

Full Stack Web Developer

Spring Boot and H2 Database
Thymeleaf and Tailwind CSS



Full Stack Web Developer

Day 1

Section 1 Intro

- Full Stack Web Developer
- Setup Environment
 - Install JDK, Maven, VS Code & Extension
- Hello Spring Boot
 - Create Spring Project
- Project Structure
- SSR vs CSR / MVC Design Pattern
- Project Workflow

Section 2 Back-End

- Controller
- **Database Design**
- H2 Database
- Entity, Repository, Service

Day 2

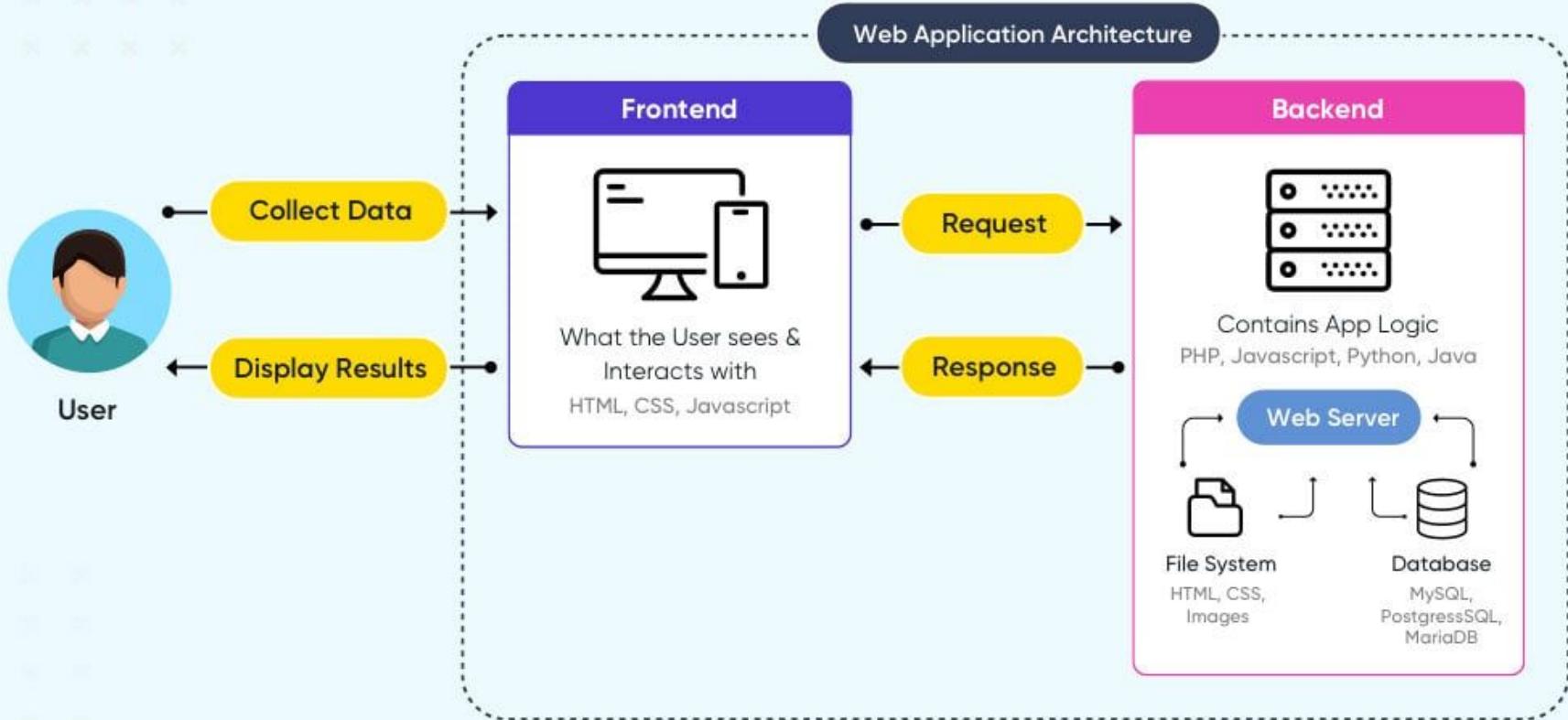
Section 3 Front-End

- **User Interface Design**
- ThymeLeaf
- Tailwind CSS

Section 4 Workshop

- Build Spring Boot Application
- Workshop
 - Todo List Application

Web Application



net solutions

<https://www.netsolutions.com/insights/web-application-architecture-guide/>

What is Full Stack ?

Full-stack Web Developer

A full-stack web developer is a person who can develop both client and server software.

In addition to mastering HTML and CSS, he/she also knows how to:

- Program a client (e.g. using JavaScript, jQuery, Angular, or Vue)
- Program a server (e.g. using JAVA, .NET, PHP, Python, or Node)
- Program a database (e.g. using SQL, SQLite, or MongoDB)

What is Full Stack ?

Client Software (Front-End)

- HTML
- CSS
 - Bootstrap, Tailwind CSS
- JavaScript
 - JSON
 - jQuery
 - Backbone.js
- Angular
- React

Server Software (Back-End)

- JAVA
 - Spring Boot
- PHP
- .NET
- Python
- Node.js

Database

- SQL
 - H2 Database
- NoSQL
 - MongoDB

Other

- Requirement
- Planning
- User Interface Design
- Database Design
- Other ? (Linux, Docker, Git, CI/CD, etc)

Project Workflow (Example)

1. Requirement

Requirement

- Student Management (Section 1 - 3)
 - User can manage student data
 - student code, first name, last name
 - User can manage faculty data
 - faculty name
 - User can manage course data, User can manage student in course
 - course name
 - course description
- Todo List - Workshop (Section 4)

2. Planning

A	B	C	D	E	Day 1					Day 2				
					13:00	14:00	15:00	16:00	17:00	8:00	9:00	10:00	11:00	0:00
1	Project Setup			15 Minutes	13:00									
2	Requirement			15 Minutes	13:15									
3	User Interface Design			15 Minutes	13:30									
4	Database Design			15 Minutes	13:45									
5	Coding Back End			2 Hours	14:00									
6	Testing			15 Minutes	16:00									
7	Coding Front End			3 Hours	8:00									
8	Testing			15 Minutes	11:00									

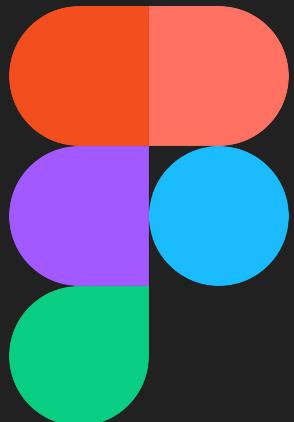


Google Sheets

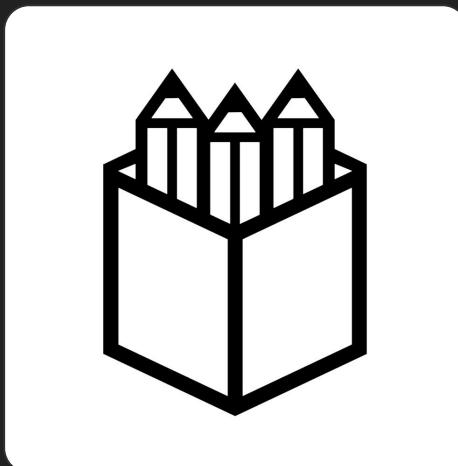


Office Excel

3. User Interface - Wireframe



Figma



Penpot



Draw.IO

3. User Interface - Faculty (Wireframe)



The wireframe shows a user interface for managing faculty. At the top, there is a navigation bar with tabs: Student Management, Faculty (which is selected), Student, Course, and Enroll. Below the navigation bar, there is a form for editing a faculty record. The form includes fields for Faculty Id (containing '1') and Faculty Name (containing 'Computer Science'). At the bottom right of the form are two buttons: 'Delete' (red) and 'Save' (blue). Below the form is a table listing faculty records.

ID	Name	Action
1	Computer Science	[View]
2	Multimedia Technology	[View]
3	Software Engineer	[View]
4	Computer Engineer	[View]

3. User Interface - Student (Wireframe)



Student Management Faculty **Student** Course Enroll

Student Id	1	Student Code	673001	
First Name	Shelli	Last Name	Kan	
Faculty	Computer Engineer			
Delete Save				
ID	Student Code	First Name	Last Name	Action
1	673001	Shelli	Kann	[View]
2	673002	Christie	Salsbury	[View]
3	673003	Shepperd	Skyner	[View]
4	673004	Jermaine	Fraanchyonok	[View]

3. User Interface - Course (Wireframe)



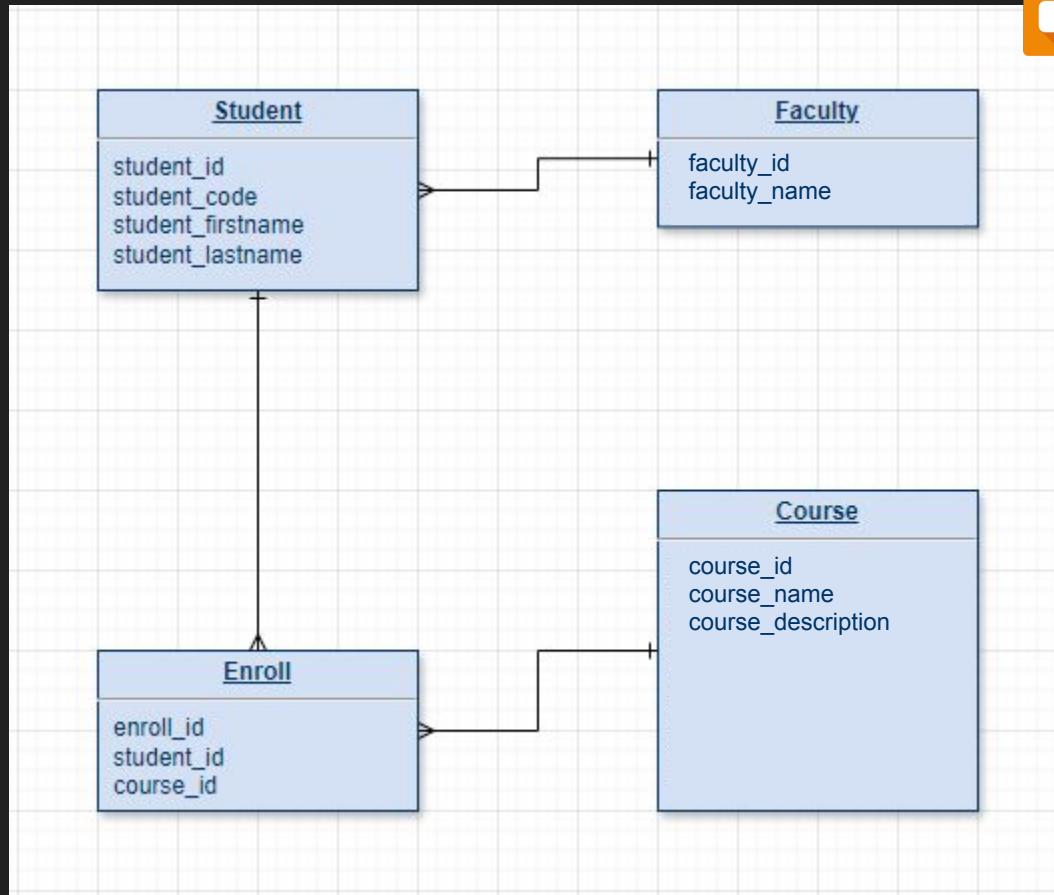
The wireframe shows a user interface for managing courses. At the top, there is a navigation bar with tabs: Student Management, Faculty, Student, Course (which is highlighted in black), and Enroll. Below the navigation bar is a form for editing a course record. The form fields are: Course Id (value: 1), Course Name (value: Java Programming), and Course Description (value: Java Language). To the right of the form are two buttons: a red "Delete" button and a blue "Save" button. At the bottom of the screen is a table listing all courses.

ID	Name	Action
1	Java Programming	[View]
2	Operating System	[View]
3	Computer	[View]
4	Law	[View]

3. User Interface - Enroll (Wireframe)



4. Database - ER Diagram (Entity Relationship)



Understand GET and POST

The Method Attribute

The `method` attribute specifies the HTTP method to be used when submitting the form data.

The form-data can be sent as URL variables (with `method="get"`) or as HTTP post transaction (with `method="post"`).

The default HTTP method when submitting form data is GET.

Example

This example uses the GET method when submitting the form data:

```
<form action="/action_page.php" method="get">
```

[Try it Yourself »](#)

Example

This example uses the POST method when submitting the form data:

```
<form action="/action_page.php" method="post">
```

[Try it Yourself »](#)

https://www.w3schools.com/html/html_forms_attributes.asp

API - Faculty

Student Management Faculty Student Course Enroll

Faculty Id
1

Faculty Name
Computer Science

Delete Save

ID	Name	Action
1	Computer Science	[View] 1
2	Multimedia Technology	[View] 2
3	Software Engineer	[View]
4	Computer Engineer	[View]

API - Faculty

(1) getAll

method **GET** - /faculty

(2) getById

method **GET** - /faculty/{faculty-id}

(3) insertFaculty, updateFaculty

method **POST**

- insert - /faculty/
- update - /faculty/{faculty-id}

(4) deleteFaculty

method **GET** - /faculty/delete/{faculty-id}

API - Student

Student Management Faculty **Student** Course Enroll

Student Id		Student Code		
1		673001		
First Name	Shelli	Last Name	Kan	
Faculty	Computer Engineer			
Delete Save				
ID	Student Code	First Name	Last Name	Action
1	673001	Shelli	Kann	[View]
2	673002	Christie	Salsbury	[View]
3	673003	Shepperd	Skyner	[View]
4	673004	Jermaine	Fraanchyonok	[View]

API - Student

(1) getAll

method **GET** - /student

(2) getById

method **GET** - /student/{student-id}

(3) insertStudent, updateStudent

method **POST**

- insert - /student/
- update - /student/{student-id}

(4) deleteStudent

method **GET** - /student/delete/{student-id}

API - Course

The screenshot shows a web-based course management system. At the top, there is a navigation bar with tabs: Student Management, Faculty, Student, Course (which is currently selected), and Enroll. Below the navigation bar, there is a form for adding or editing a course. The form fields are:

- Course Id: 1
- Course Name: Java Programming
- Course Description: Java Language

At the bottom right of the form are two buttons: a red "Delete" button and a blue "Save" button. A red circle with the number 2 points to the "Save" button.

Below the form is a table listing courses. The table has columns: ID, Name, and Action. The data in the table is:

ID	Name	Action
1	Java Programming	[View]
2	Operating System	[View]
3	Computer	[View]
4	Law	[View]

A red circle with the number 1 points to the first row of the table. A red circle with the number 3 points to the "Action" column header. A red circle with the number 4 points to the "Delete" button in the form.

API - Course

(1) getAll

method **GET** - /course

(2) getById

method **GET** - /course/{course-id}

(3) insertCourse, updateCourse

method **POST**

- insert - /course/
- update - /course/{course-id}

(4) deleteCourse

method **GET** - /course/delete/{course-id}

API - Enroll

Diagram illustrating the API - Enroll process flow:

The process starts at the **Enroll** tab in the Student Management system.

- Step 1:** Select the student (Shelli Kann) and course (Java Programming).
- Step 2:** Click the **Save** button to save the enrollment.
- Step 3:** The enrollment is added to the **Student List**.
- Step 4:** The **Delete** button is used to remove the enrollment from the list.
- Step 5:** The enrollment is listed in the **Enrollment** table.
- Step 6:** The **[View]** link is clicked to view the details of the enrollment.

Student Management

Enroll Id: 1

Student: Shelli Kann

Course: Java Programming

Save

Student List

ID	Student Code	First Name	Last Name	Action
1	673001	Shelli	Kann	[Course]
2	673002	Christie	Salsbury	[Course]
3	673003	Shepperd	Skyner	[Course]
4	673004	Jermaine	Fraanchyonok	[Course]

Enrollment

ID	Name	Action
1	Java Programming	[View]
2	Operating System	[View]
3	Computer	[View]
4	Law	[View]

API - Enroll

(1) getAll

method **GET** - /enroll

(2) getById

method **GET** - /enroll/{enroll-id}

(3) insertEnroll, updateEnroll

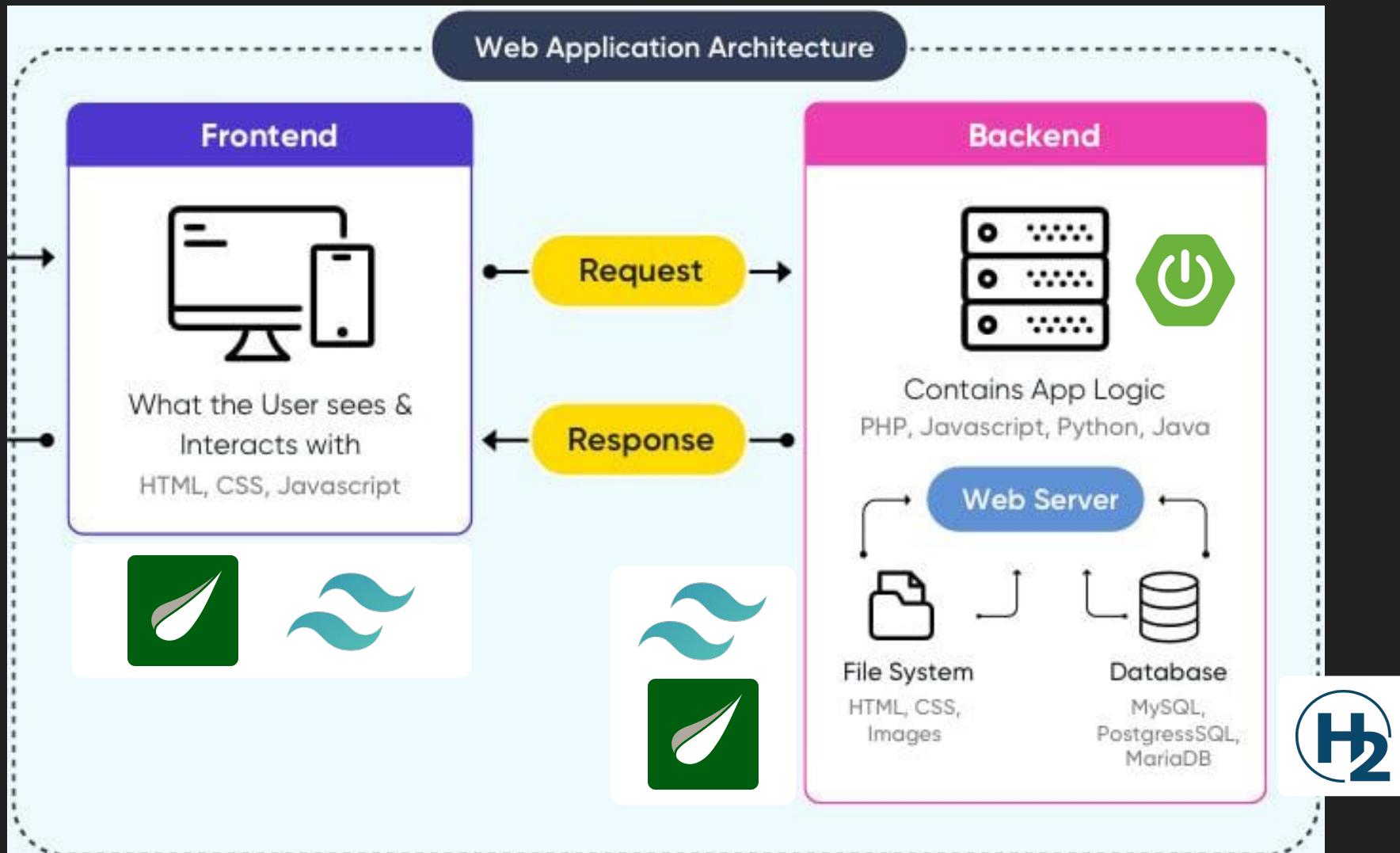
method **POST**

- insert - /enroll/
- update - /enroll/{enroll-id}

(4) deleteEnroll

method **GET** - /enroll/delete/{enroll-id}

Technologies



<https://www.netsolutions.com/insights/web-application-architecture-guide/>

Setup Environment

- [Install OpenJDK 21](#)
- [Install Maven 3](#)
- [Install Visual Studio Code & Extension](#)

Install OpenJDK 21

Install OpenJDK 21

The screenshot shows the Adoptium website (adoptium.net) with several numbered callouts:

- Callout 1: A red box highlights the "adoptium.net" logo in the top navigation bar.
- Callout 2: A red box highlights the "Latest LTS Release" button for Java 21 on Windows x64.

ADOPTIUM Home Marketplace Documentation FAQ Projects Further Information

adoptium.net

Prebuilt OpenJDK Binaries for Free!

