    color: white;
    text-decoration: none;
    border-radius: 4px;
    font-weight: bold;
    margin-top: 15px;
}

.btn-create:hover {
```

The 'gof.css' file is located in the 'static/css' directory under 'src/main/resources'. Other files visible in the project tree include 'pom.xml', 'SpringBootThymeLeafNLayerApplication.java', 'User.java', 'UserRepository.java', 'UserService.java', 'UserServiceImpl.java', 'UserController.java', and 'application.properties'.

# Create the css



The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "SpringBootThymeLeafNLayer". It contains a "src" folder with "main" and "test" subfolders. "main" further contains "java", "resources", and "static.css". "resources" has "static.css" which contains "gof.css".
- Code Editor:** The "gof.css" file is open in the code editor. The code defines several CSS classes:

```
.btn-edit {
    margin-top: 15px;
}

.btn-delete {
    display: inline-block;
    padding: 8px 16px;
    background-color: red;
    color: white;
    text-decoration: none;
    border-radius: 4px;
    font-weight: bold;
    margin-top: 15px;
}

.btn-create:hover {
    background-color: #45a049;
}

table {
    border-collapse: collapse;
    width: 100%;
}

th, td {
    border: 1px solid #ddd;
    padding: 8px;
}

th {
    background-color: #f2f2f2;
}

tr:nth-child(even) {
    background-color: #f9f9f9;
}

.hobby-list {
    margin: 0;
    padding-left: 20px;
}
```

- Status Bar:** The status bar at the bottom shows the file path "SpringBootThymeLeafNLayer > src > main > resources > static > css > gof.css", the time "28:22", and encoding information "CRLF UTF-8 4 spaces".

# Create register.html

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

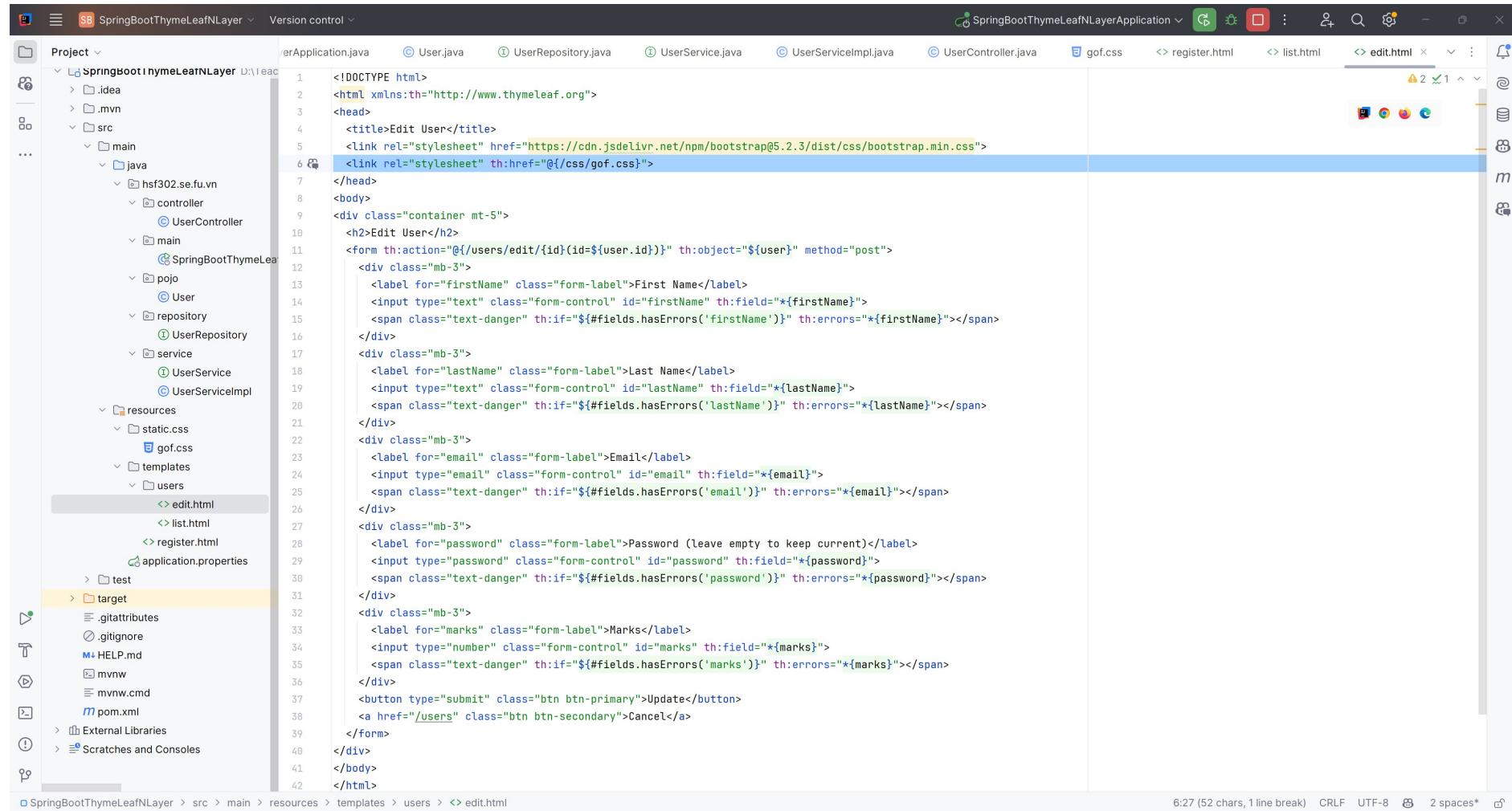
- Project Structure:** The project is named "SpringBootThymeLeafNLayer". It contains a "src" directory with "main" and "java" sub-directories. The "java" directory contains "hsf302.se.fu.vn" which has "controller", "main", "pojo", "repository", "service", and "resources" sub-directories. The "resources" directory contains "static.css", "gof.css", and "templates". The "templates" directory is currently selected, and "register.html" is highlighted.
- Code Editor:** The main editor area displays the content of "register.html". The code is a Thymeleaf template for a user registration form. It includes fields for First Name, Last Name, Email, Password, and Marks, each with validation messages. The code uses Bootstrap classes like "form-control" and "text-danger".
- Toolbars and Status Bar:** The top bar shows tabs for various files like "SpringBootThymeLeafNLayerApplication.java", "User.java", etc. The bottom status bar shows the file path "SpringBootThymeLeafNLayer > src > main > resources > templates > register.html" and the status "21:27 CRLF UTF-8 4 spaces".

# Create list.html

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

- Project Structure:** The project is named "SpringBootThymeLeafLayer". It contains a "src" directory with "main", "java", and "resources" sub-directories. "resources" contains "static.css", "gof.css", and "templates" which includes "users" (containing "edit.html" and "list.html") and "register.html".