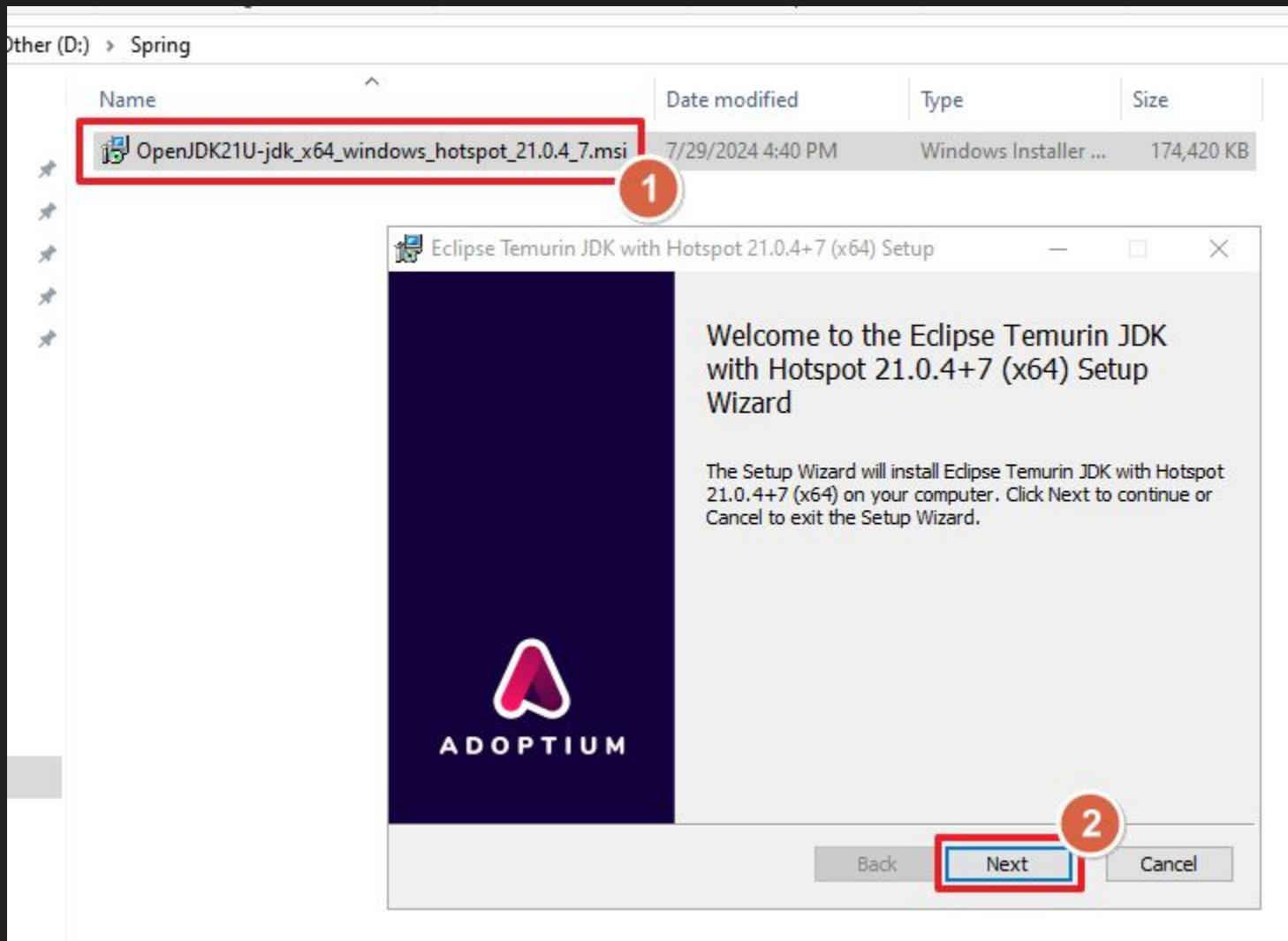
Java™ is the world's leading programming language and platform. The Adoptium Working Group promotes and supports high-quality, TCK certified runtimes and associated technology for use across the Java ecosystem. Eclipse Temurin is the name of the OpenJDK distribution from Adoptium.

Download Temurin™ for Windows x64

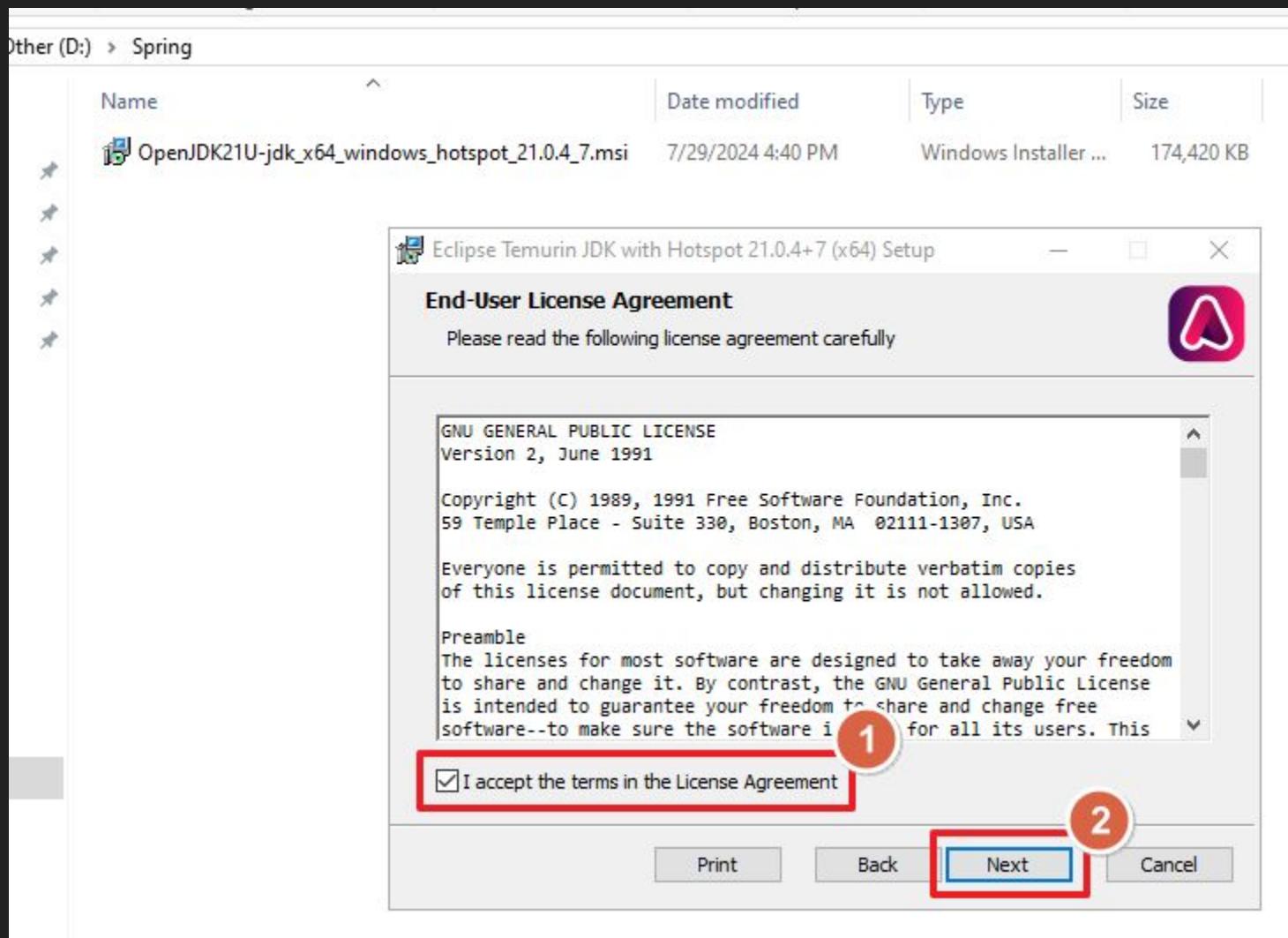
[Latest LTS Release
jdk-21.0.4+7](#)

Other platforms and versions ↗

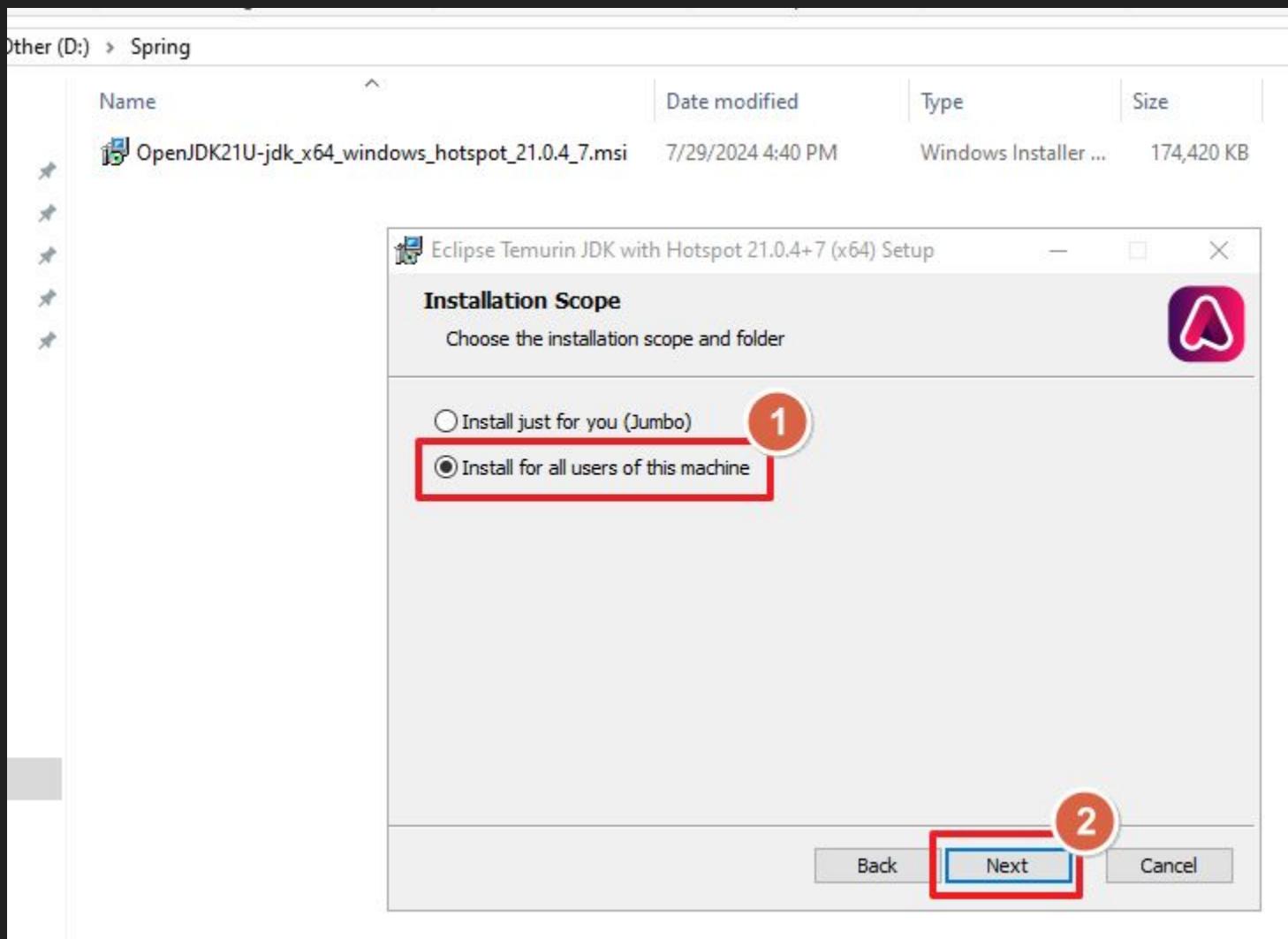
Install OpenJDK 21



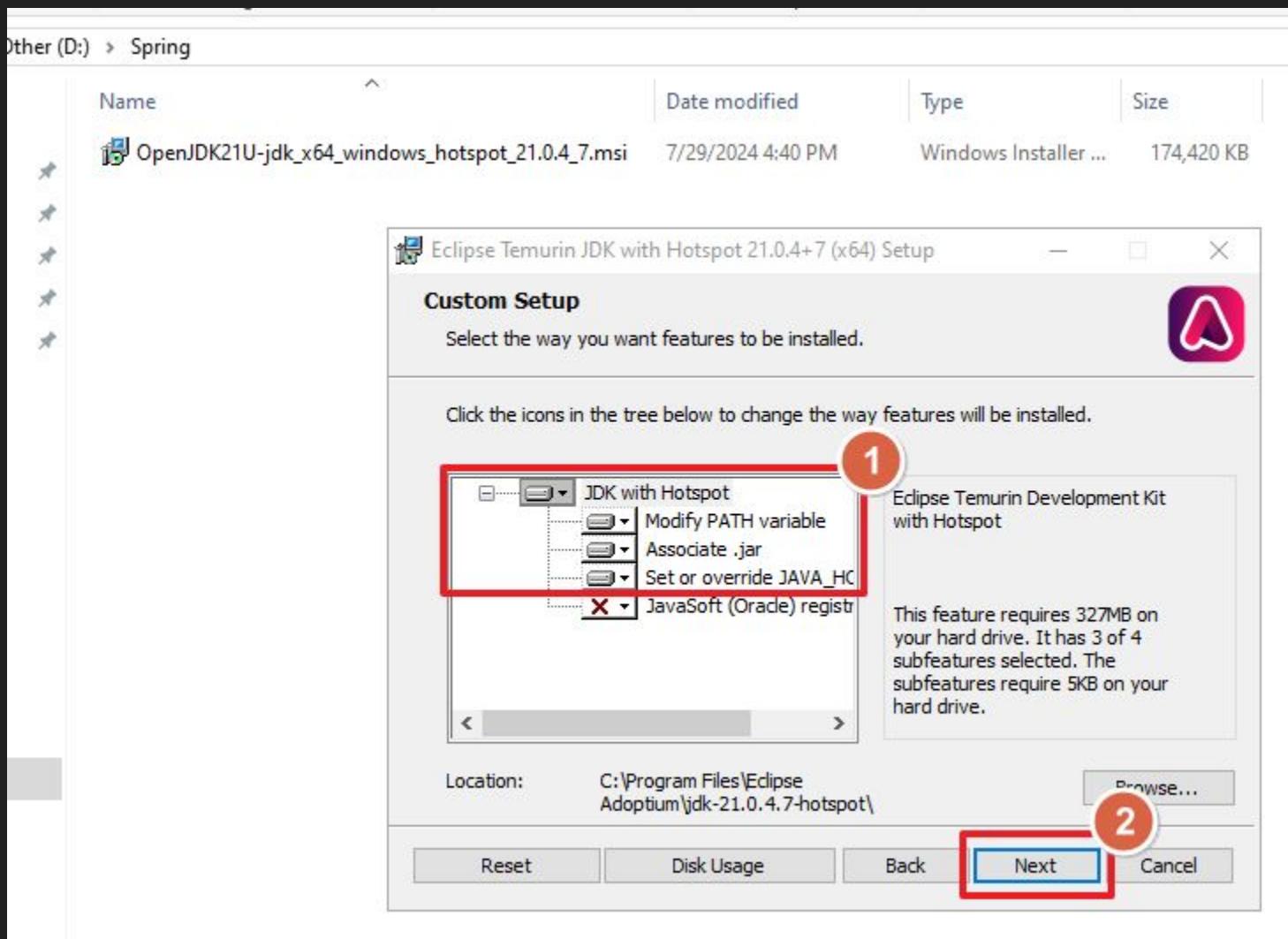
Install OpenJDK 21



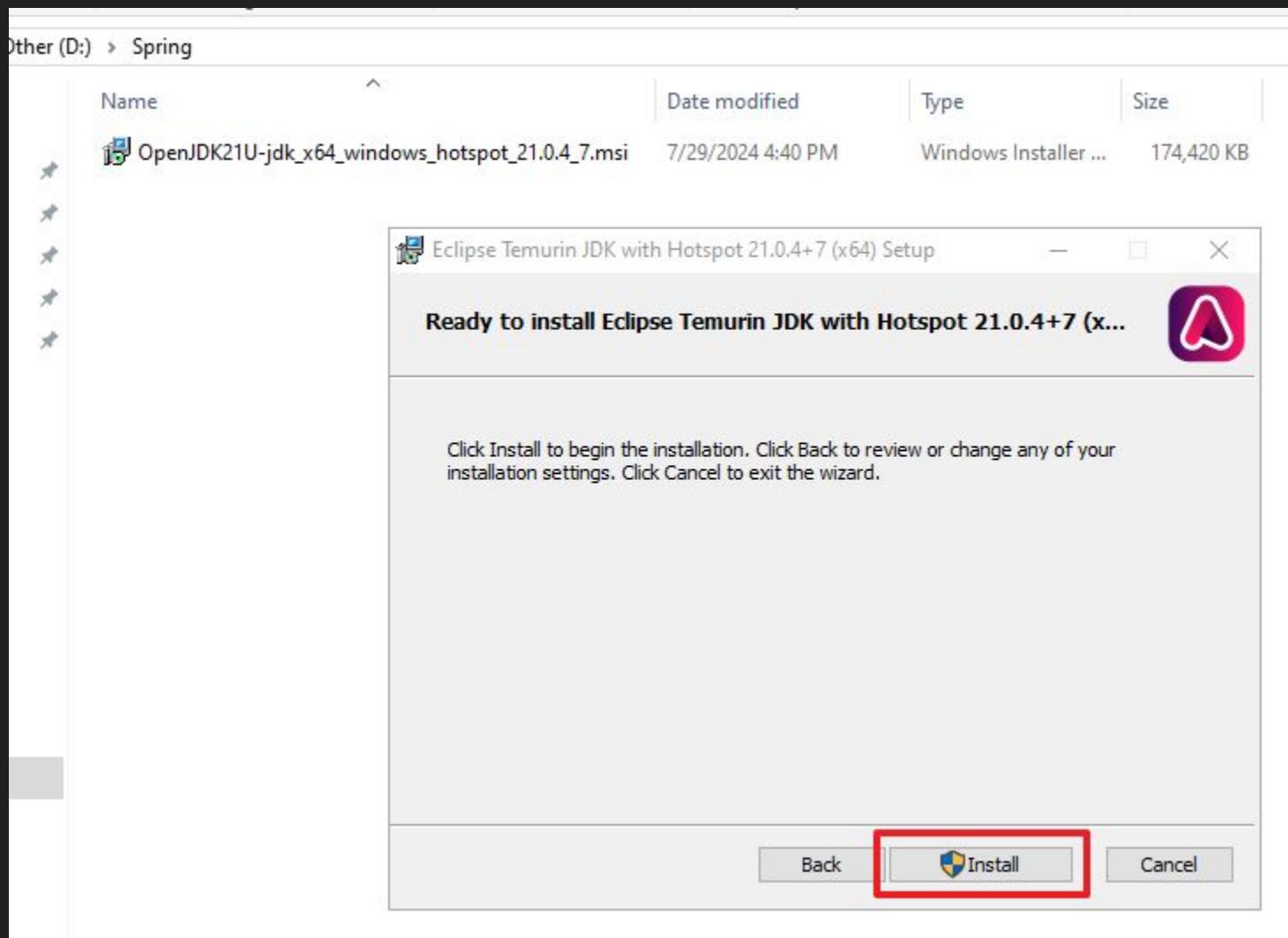
Install OpenJDK 21



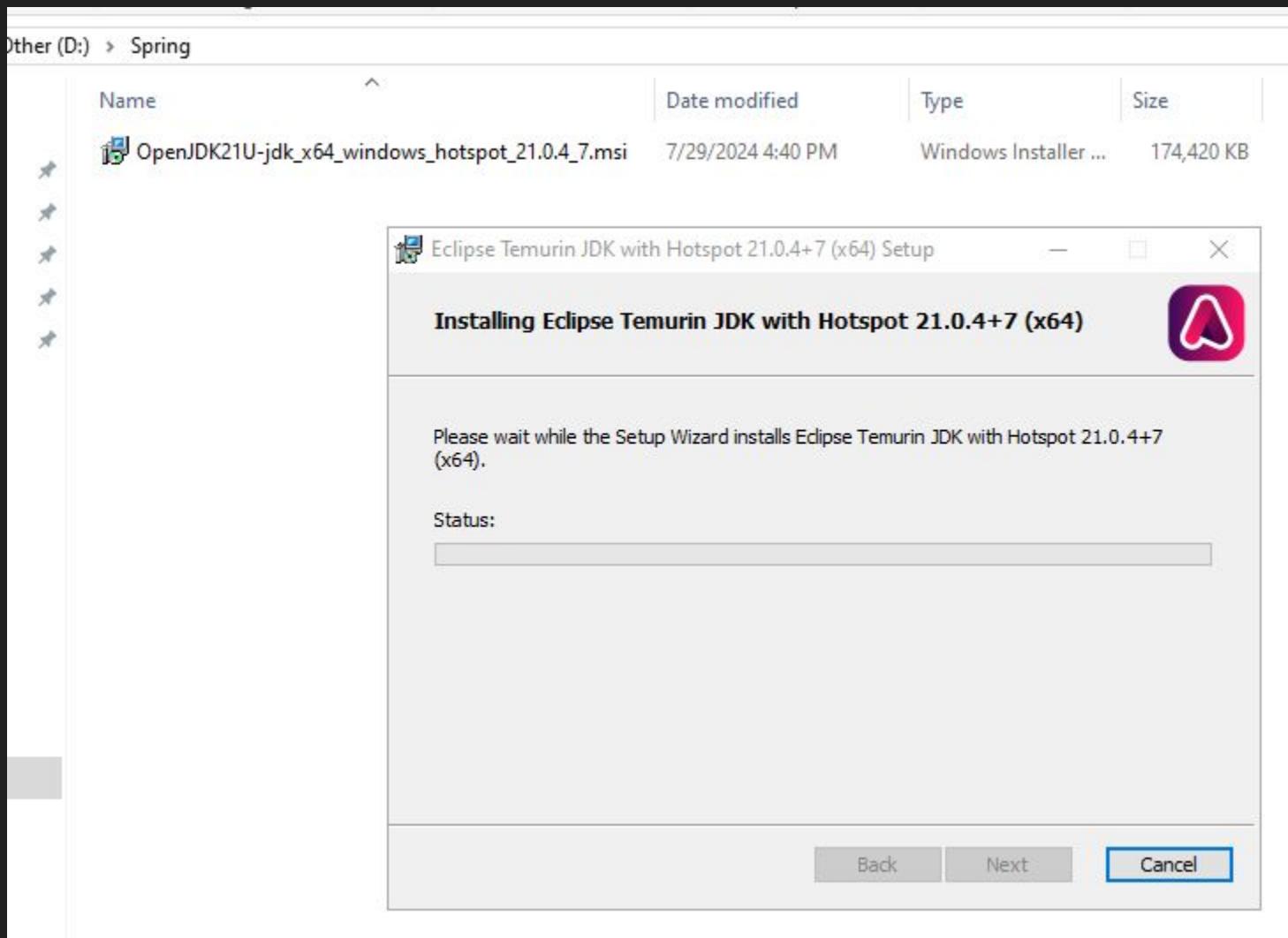
Install OpenJDK 21



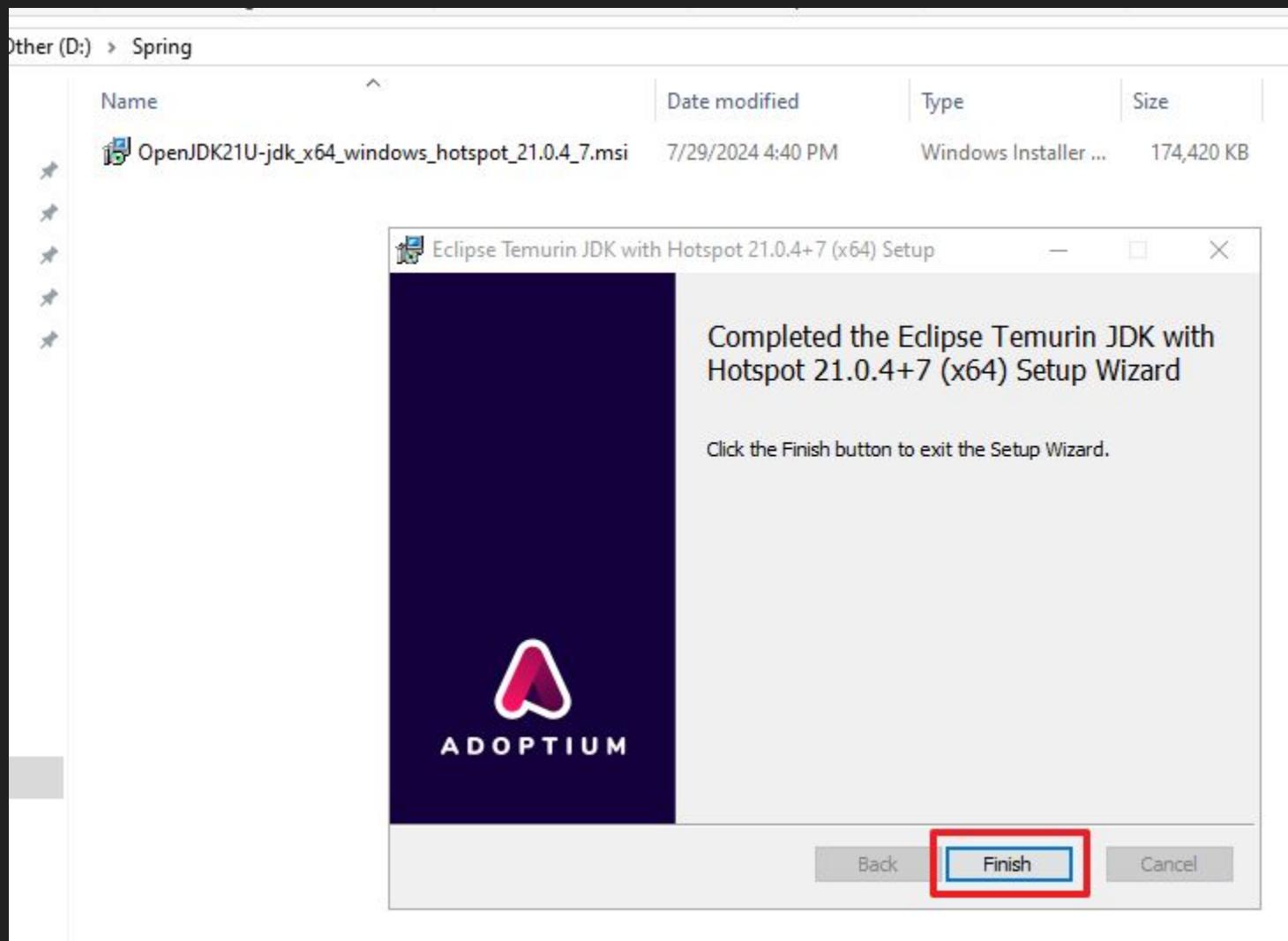
Install OpenJDK 21



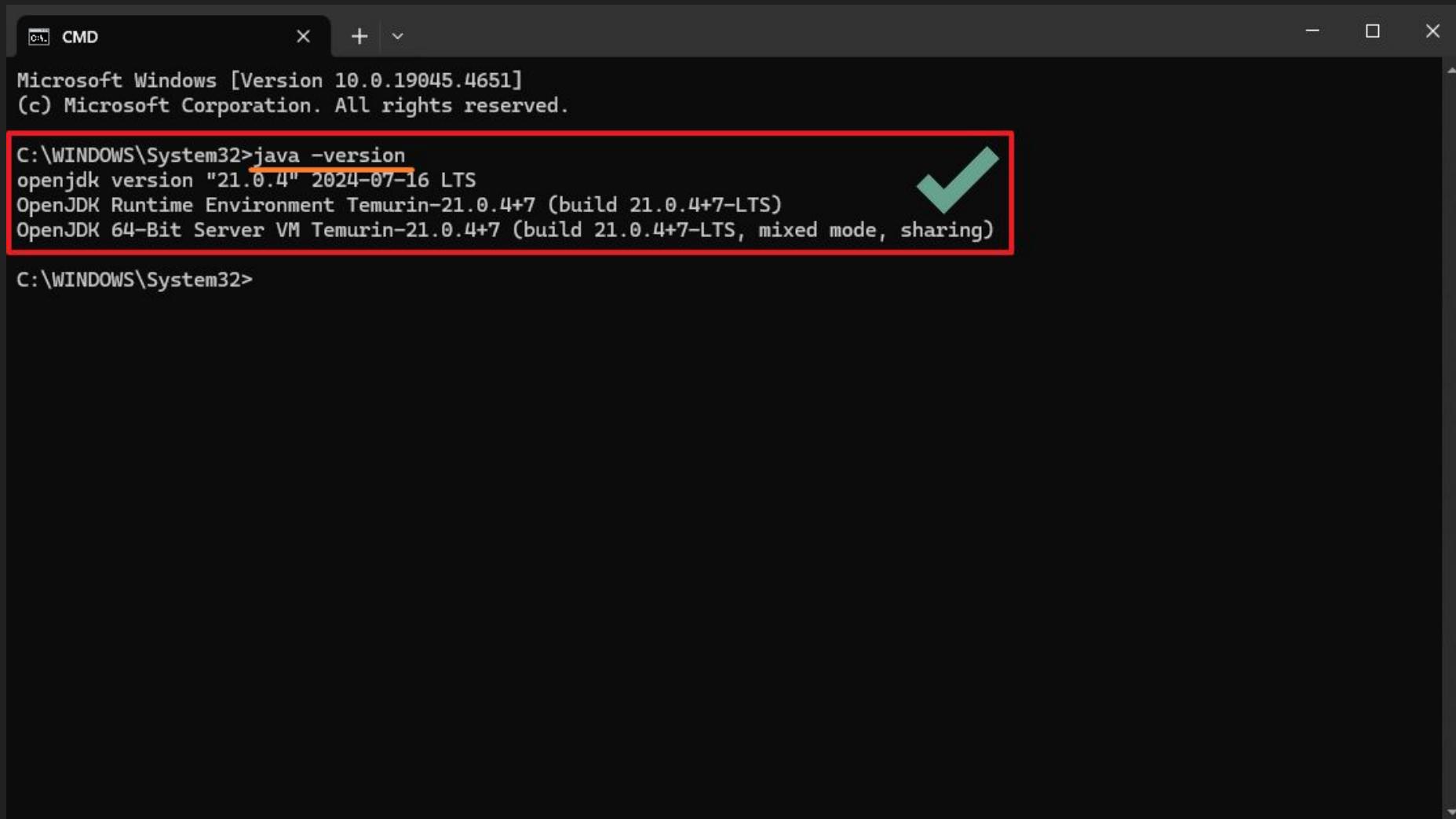
Install OpenJDK 21



Install OpenJDK 21



Install OpenJDK Complete



A screenshot of a Windows Command Prompt (CMD) window titled "CMD". The window shows the system information and the result of running the "java -version" command. The output is as follows:

```
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\System32>java -version
openjdk version "21.0.4" 2024-07-16 LTS
OpenJDK Runtime Environment Temurin-21.0.4+7 (build 21.0.4+7-LTS)
OpenJDK 64-Bit Server VM Temurin-21.0.4+7 (build 21.0.4+7-LTS, mixed mode, sharing)
```

The entire command output area is highlighted with a red rectangular box. To the right of the box, there is a large green checkmark icon.

Install Maven 3

Install Maven 3

The screenshot shows the Apache Maven Project website at <http://maven.apache.org/>. A red box highlights the URL in the browser's address bar, and a red circle with the number '1' is placed over it. Another red box highlights the 'Download, Install, Configure, Run Maven' link under the 'Use' section, and a red circle with the number '2' is placed over it.

maven.apache.org

Apache Maven Project

http://maven.apache.org/

aven / Welcome to Apache Maven

maven.apache.org

Welcome to Apache Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build process, reporting and artifacts.

If you think that Maven could help your project, you can find out more information in the "About Maven" section of the navigation. This site is separated into the following sections, depending on how you'd like to use Maven:

Use	Extend	Contribute	More
Download, Install, Configure, Run Maven Information for people needing to build a project that uses Maven	Write Maven Plugins Information for developers writing Maven plugins.	Help Maven Information if you'd like to get involved. Maven is an open source community and welcomes contributions.	Maven Plugins Lists of available Maven Plugins
Apache Maven The official Apache Maven site	Apache Plugins Information about Apache Plugins	Apache Dev Guide Information about the Apache Dev Guide	Apache Site Help Information about the Apache Site Help

Each guide is divided into a number of trails to get you started on a particular topic, and includes a reference area and a "cookbook" of common tasks.

Install Maven 3

Apache Maven 3.9.8 is the latest release: it is the recommended version for all users.

System Requirements

Java Development Kit (JDK)	Maven 3.9+ requires JDK 8 or above to execute. It still allows you to build against 1.3 and other JDKs.
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, disk space will be at least 500MB.
Operating System	No minimum requirement. Start up scripts are included as shell scripts (tested on many Unix flavors).

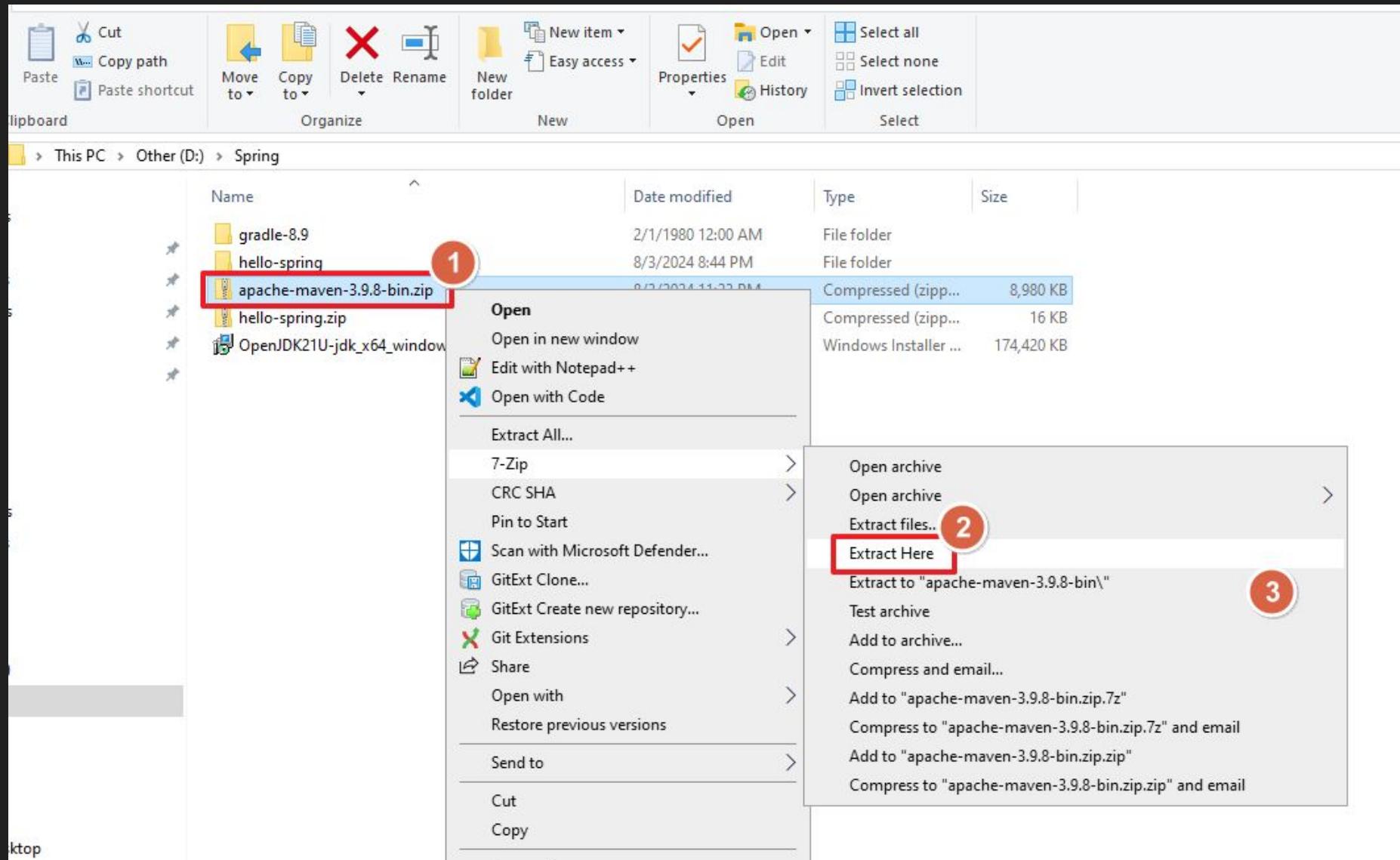
Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [installation instructions](#). In order to guard against corrupted downloads/installations, it is highly recommended to [verify the signature](#) of the release bundles against the checksums listed below.

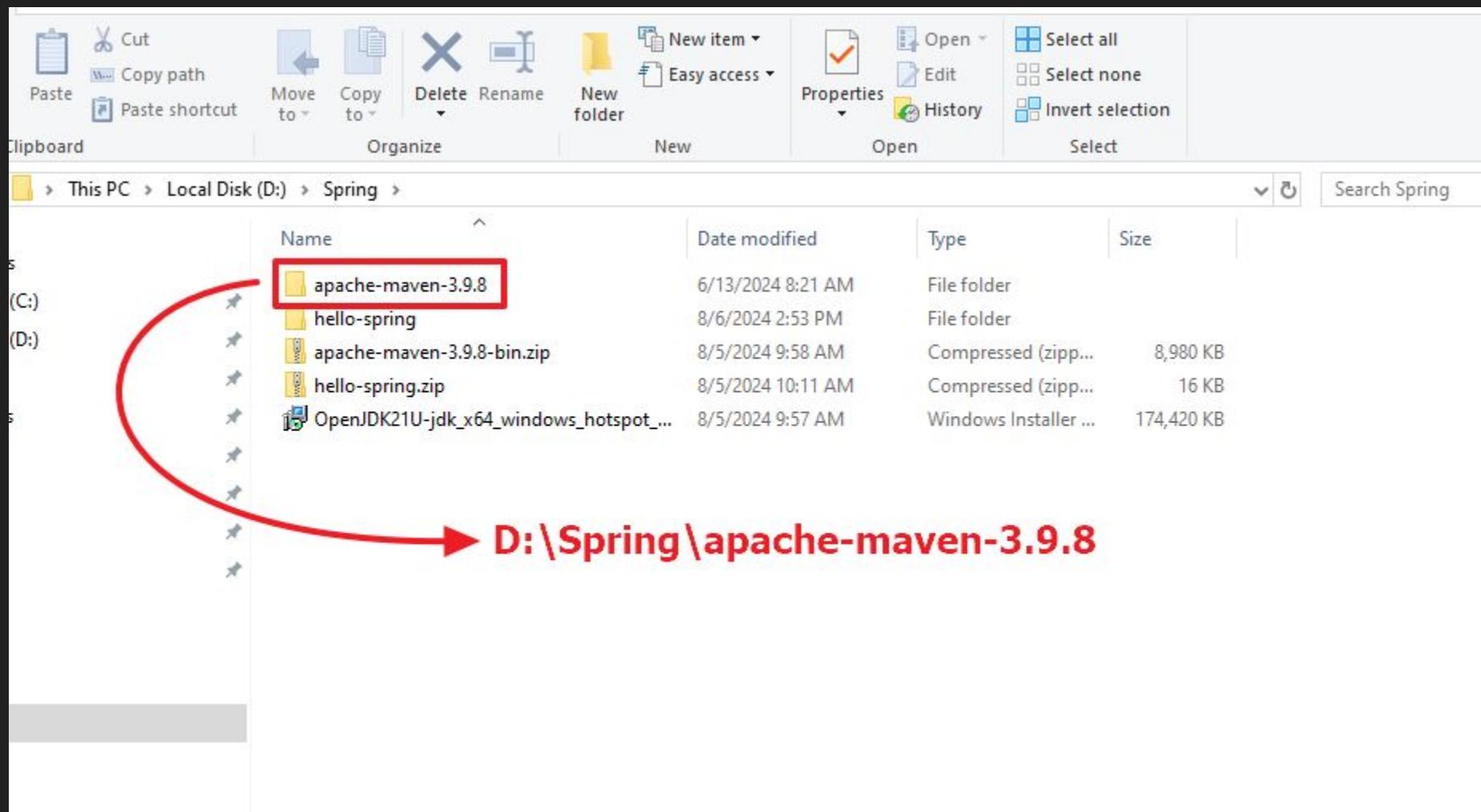
	Link	Checksums
Binary tar.gz archive	apache-maven-3.9.8-bin.tar.gz	apache-maven-3.9.8-bin.tar.gz
Binary zip archive	apache-maven-3.9.8-bin.zip	apache-maven-3.9.8-bin.zip
Source tar.gz archive	apache-maven-3.9.8-src.tar.gz	apache-maven-3.9.8-src.tar.gz
Source zip archive	apache-maven-3.9.8-src.zip	apache-maven-3.9.8-src.zip

- 3.9.8 [Release Notes](#) and [Release Reference Documentation](#)
- [latest source code from source repository](#)
- Distributed under the [Apache Licence, version 2.0](#)