- Code Editor:** The file "list.html" is open in the code editor. The code is a Thymeleaf template for displaying a list of users. It includes imports for Thymeleaf (th), Bootstrap CSS, and a custom CSS file "gof.css". The template features a header with a title and links, a container with a h2, a button to add a new user, and a table with columns for ID, First Name, Last Name, Email, Marks, and Actions (Edit and Delete).
- Toolbars and Status Bar:** The top bar shows tabs for various files like "SpringBootThymeLeafLayerApplication.java", "User.java", etc. The status bar at the bottom right shows "7:1 CRLF UTF-8 4 spaces".

# Create edit.html



The screenshot shows the IntelliJ IDEA interface with the project 'SpringBootThymeLeafNLayer' open. The code editor displays the 'edit.html' file, which is a Thymeleaf template for an edit user form. The template includes fields for FirstName, LastName, Email, Password, and Marks, with validation messages using the 'text-danger' class when errors occur. The project structure on the left shows the directory tree for the application, including 'src' with 'main' and 'resources' folders, and 'target' with build artifacts.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit User</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/gof.css}">
</head>
<body>
<div class="container mt-5">
    <h2>Edit User</h2>
    <form th:action="@{/users/edit/{id}(id=${user.id})" th:object="${user}" method="post">
        <div class="mb-3">
            <label for="firstName" class="form-label">First Name</label>
            <input type="text" class="form-control" id="firstName" th:field="*{firstName}">
            <span class="text-danger" th:if="#{fields.hasErrors('firstName')}" th:errors="*{firstName}"></span>
        </div>
        <div class="mb-3">
            <label for="lastName" class="form-label">Last Name</label>
            <input type="text" class="form-control" id="lastName" th:field="*{lastName}">
            <span class="text-danger" th:if="#{fields.hasErrors('lastName')}" th:errors="*{lastName}"></span>
        </div>
        <div class="mb-3">
            <label for="email" class="form-label">Email</label>
            <input type="email" class="form-control" id="email" th:field="*{email}">
            <span class="text-danger" th:if="#{fields.hasErrors('email')}" th:errors="*{email}"></span>
        </div>
        <div class="mb-3">
            <label for="password" class="form-label">Password (leave empty to keep current)</label>
            <input type="password" class="form-control" id="password" th:field="*{password}">
            <span class="text-danger" th:if="#{fields.hasErrors('password')}" th:errors="*{password}"></span>
        </div>
        <div class="mb-3">
            <label for="marks" class="form-label">Marks</label>
            <input type="number" class="form-control" id="marks" th:field="*{marks}">
            <span class="text-danger" th:if="#{fields.hasErrors('marks')}" th:errors="*{marks}"></span>
        </div>
        <button type="submit" class="btn btn-primary">Update</button>
        <a href="/users" class="btn btn-secondary">Cancel</a>
    </form>
</div>
</body>
</html>
```