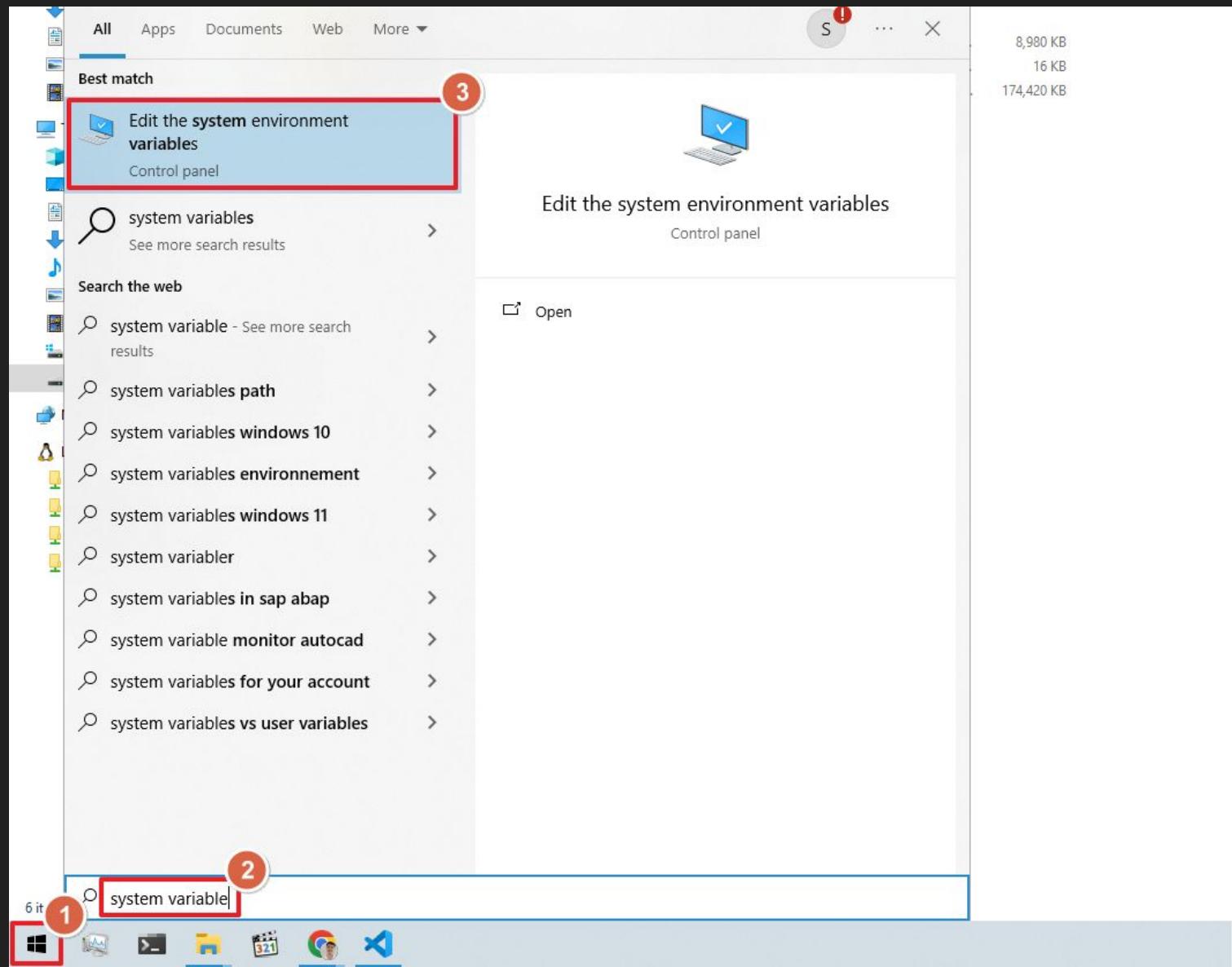
Install Maven 3



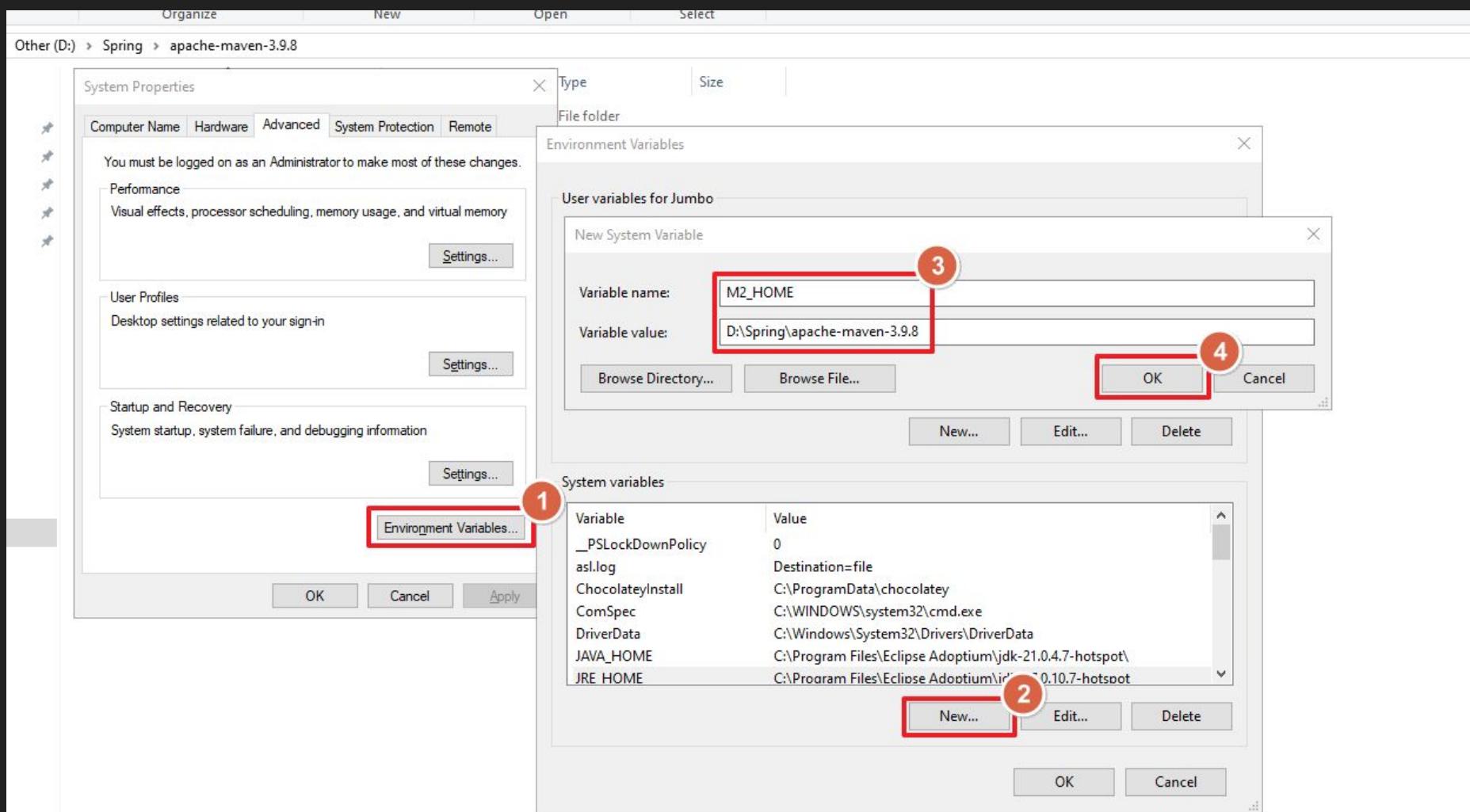
Install Maven 3



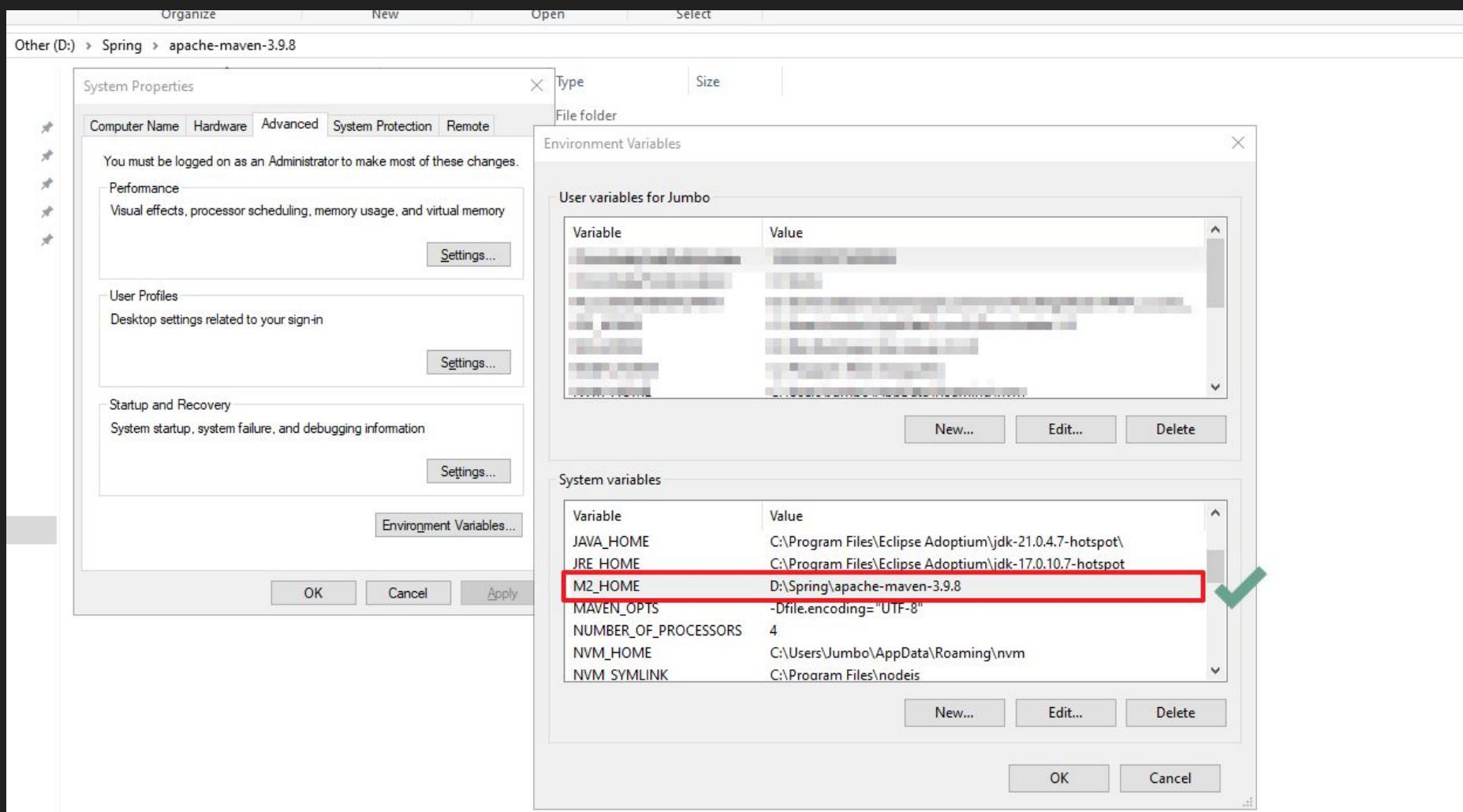
Install Maven 3



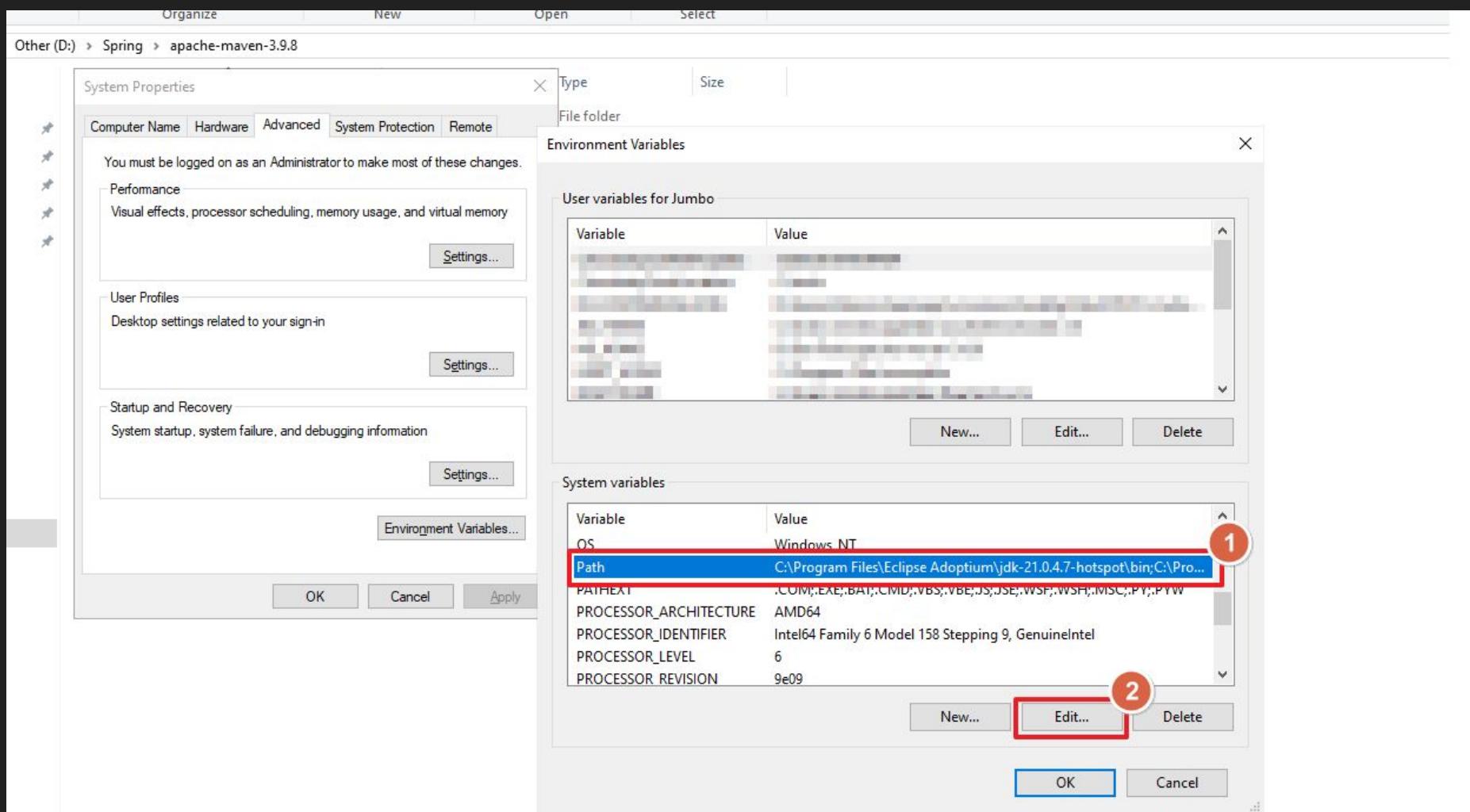
Install Maven 3



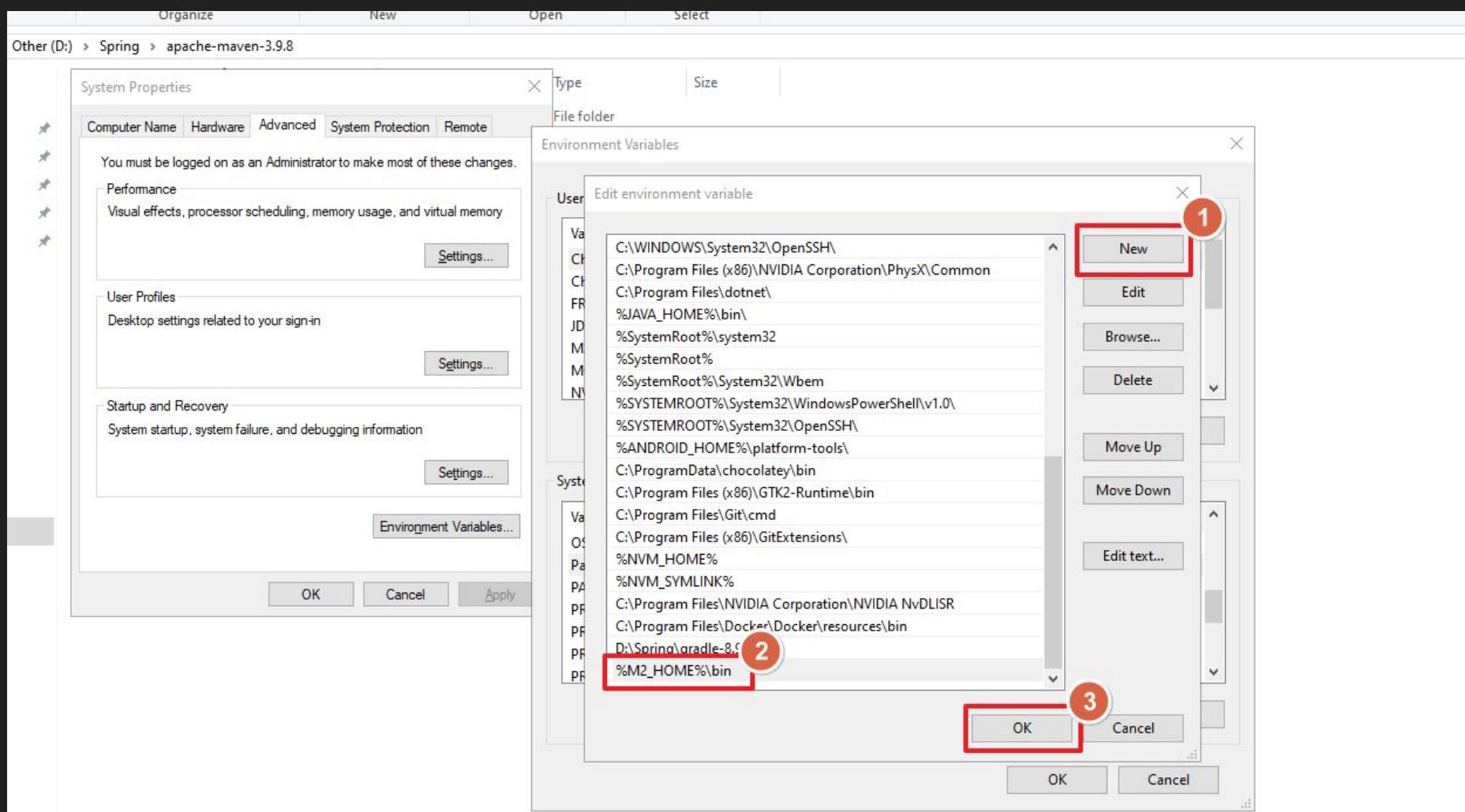
Install Maven 3



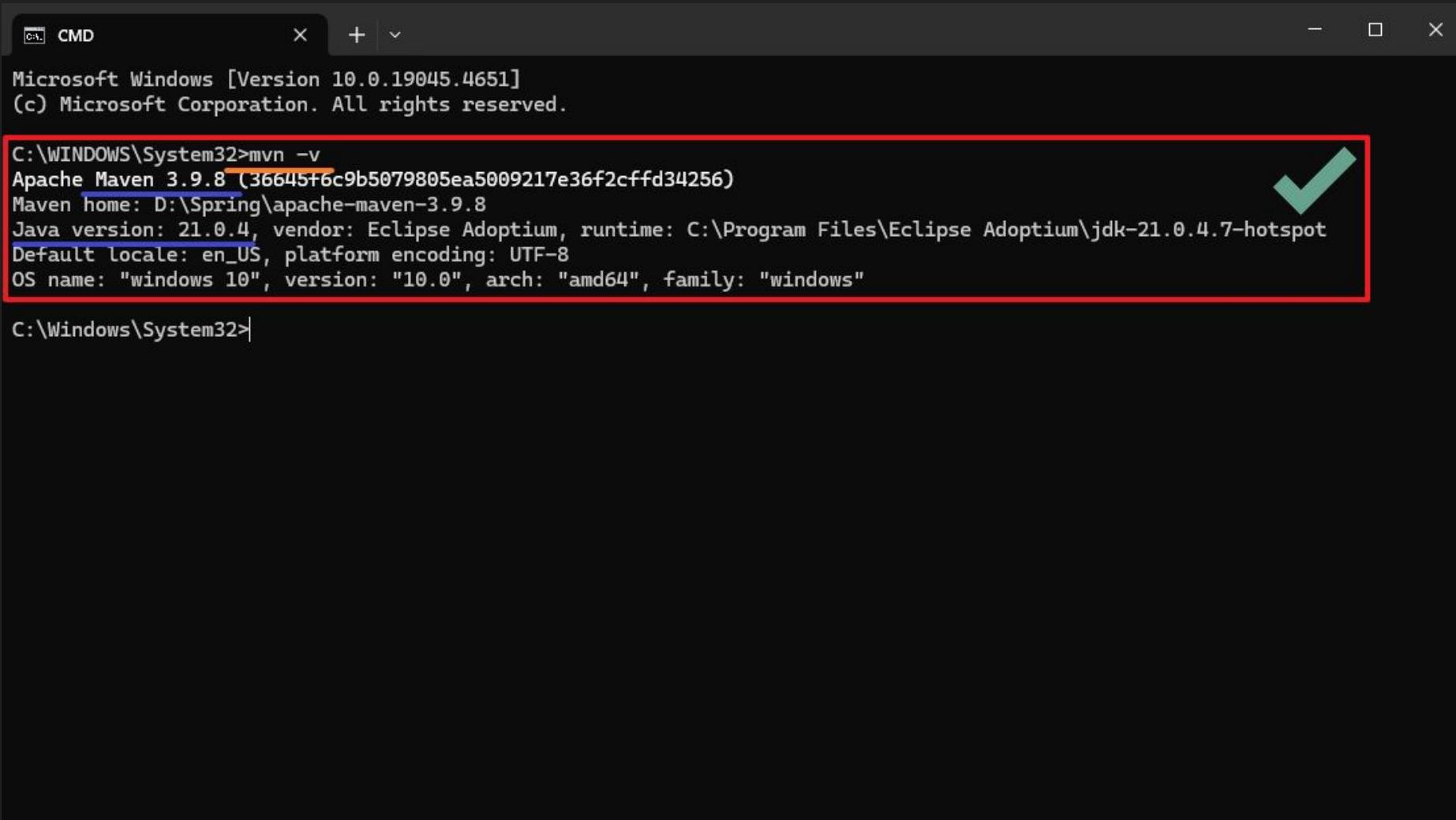
Install Maven 3



Install Maven 3



Install Maven 3



```
C:\CMD X + V Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\System32>mvn -v
Apache Maven 3.9.8 (36645+6c9b5079805ea5009217e36f2cffd34256)
Maven home: D:\Spring\apache-maven-3.9.8
Java version: 21.0.4, vendor: Eclipse Adoptium, runtime: C:\Program Files\Eclipse Adoptium\jdk-21.0.4.7-hotspot
Default locale: en_US, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Windows\System32>
```

Install Visual Studio Code & Extension

Visual Studio Code

The screenshot shows the official website for Visual Studio Code (`code.visualstudio.com`). A red box highlights the URL bar at the top left, and a red circle with the number 1 is centered over it. Another red box highlights the large blue "Download for Windows" button in the center, and a red circle with the number 2 is centered over it. The page features a dark background with white text and icons. It includes a navigation bar with links to Docs, Updates, Blog, API, Extensions, and FAQ. A banner at the top right announces "Version 1.92 is now available! Read about the new features and fix". The main content area features the text "code.visualstudio.com" in large red letters, followed by "Free. Built on open source. Runs everywhere." in a smaller white box. Below this is a large white text area with the heading "Code Editing. Redefined." and a "Download for Windows" button. At the bottom, there's a note about agreeing to the license and privacy statement, and a "my-app" code editor sidebar on the right.

code.visualstudio.com

1

Visual Studio Code Docs Updates Blog API Extensions FAQ

Version 1.92 is now available! Read about the new features and fix

code.visualstudio.com

Free. Built on open source. Runs everywhere.

Code Editing. Redefined.

Download for Windows

2

Web, Insiders edition, or other platform.

By using VS Code, you agree to its [license](#) and [privacy statement](#).

EXPLORER

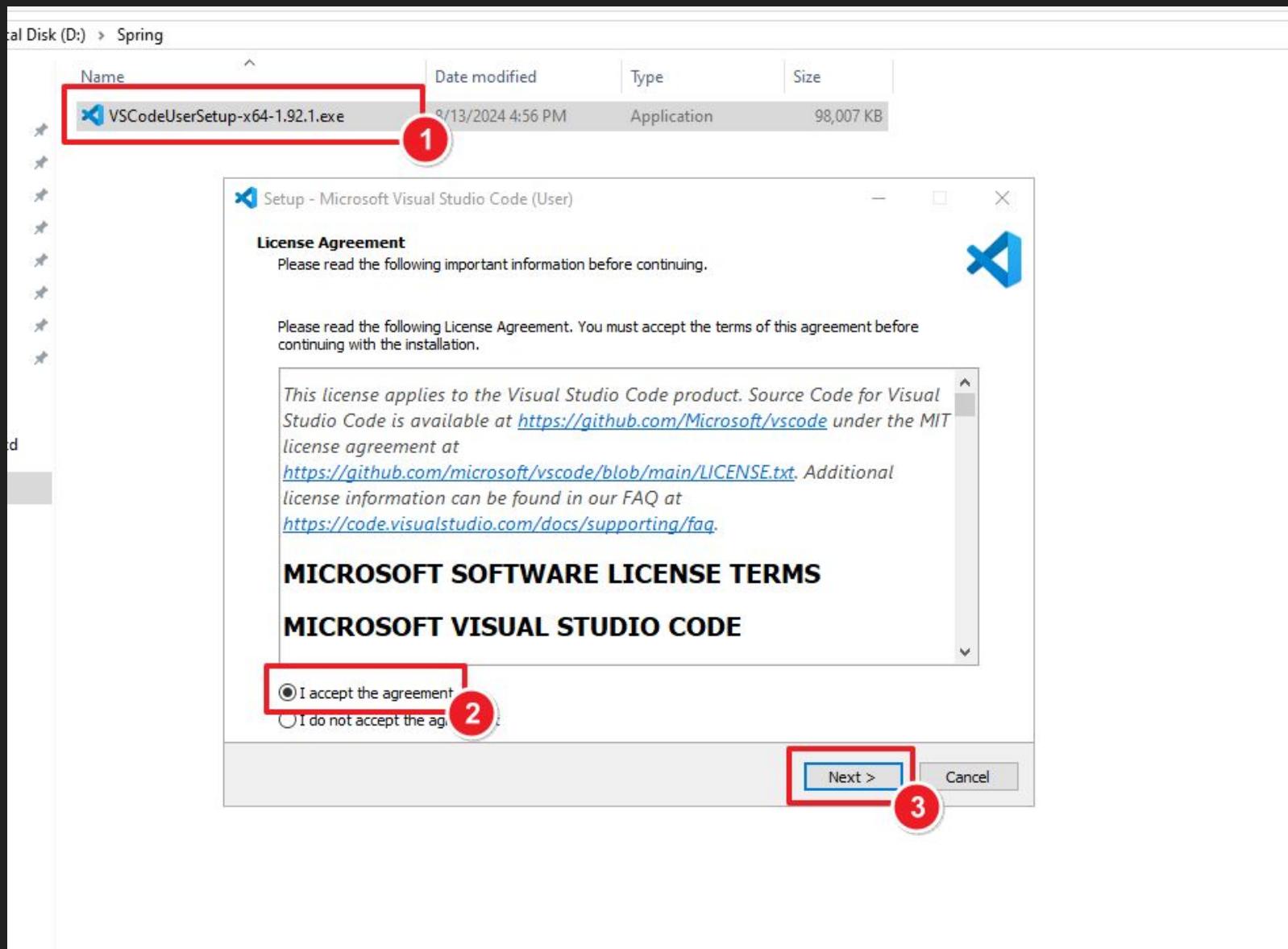
MY-APP

- > components
- > actionBar
- > breadcrumbs
- > button
- # button.css
- TS button.ts
- > countBadge
- > dialog
- > dropdown
- > findInput
- > grid
- > hover
- > inputBox
- ↳ .gitignore
- ↳ .mailmap
- ↳ .mention-bot
- ↳ .yarnrc
- ↳ yarn.lock

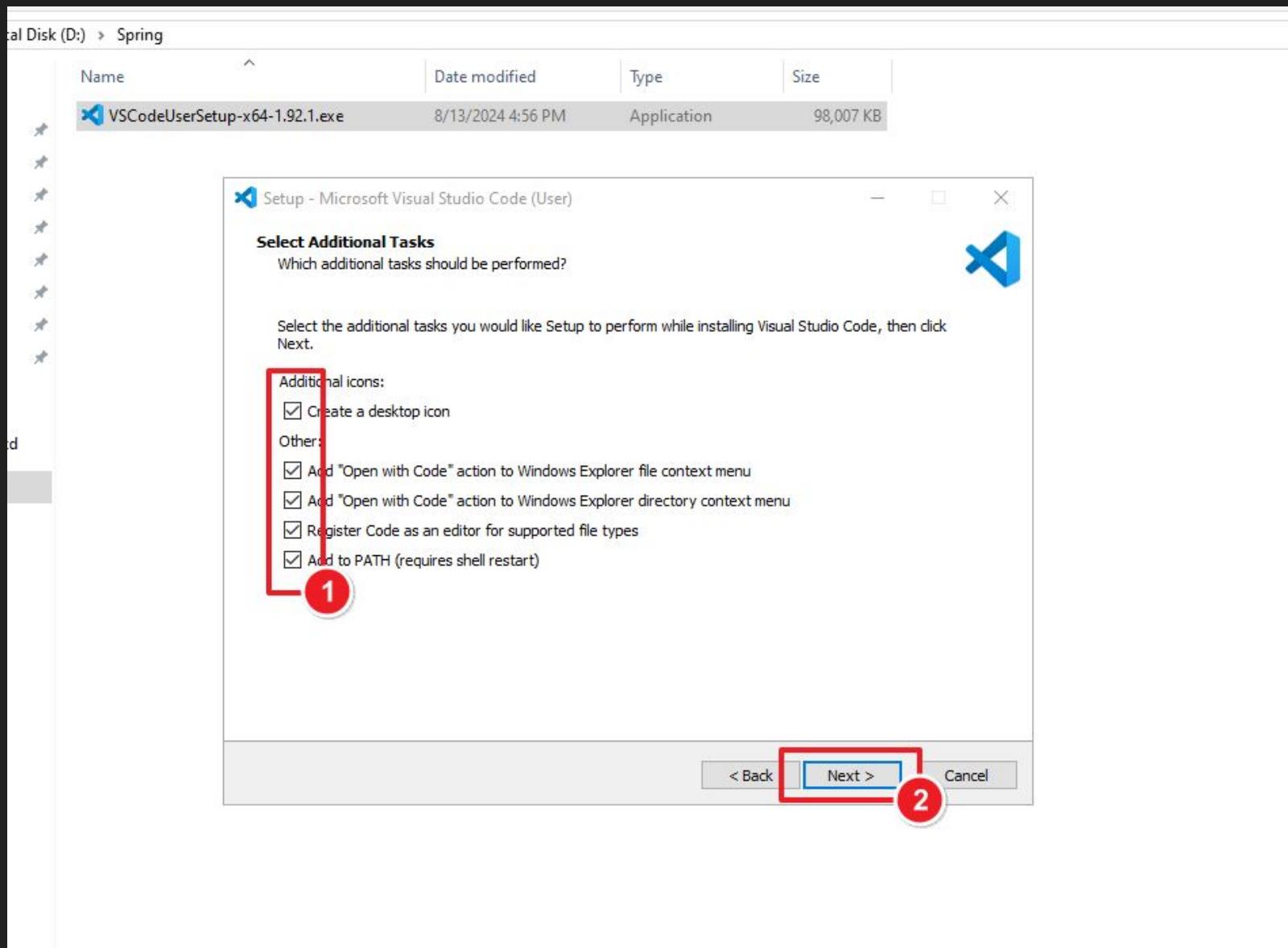
PROBLEMS

main 0 1↑

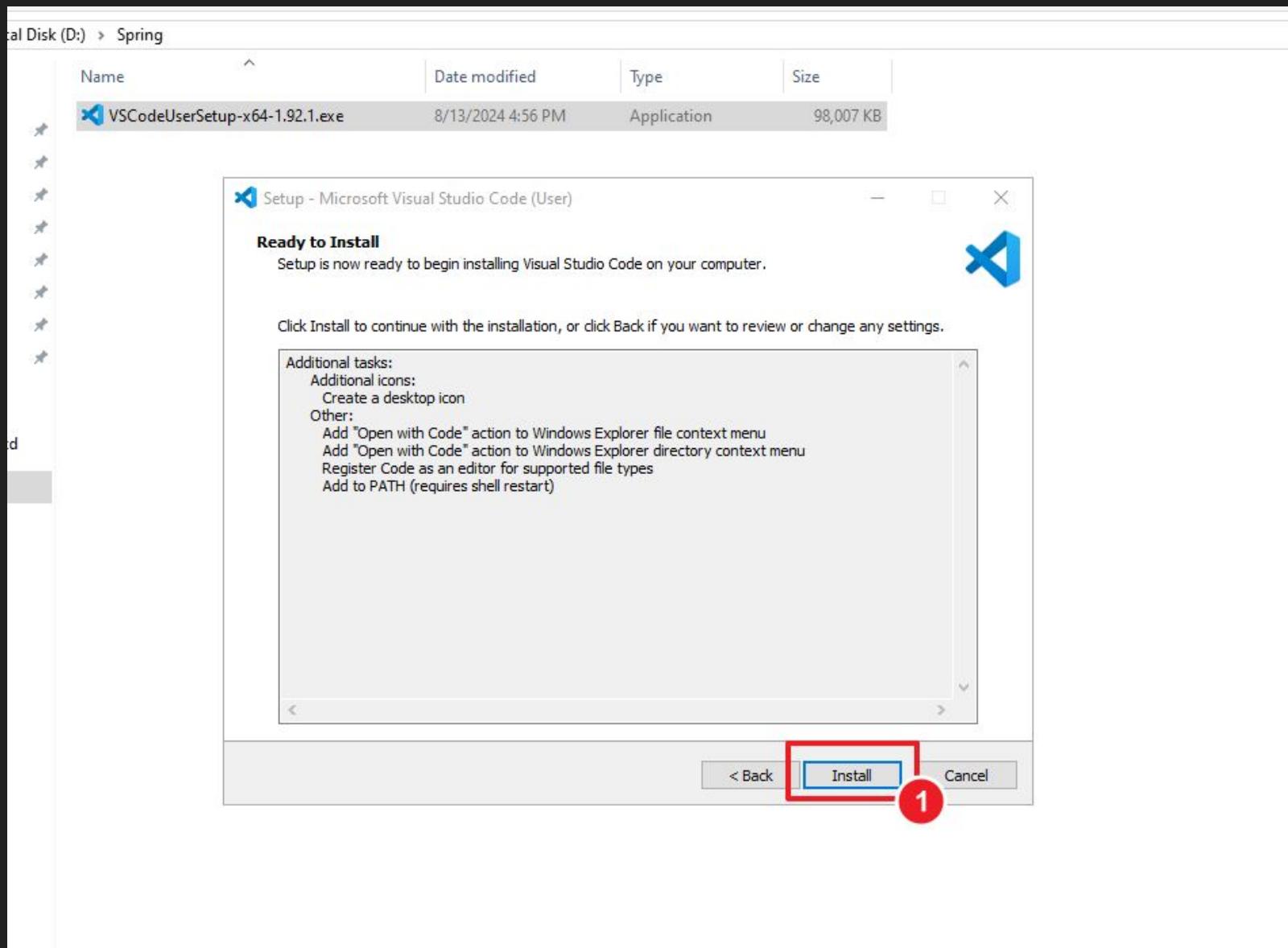
Visual Studio Code



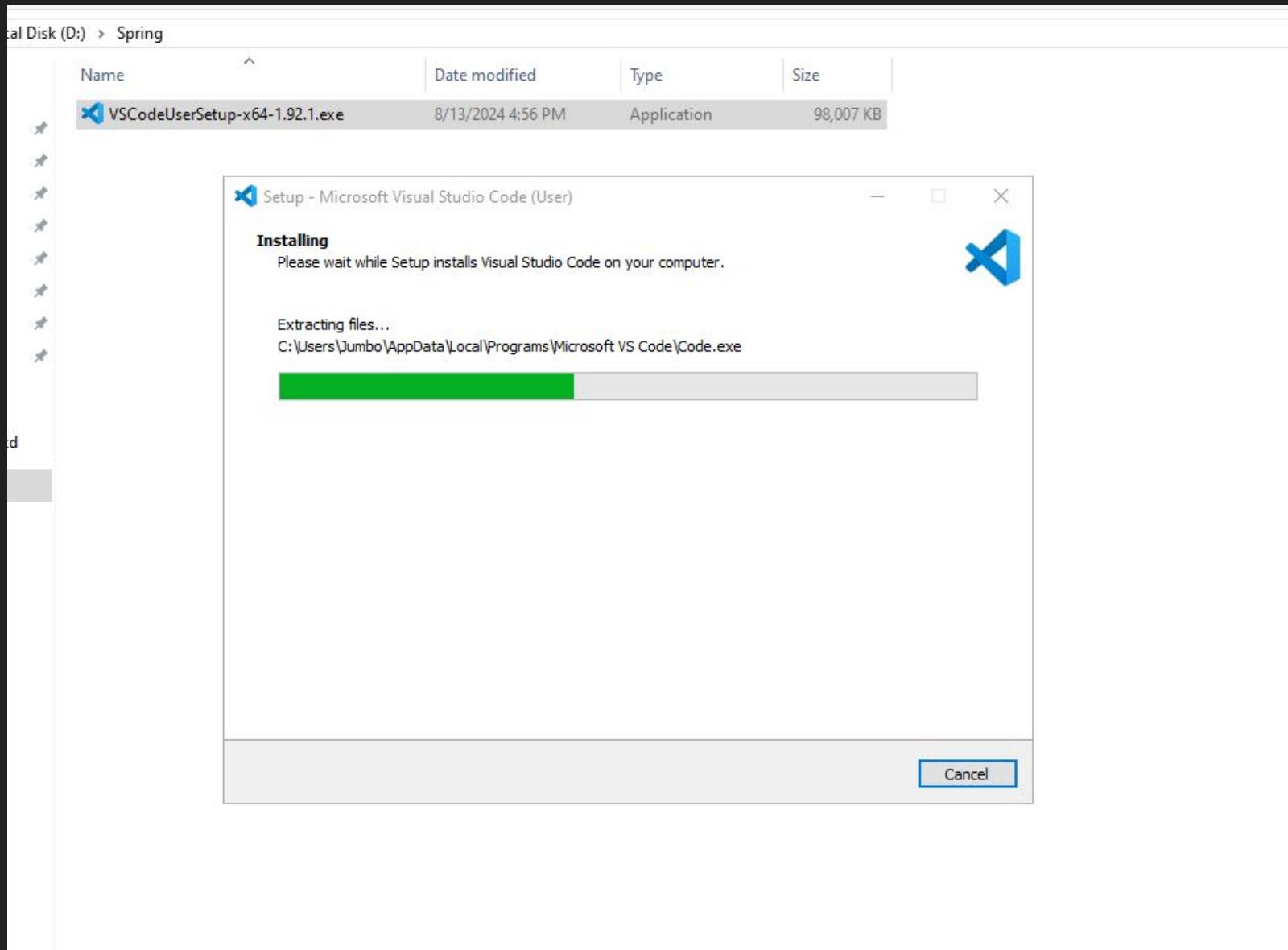
Visual Studio Code



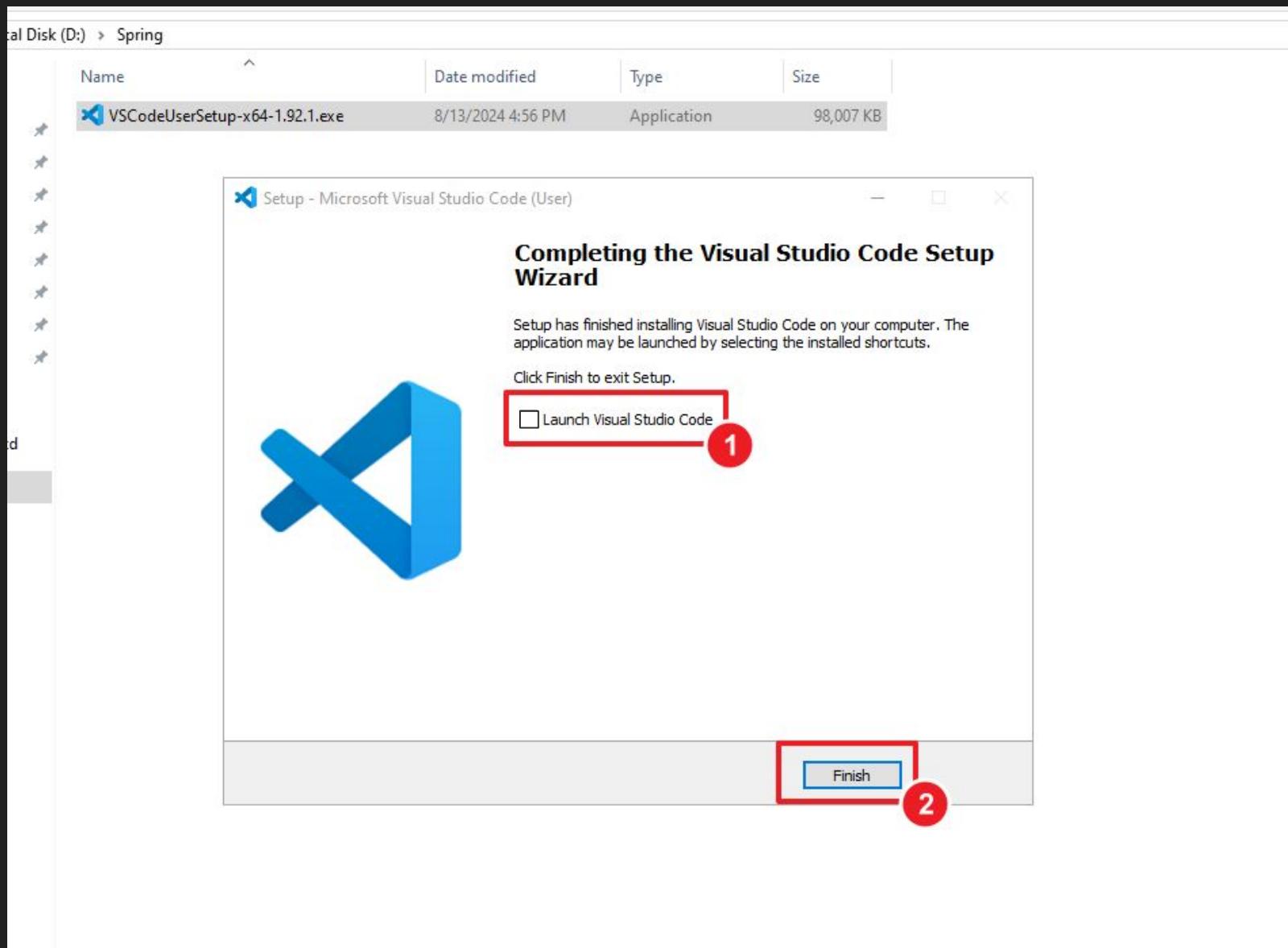
Visual Studio Code



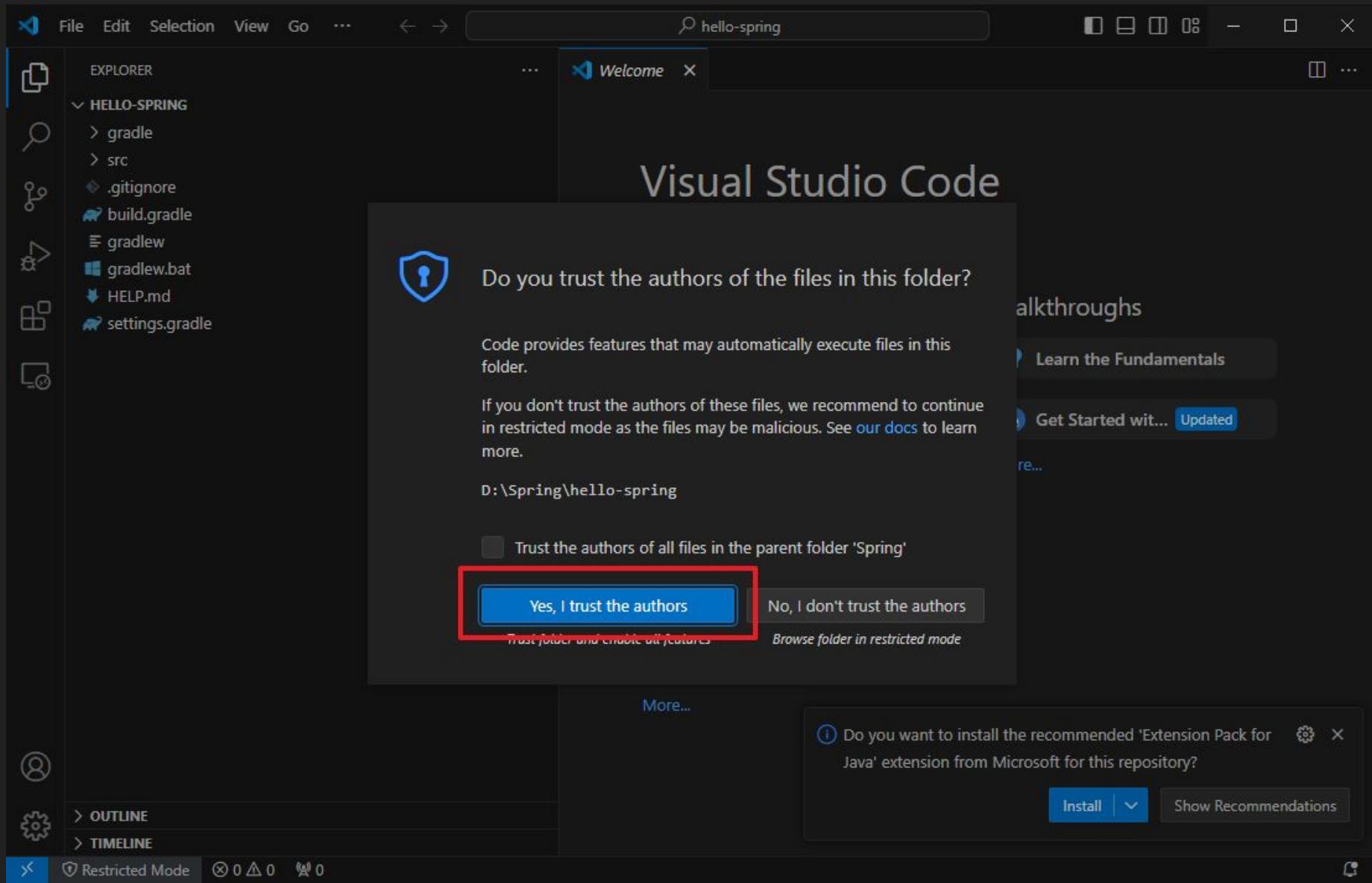
Visual Studio Code



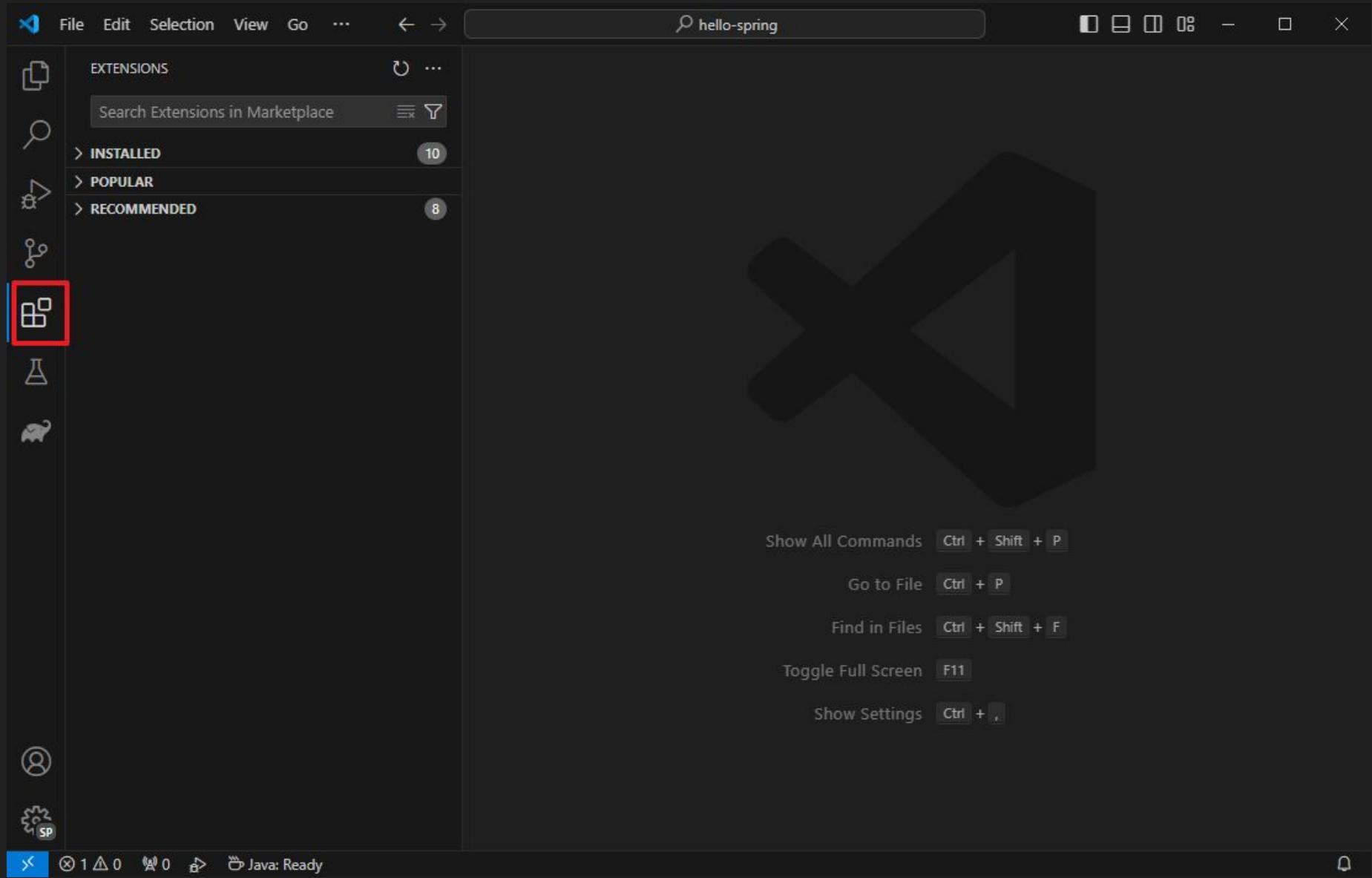
Visual Studio Code



Visual Studio Code



Visual Studio Code (Extension)