# Result



The screenshot shows a web browser window titled "Users List" with the URL "localhost:8080/users". The browser interface includes standard controls like back, forward, and search, along with a sidebar for bookmarks. The main content area displays a table titled "Users List" with one row of data. The table columns are labeled "ID", "First Name", "Last Name", "Email", "Marks", and "Actions". The single data row corresponds to the user "Nguyen Lam" with email "lamnn15@gmail.com" and marks "10". The "Actions" column contains two buttons: "Edit" (yellow) and "Delete" (red).

| ID | First Name | Last Name | Email             | Marks | Actions                                       |
|----|------------|-----------|-------------------|-------|---|
| 1  | Nguyen     | Lam       | lamnn15@gmail.com | 10    | <button>Edit</button> <button>Delete</button> |

# Result



Screenshot of a web browser showing the "Edit User" form at localhost:8080/users/edit/1.

The browser window title is "Edit User". The address bar shows the URL "localhost:8080/users/edit/1". The page content is as follows:

**Edit User**

First Name: Nguyen

Last Name: Lam

Email: lamnn15@gmail.com

Password (leave empty to keep current):

Marks: 10

**Update** **Cancel**

# Result



Register User

localhost:8080/users/register

FPT DesignPattern Unity Mining STEM All Bookmarks

## Register New User

First Name

First name is required

Last Name

Last name is required

Email

Email is required

Password

Password must be at least 6 characters long

Marks

Register Cancel

# Summary

The concepts are introduced:

- ◆ Create a new Maven project in Eclipse IDE
- ◆ Add the necessary dependencies for Spring MVC and data access
- ◆ Create the Model - Define the entity classes that represent domain objects
- ◆ Define the data access object (DAO) interfaces and their implementations for interacting with the database.
- ◆ Create the Controller
- ◆ Set Up the Views
- ◆ Create the views using JSP, Thymeleaf, or another templating engine
- ◆ Implement the Business Logic
- ◆ Testing
- ◆ Run the Application