Visual Studio Code (Extension)

The screenshot shows the Visual Studio Code interface with the Extensions sidebar open. A search bar at the top right contains the text "hello-spring".

Step 1: The search bar has a red circle with the number "1" above it, and the search term "pack for java" is highlighted with a red box.

Step 2: The results list shows the "Extension Pack for Java" extension by Microsoft at the top, with a red box highlighting its listing and a red circle with the number "2" above it.

Step 3: The extension details page for "Extension Pack for Java" is shown. A red box highlights the "Install" button, and a red circle with the number "3" is placed above it.

Extension Pack for Java Details:

- Extension Pack for Java** v0.27.0 by Microsoft ([microsoft.com](#))
- 28,594,382 installs
- Popular extensions for Java development that provides Java Int...

Categories:

- Programming Languages
- Snippets
- Linters
- Debuggers
- Formatters
- Extension Packs

Resources:

- Marketplace
- Issues
- Repository
- License
- Microsoft

Extension Pack for Java Description:

Extension Pack for Java is a collection of popular extensions that can help write, test and debug Java applications in Visual Studio Code. Check out [Java in VS Code](#) to get started.

Extensions Included:

- IntelliCode
- Language Support for Java(TM) by Red Hat

More Info:

Visual Studio Code (Extension)

The screenshot shows the Visual Studio Code interface with the following elements:

- Top Bar:** File, Edit, Selection, View, Go, ...
- Search Bar:** hello-spring
- Sidebar Icons:** File, Edit, Selection, View, Go, ...
- Left Panel:** EXTENSIONS: MARKETPLACE
- Search Results:** A list of extensions related to Spring Boot:
 - Spring Boot Extension Pack** by VMware (version 2.5M, 5 stars) - highlighted with a red box and circled '1'.
 - Icon: Green gear with a leaf.
 - Description: A collection of extensions for developing Spring Boot applications.
 - Developer: VMware
 - Install button
 - Spring Initializr Java Support** by Microsoft (version 3.8M, 4 stars)
 - Icon: Green power button.
 - Description: A lightweight extension based on Spring Initializr.
 - Developer: Microsoft
 - Install button
 - Spring Boot Developer Extension Pack** by Developer Soapbox (version 56K, 5 stars)
 - Icon: Green gear.
 - Description: Useful extensions needed for effectively developing Spring Boot applications.
 - Developer: Developer Soapbox
 - Install button
 - Spring Boot Extra Pack** by Abdul Gaffur A Dama (version 1K)
 - Icon: Grey gear.
 - Description: Collection for developing Spring Boot applications.
 - Developer: Abdul Gaffur A Dama
 - Install button
 - Extension Pack for Java Developers** by Loiane Groner (version 101K, 5 stars)
 - Icon: Blue book.
 - Description: Some of the most popular and useful Java extensions.
 - Developer: Loiane Groner
 - Install button
 - Zee Spring Boot Extension Pack** by Buddhika (version 1K)
 - Icon: Grey gear.
 - Description: An extension pack for Spring Boot development.
 - Developer: Buddhika
 - Install button
 - Advanced Java + Quarkus + Spring Boot** by Arthur Gultarte Brandi (version 899)
 - Icon: Flame and gear.
 - Description: This collection includes all extensions for Java, Quarkus, and Spring Boot.
 - Developer: Arthur Gultarte Brandi
 - Install button
 - spring-boot-pack** by Malolan (version 8K)
 - Icon: Grey gear.
 - Description: Extension pack for springboot projects.
 - Developer: Malolan
 - Install button
- Bottom Status Bar:** (1) 0 0 0 0 Java: Ready

Extension Details View: The 'Spring Boot Extension Pack' page is open, showing:

- Title:** Spring Boot Extension Pack v0.2.1
- Developer:** VMware | vmware.com
- Downloads:** 2,507,607
- Rating:** ★★★★★ (18)
- Description:** A collection of extensions for developing Spring Boot applications.
- Install Button:** Circled '2'.
- Details Tab:** Shows the extension pack's details.
- Features Tab:** Shows the included extensions:
 - Spring Initializr Java Support** by Microsoft
- Categories:** Programming Languages, Linters, Extension Packs
- Resources:** Marketplace, Issues, Repository, VMware
- More Info:** Published 2017-11-29, 02:43:15, Last 2023-02-01.

Text Overlay: VS Code Spring Boot Application Development Extension Pack

Visual Studio Code (Extension)

The screenshot shows the Visual Studio Code interface with the following elements:

- File Bar:** File, Edit, Selection, View, Go, ...
- Search Bar:** hello-spring
- Toolbar:** Minimize, Maximize, Close, etc.
- Left Sidebar:** EXTENSIONS, REPLACE, thunder, Thunder Client (highlighted with red box and circled 1), Thunder (circled 2), Thunder Focus, Velvet Thunder, Thunder Theme, thunder_dust.
- Main Area:**
 - Extension Details:** Thunder Client (v2.24.13) by Thunder Client (thunderclient.com). It has 4,190,476 downloads and 5 stars. The Install button is highlighted with a red box and circled 3.
 - Description:** Thunder Client is a lightweight Rest API Client Extension for VS Code, hand-crafted by Ranga Vadhineni with a focus on simplicity, clean design and local storage.
 - Resources:** Marketplace, Issues, Repository, License, Thunder Client.
 - Milestones:** The extension was launched on March 31st, 2021, with 500K downloads on Dec 20th, 2021.
- Bottom Status Bar:** Java: Ready, various icons for file operations, and a notification bell.

Visual Studio Code (Extension)

The screenshot shows the Visual Studio Code interface with the following highlights:

- Search Bar:** The search bar at the top contains the text "hello-spring".
- Marketplace Search:** A red box highlights the search bar in the Extensions view, with a red circle containing the number "1" above it.
- Extension Details:** The "Material Icon Theme" extension by Philipp Kief is selected in the search results. A red box highlights the extension card, with a red circle containing the number "2" above it. The "Install" button is also highlighted with a red box and a red circle containing the number "3".
- Bottom Status Bar:** The status bar at the bottom shows various icons and the message "Java: Ready".

Create Spring Boot Project with Spring Initializr

Spring Initializr

The screenshot shows the Spring Initializr interface. At the top, there is a navigation bar with icons for home, search, and user profile, followed by the URL 'start.spring.io'. Below the header, the page title is 'spring initializr' with a green leaf icon, and the main heading is 'start.spring.io' in red.

Project

- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.4.0 (SNAPSHOT)
- 3.4.0 (M1)
- 3.3.3 (SNAPSHOT)
- 3.3.2
- 3.2.9 (SNAPSHOT)
- 3.2.8

Spring Initializr



Project
 Gradle - Groovy
 Maven 1

Language
 Java 2 Kotlin
 Groovy

Spring Boot
 3.4.0 (SNAPSHOT) 3.4.0 (M1) 3.3.3 (SNAPSHOT)
 3.2.9 (SNAPSHOT) 3.2.8 3

Project Metadata
4

5 War

6 17

Dependencies

No dependency selected

7 ADD DEPENDENCIES... CTRL + B

Spring Initializr

spring initializr

Web, Security, JPA, Actuator, Devtools...

Press Ctrl for multiple adds

DEVELOPER TOOLS

GraalVM Native Support
Support for compiling Spring applications to native executables using the GraalVM native-image compiler.

GraphQL DGS Code Generation
Generate data types and type-safe APIs for querying GraphQL APIs by parsing schema files.

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok
Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor
Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex. application.properties/.yml files).

Docker Compose Support
Provides docker compose support for enhanced development experience.

Spring Modular
Support for building modular monolithic applications.

WEB

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

Spring Reactive Web
Build reactive web applications with Spring WebFlux and Netty.

ADD DEPENDENCIES... CTRL + B

Project
 Gradle - Groovy Gradle - Kotlin
 Maven

Spring Boot
 3.4.0 (SNAPSHOT) 3.4.0 (M1)
 3.2.9 (SNAPSHOT) 3.2.8

Project Metadata
Group: com.workshop
Artifact: student
Name: student
Description: Workshop
Package name: com.workshop.student
Packaging: Jar War
Java: 22 21 17

Spring Initializr

spring initializr

Web, Security, JPA, Actuator, Devtools...

Press Ctrl for multiple adds

TEMPLATE ENGINES

Thymeleaf
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Apache Freenamer
Java library to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data.

Mustache
Logic-less templates for both web and standalone environments. There are no if statements, else clauses, or for loops. Instead there are only tags.

Groovy Templates
Groovy templating engine.

SECURITY

Spring Security
Highly customizable authentication and access-control framework for Spring applications.

OAuth2 Client
Spring Boot integration for Spring Security's OAuth2/OpenID Connect client features.

OAuth2 Authorization Server
Spring Boot integration for Spring Authorization Server.

OAuth2 Resource Server
Spring Boot integration for Spring Security's OAuth2 resource server features.

ADD DEPENDENCIES... CTRL + B

Project
 Gradle - Groovy Gradle - Kotlin
 Maven

Spring Boot
 3.4.0 (SNAPSHOT) 3.4.0 (M1)
 3.2.9 (SNAPSHOT) 3.2.8

Project Metadata
Group: com.workshop
Artifact: student
Name: student
Description: Workshop
Package name: com.workshop.student
Packaging: Jar War
Java: 22 21 17

Spring Initializr

spring initializr

Web, Security, JPA, Actuator, Devtools...

Press Ctrl for multiple adds

SQL

JDBC API
Database Connectivity API that defines how a client may connect and query a database.

Spring Data JPA
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Data JDBC
Persist data in SQL stores with plain JDBC using Spring Data.

Spring Data R2DBC
Provides Reactive Relational Database Connectivity to persist data in SQL stores using Spring Data in reactive applications.

MyBatis Framework
Persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations.

Liquibase Migration
Liquibase database migration and source control library.

Flyway Migration
Version control for your database so you can migrate from any version (incl. an empty database) to the latest version of the schema.

JOOQ Access Layer
Generate Java code from your database and build type safe SQL queries through a fluent API.

IBM DB2 Driver
A JDBC driver that provides access to IBM DB2.

ADD DEPENDENCIES... CTRL + B

Project

Gradle - Groovy Gradle - Kotlin
 Maven

Spring Boot

3.4.0 (SNAPSHOT) 3.4.0 (M1)
 3.2.9 (SNAPSHOT) 3.2.8

Project Metadata

Group: com.workshop

Artifact: student

Name: student

Description: Workshop

Package name: com.workshop.student

Packaging: Jar War

Java: 22 21 17

Spring Initializr

spring initializr

Web, Security, JPA, Actuator, Devtools...

Press Ctrl for multiple adds

SQL

JOOQ Access Layer
Generate Java code from your database and build type safe SQL queries through a fluent API.

IBM DB2 Driver
A JDBC driver that provides access to IBM DB2.

Apache Derby Database
An open source relational database implemented entirely in Java.

H2 Database
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint.
Supports embedded and server modes as well as a browser based console application.

HyperSQL Database
Lightweight 100% Java SQL Database Engine.

MariaDB Driver
MariaDB JDBC and R2DBC driver.

MS SQL Server Driver
A JDBC and R2DBC driver that provides access to Microsoft SQL Server and Azure SQL Database from any Java application.

MySQL Driver
MySQL JDBC driver.

Oracle Driver
A JDBC driver that provides access to Oracle.

PostgreSQL Driver

ADD DEPENDENCIES... CTRL + B

Project
 Gradle - Groovy Gradle - Kotlin
 Maven

Spring Boot
 3.4.0 (SNAPSHOT) 3.4.0 (M1)
 3.2.9 (SNAPSHOT) 3.2.8

Project Metadata
Group: com.workshop
Artifact: student
Name: student
Description: Workshop
Package name: com.workshop.student
Packaging: Jar War
Java: 22 21 17

Generate Project

The screenshot shows the Spring Initializer interface for generating a new Spring Boot project. The interface is divided into two main sections: Project Configuration on the left and Dependencies on the right.

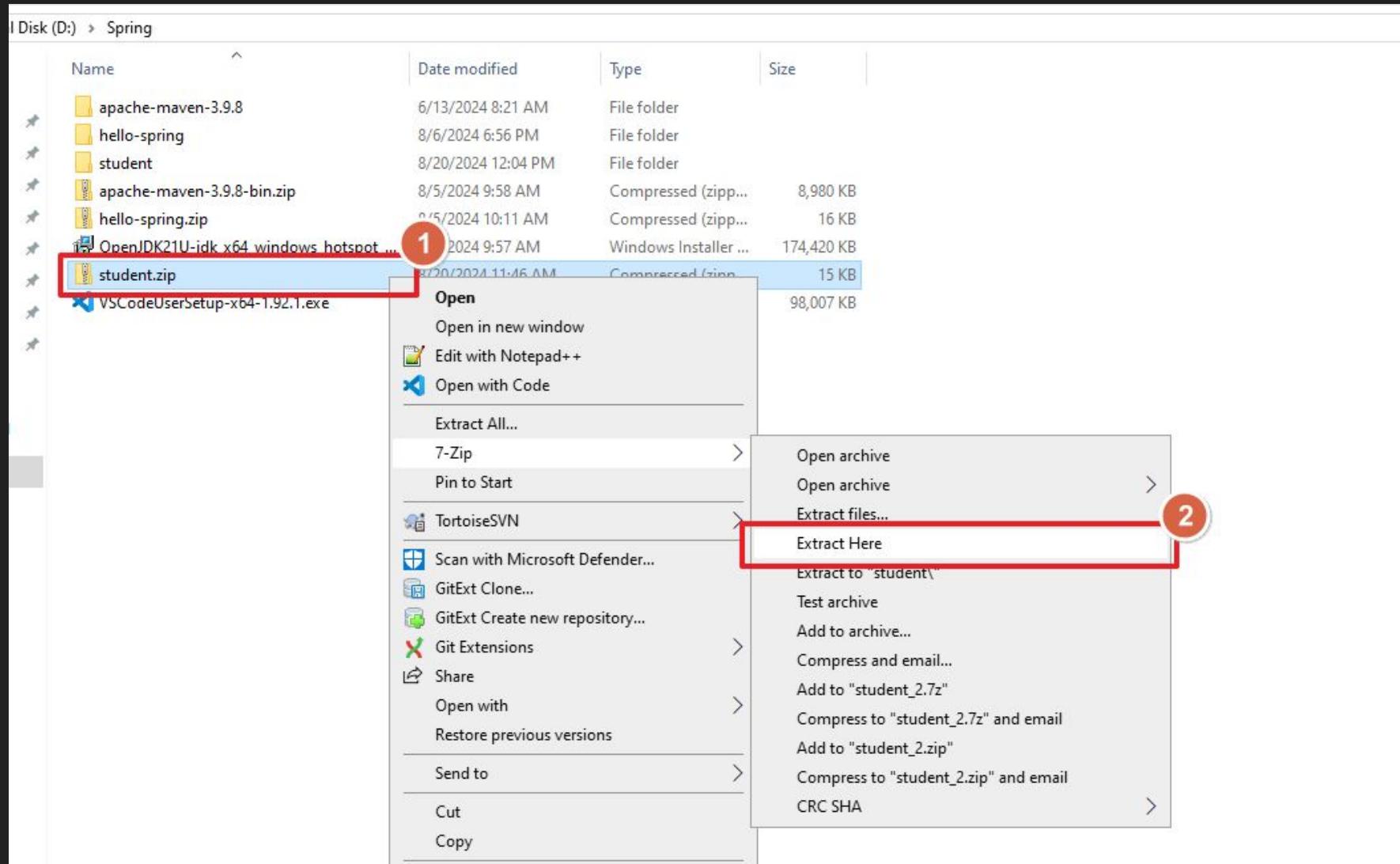
Project Configuration (Left Side):

- Project:** Maven (1)
- Language:** Java (2)
- Spring Boot:** 3.3.2 (3)
- Project Metadata:**
 - Group: com.workshop (4)
 - Artifact: student
 - Name: student
 - Description: Workshop
 - Package name: com.workshop.student
 - Packaging: Jar (5) War
 - Java: 22 (6) 21 (7) 17

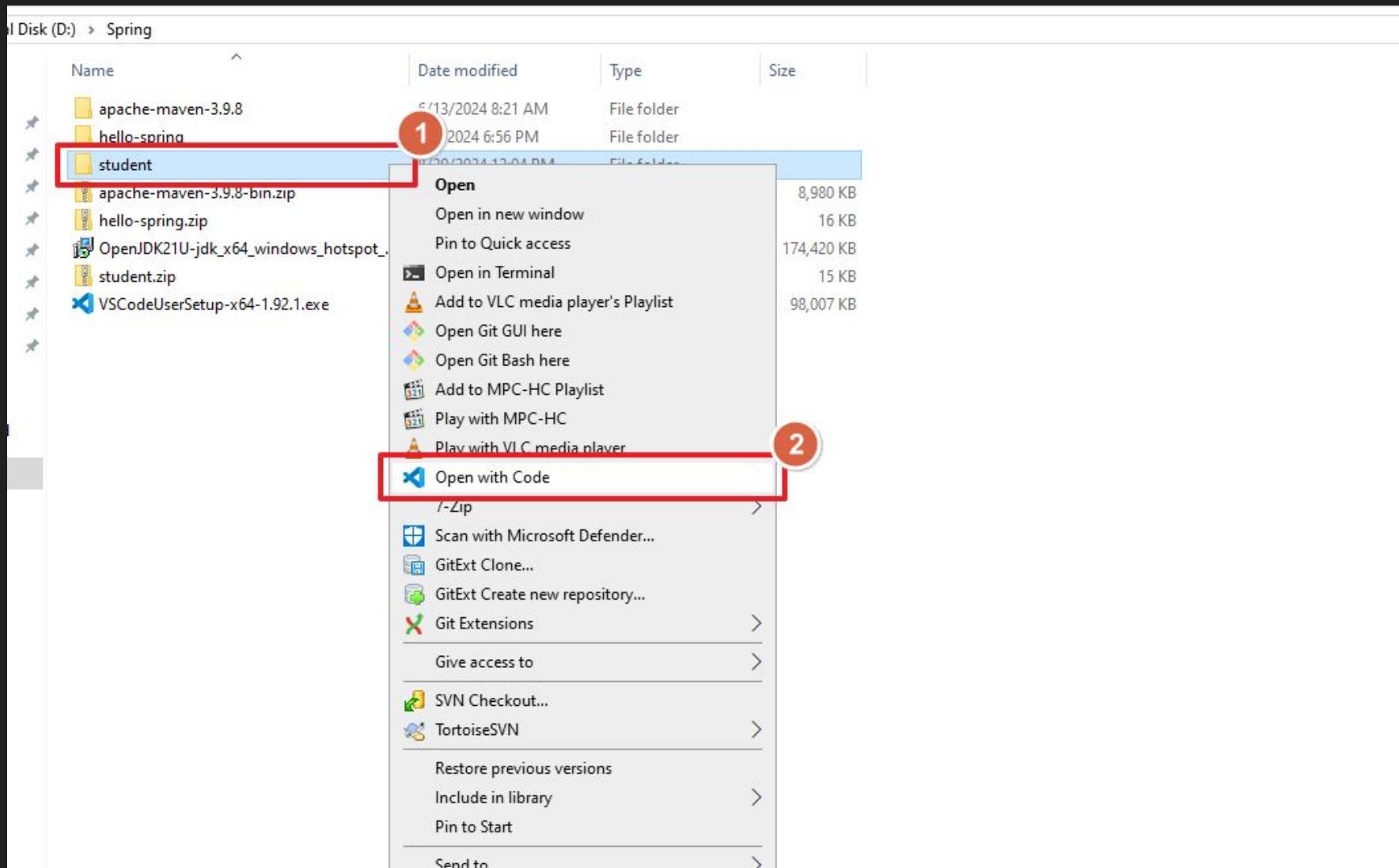
Dependencies (Right Side):

- Dependencies:** ADD DEPENDENCIES... CTRL + B (8)
- Spring Boot Dev Tools** [DEVELOPER TOOLS] (7)
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- Lombok** [DEVELOPER TOOLS]
Java annotation library which helps to reduce boilerplate code.
- Spring Web** [WEB]
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Thymeleaf** [TEMPLATE ENGINES]
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.
- Spring Data JPA** [SQL]
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- H2 Database** [SQL]
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

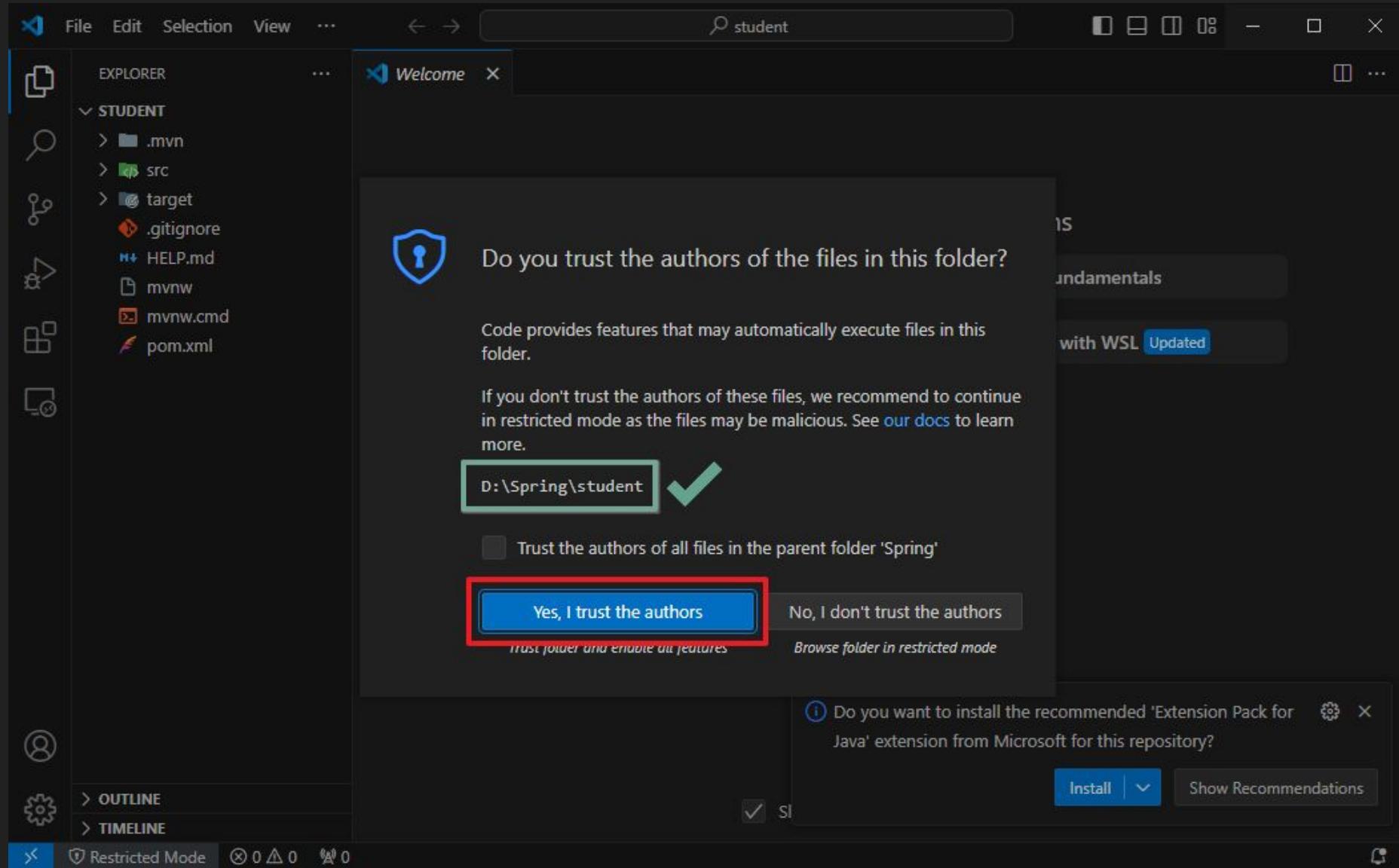
Extract Project



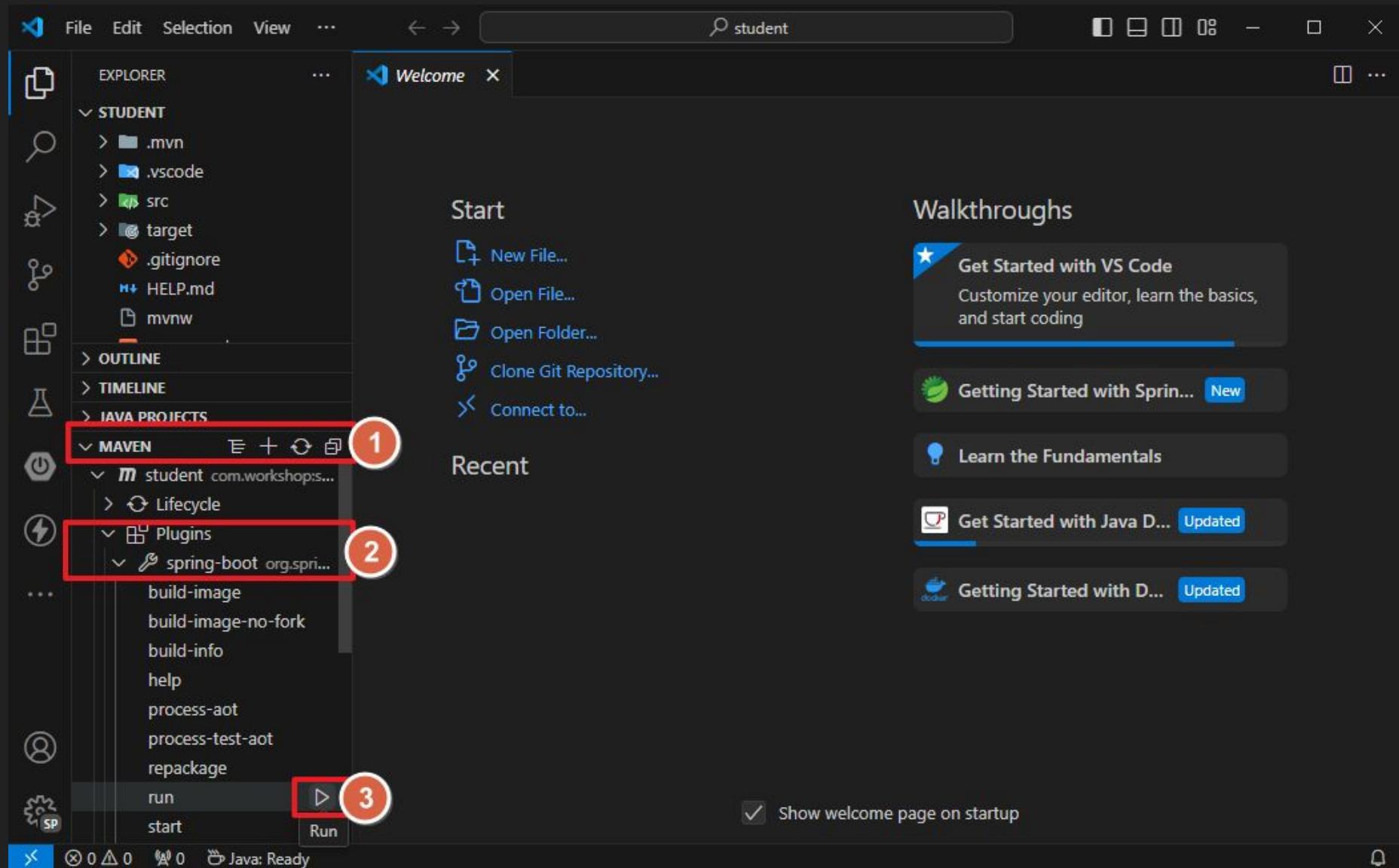
Open Project with Visual Studio Code



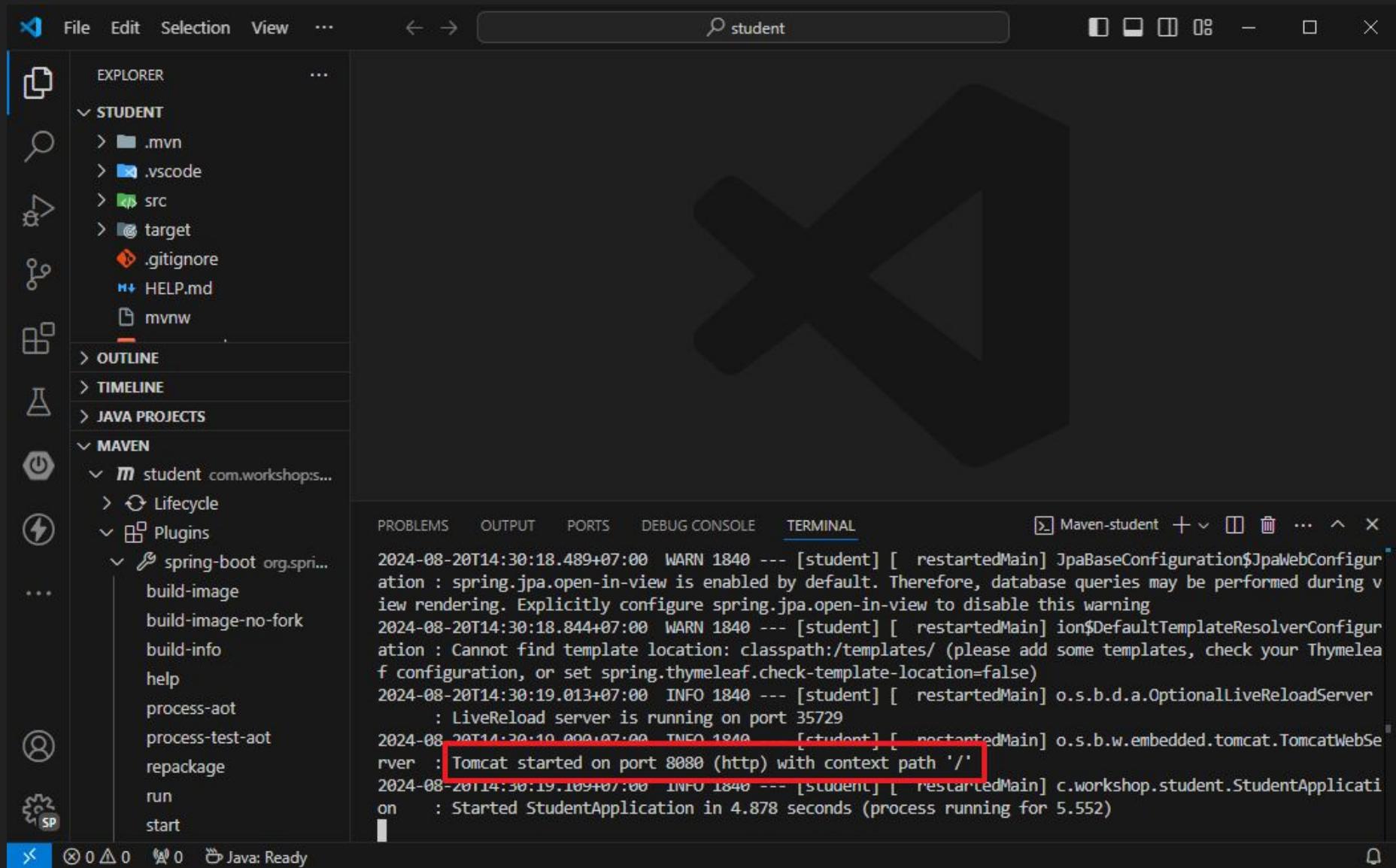
Trust the author



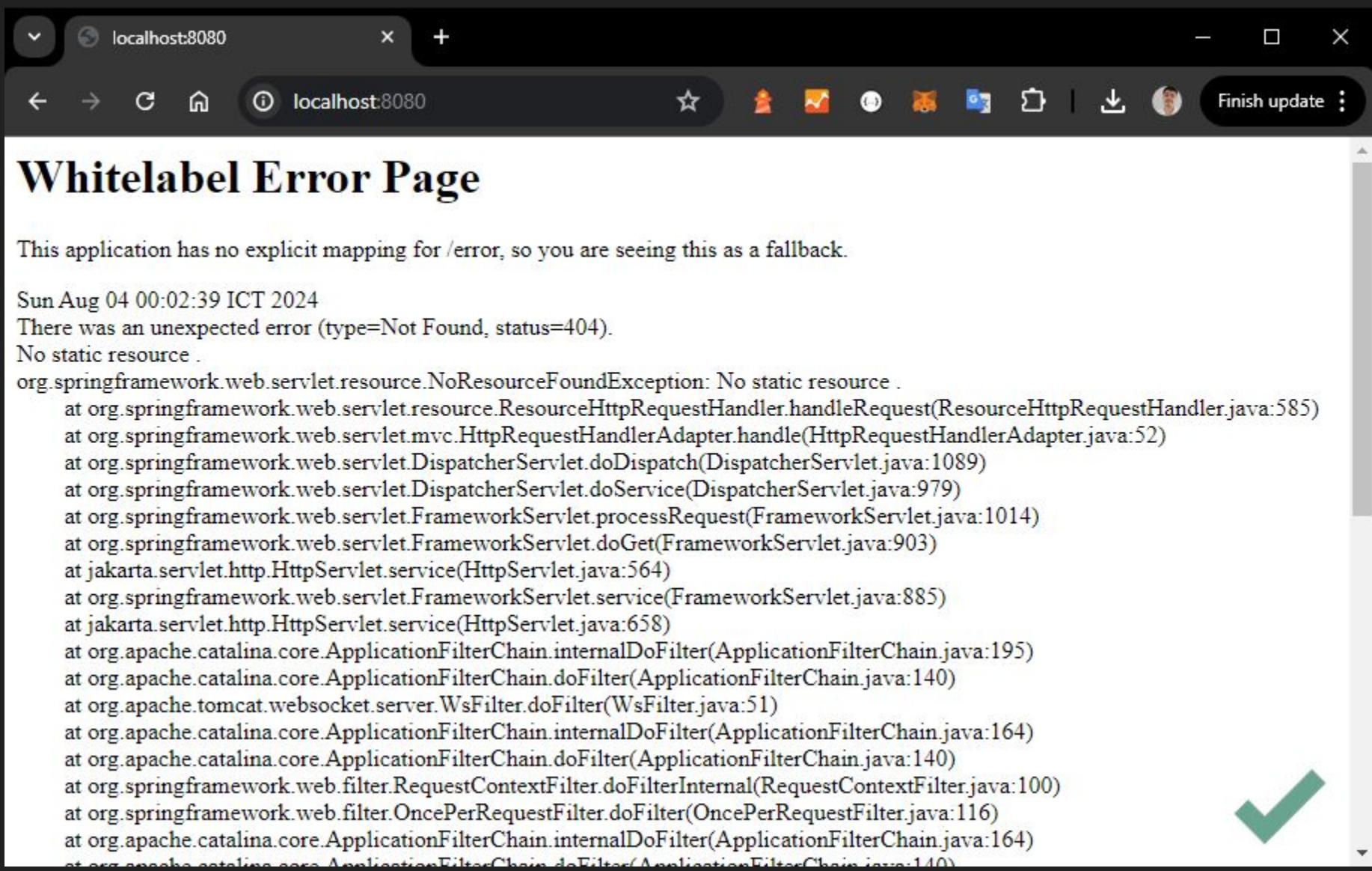
Start Project



Start Project



Start Project



localhost:8080

localhost:8080

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Aug 04 00:02:39 ICT 2024

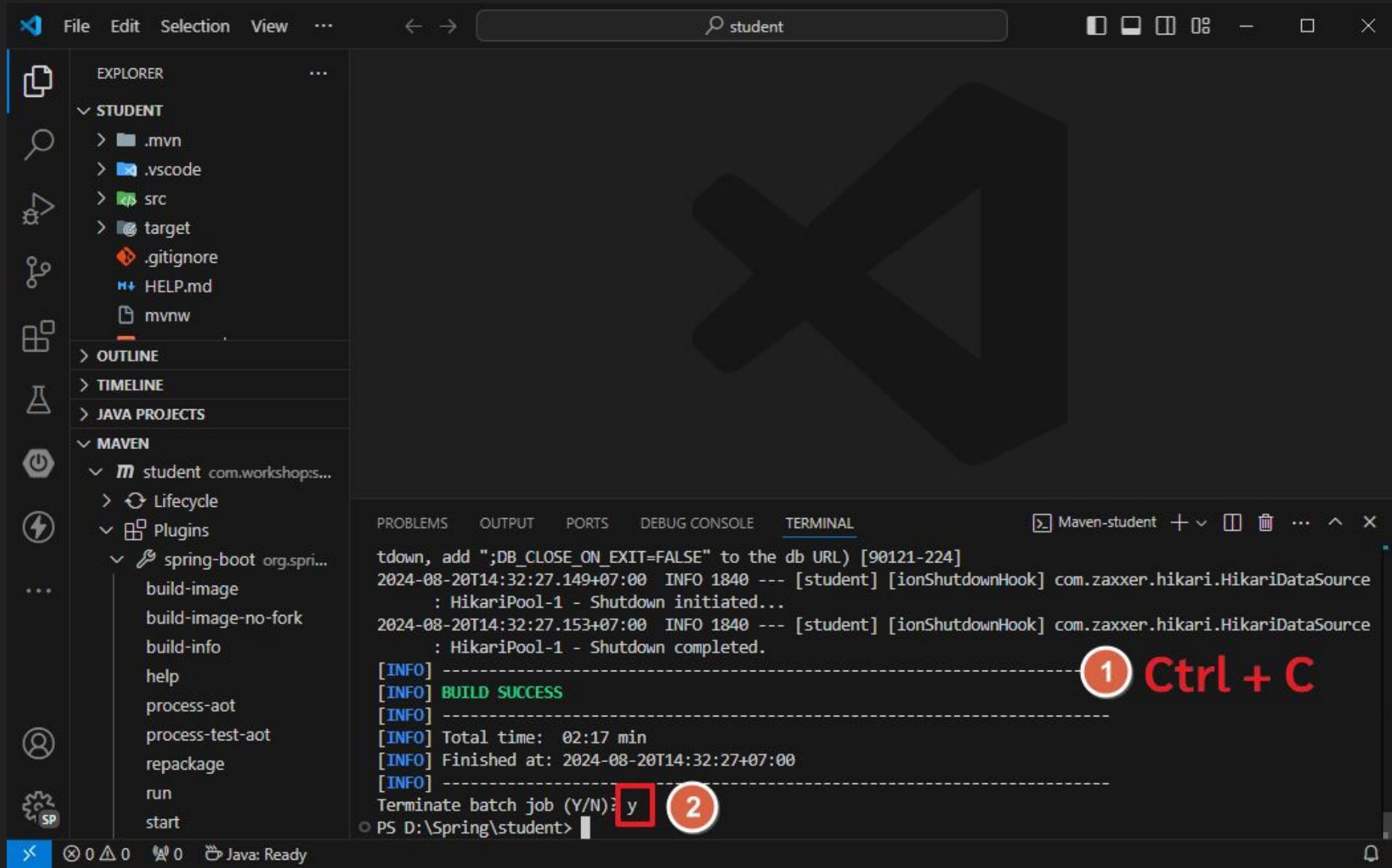
There was an unexpected error (type=Not Found, status=404).

No static resource .

```
org.springframework.web.servlet.resource.NoResourceNotFoundException: No static resource .
at org.springframework.web.servlet.resource.ResourceHttpRequestHandler.handleRequest(ResourceHttpRequestHandler.java:585)
at org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter.handle(HttpRequestHandlerAdapter.java:52)
at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1089)
at org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:979)
at org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1014)
at org.springframework.web.servlet.FrameworkServlet doGet(FrameworkServlet.java:903)
at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:564)
at org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:885)
at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:658)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:195)
at org.apache.catalina.core.ApplicationFilterChain doFilter(ApplicationFilterChain.java:140)
at org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:164)
at org.apache.catalina.core.ApplicationFilterChain doFilter(ApplicationFilterChain.java:140)
at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestContextFilter.java:100)
at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:116)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:164)
at org.apache.catalina.core.ApplicationFilterChain doFilter(ApplicationFilterChain.java:140)
```



Stop Project



Stop Project

A screenshot of a web browser window titled "localhost". The address bar shows "localhost:8080". The page content displays an error message: "This site can't be reached" with a network icon, followed by "localhost refused to connect." Below this, under "Try:", there are two bullet points: "• Checking the connection" and "• Checking the proxy and the firewall". At the bottom left is a "Reload" button, and at the bottom right is a "Details" button.

This site can't be reached

localhost refused to connect.

Try:

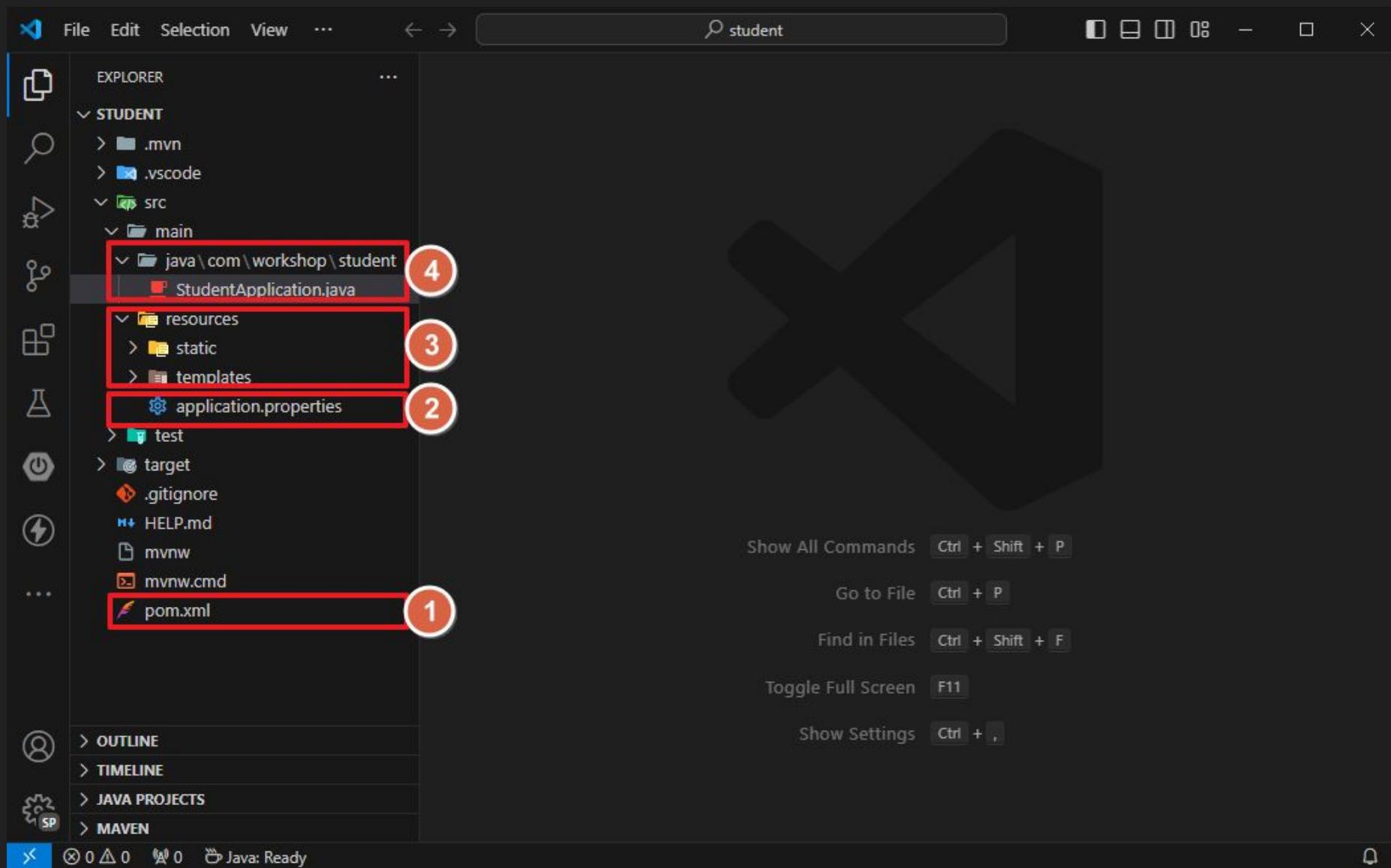
- Checking the connection
- Checking the proxy and the firewall

ERR_CONNECTION_REFUSED

Reload Details

Project Structure

Project Structure



Project Structure (pom.xml)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "STUDENT". Key folders include ".mvn", ".vscode", "src" (containing "main" with "StudentApplication.java" and "resources" with "static" and "templates"), "test", "target", and build files like ".gitignore", "HELP.md", "mvnw", and "mvnw.cmd".
- Search Bar:** Contains the text "student".
- Code Editor (Right):** Displays the content of the "pom.xml" file. A red box highlights the following section:

```
<groupId>com.workshop</groupId>
<artifactId>student</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>student</name>
<description>Workshop</description>
```
- Bottom Status Bar:** Shows "Ln 28, Col 11" and other status indicators.

Project Structure (pom.xml)

The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right. The code editor displays the `pom.xml` file for a Spring Boot application named "student".

Annotation 1 highlights the `<properties>` section, which defines the Java version required:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ...
  <properties>
    <java.version>21</java.version>
  </properties>
```

Annotation 2 highlights the `<dependencies>` section, which lists the dependencies for the application:

```
<dependencies> Add Spring Boot 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-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
```

The status bar at the bottom indicates "Java: Ready".

Project Structure (application.properties)

The screenshot shows the VS Code interface with a dark theme. The left sidebar contains icons for Explorer, Search, Find, Open, Close, Run, Stop, and Help. The Explorer view shows a project structure for a Spring Boot application named 'STUDENT'. The 'src' folder contains 'main' and 'test' directories. 'main' contains 'java', 'resources', and 'static'. 'resources' contains 'application.properties'. The 'application.properties' file is open in the editor, showing the line 'spring.application.name=student'. A red box highlights the 'application.properties' file and its contents. The status bar at the bottom shows 'Java: Ready'.

```
spring.application.name=student
```

Project Structure

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT folder
 - .mvn
 - .vscode
 - src
 - main
 - java\com\workshop\student (StudentApplication.java)
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN- Code Editor:** StudentApplication.java
- Code Content:**

```
1 package com.workshop.student;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class StudentApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(StudentApplication.class, args);
11     }
12 }
13
14 }
```

A red box highlights the main method code.

- Bottom Status Bar:** Ln 14, Col 1 Tab Size: 4 UTF-8 LF {} Java

MVC Design Pattern

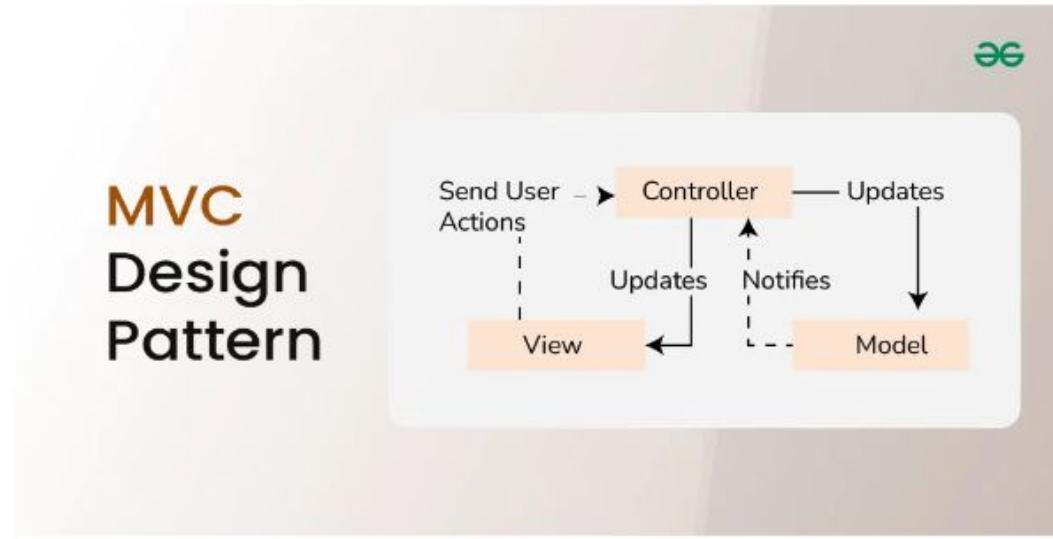
MVC Design Pattern

MVC Design Pattern

Last Updated : 19 Feb, 2024



The MVC design pattern is a software architecture pattern that separates an application into three main components: Model, View, and Controller, making it easier to manage and maintain the codebase. It also allows for the reusability of components and promotes a more modular approach to software development.



<https://www.geeksforgeeks.org/mvc-design-pattern/>

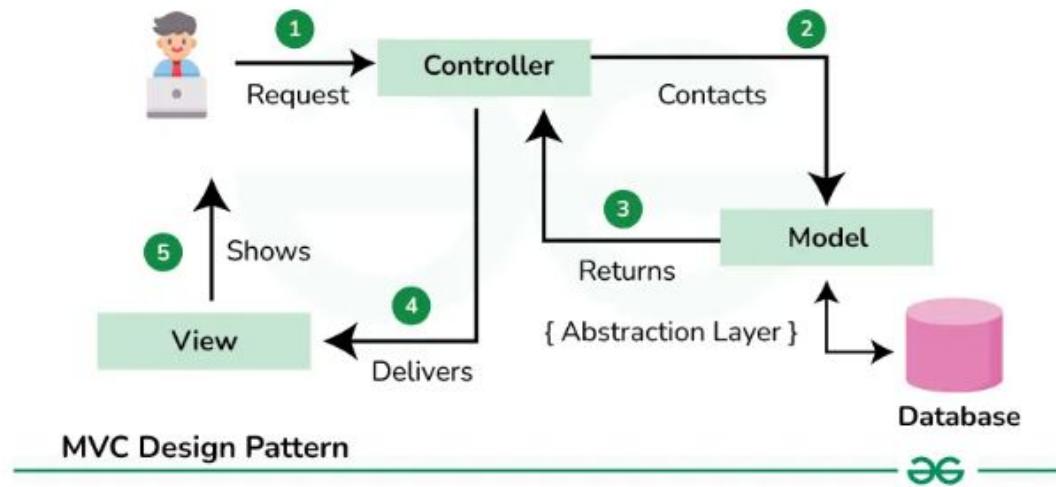
MVC Design Pattern

What is the MVC Design Pattern?

The **Model View Controller** (MVC) design pattern specifies that an application consists of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

- The MVC pattern separates the concerns of an application into three distinct components, each responsible for a specific aspect of the application's functionality.
- This separation of concerns makes the application easier to maintain and extend, as changes to one component do not require changes to the other components.

Components of the MVC Design Pattern



Spring Controller Tutorial

Form Method - GET and POST

The Method Attribute

The `method` attribute specifies the HTTP method to be used when submitting the form data.

The form-data can be sent as URL variables (with `method="get"`) or as HTTP post transaction (with `method="post"`).

The default HTTP method when submitting form data is GET.

Example

This example uses the GET method when submitting the form data:

```
<form action="/action_page.php" method="get">
```

[Try it Yourself »](#)

Example

This example uses the POST method when submitting the form data:

```
<form action="/action_page.php" method="post">
```

[Try it Yourself »](#)

https://www.w3schools.com/html/html_forms_attributes.asp

Form Method - GET and POST

Notes on GET:

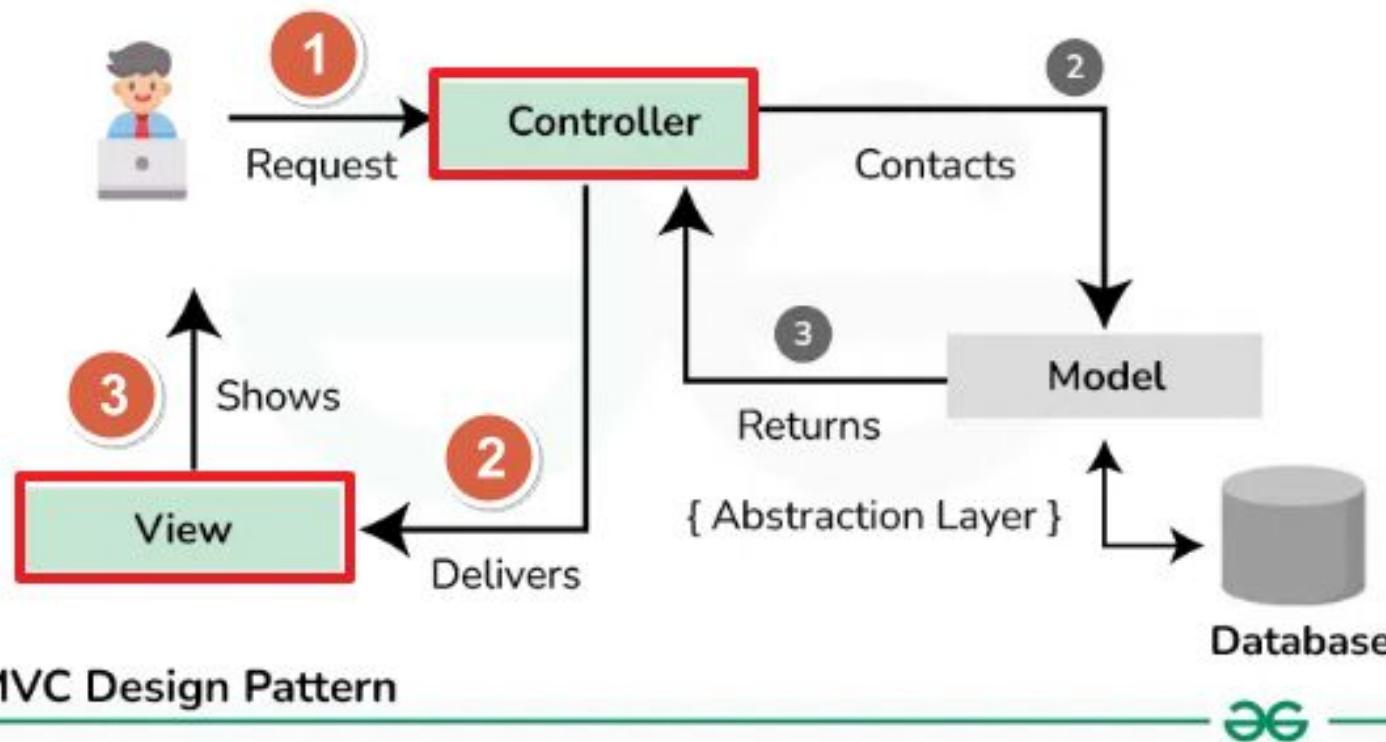
- Appends the form data to the URL, in name/value pairs
- NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
- The length of a URL is limited (2048 characters)
- Useful for form submissions where a user wants to bookmark the result
- GET is good for non-secure data, like query strings in Google

Notes on POST:

- Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)
- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

https://www.w3schools.com/html/html_forms_attributes.asp

Spring Controller



<https://www.baeldung.com/spring-controllers>

Spring Controller

@Controller <

@RequestMapping

- @GetMapping <
- @PostMapping
- @PutMapping
- @DeleteMapping

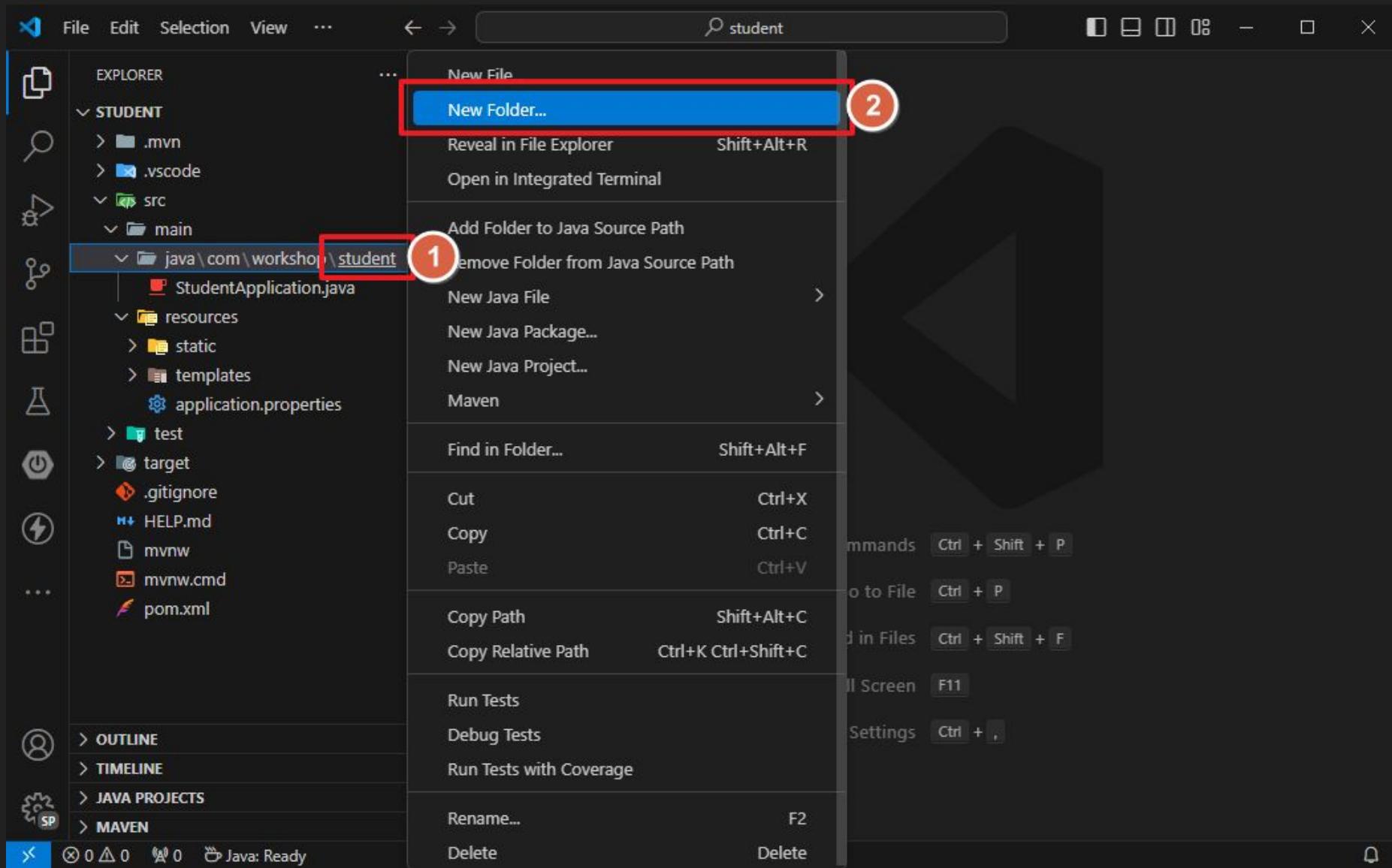
@RequestParam, @PathVariable

@RequestBody

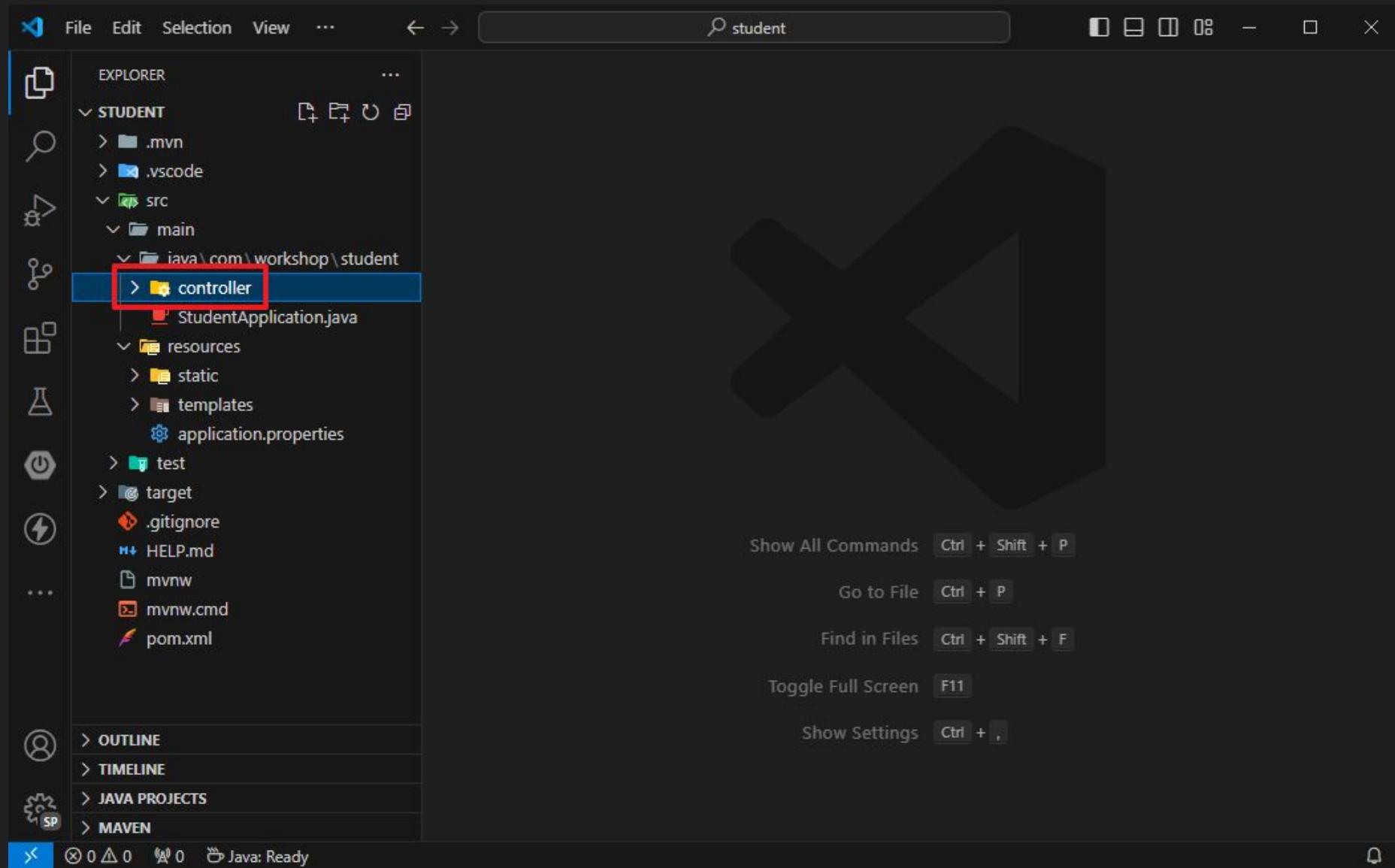
@ResponseBody, @ResponseEntity

@RestController

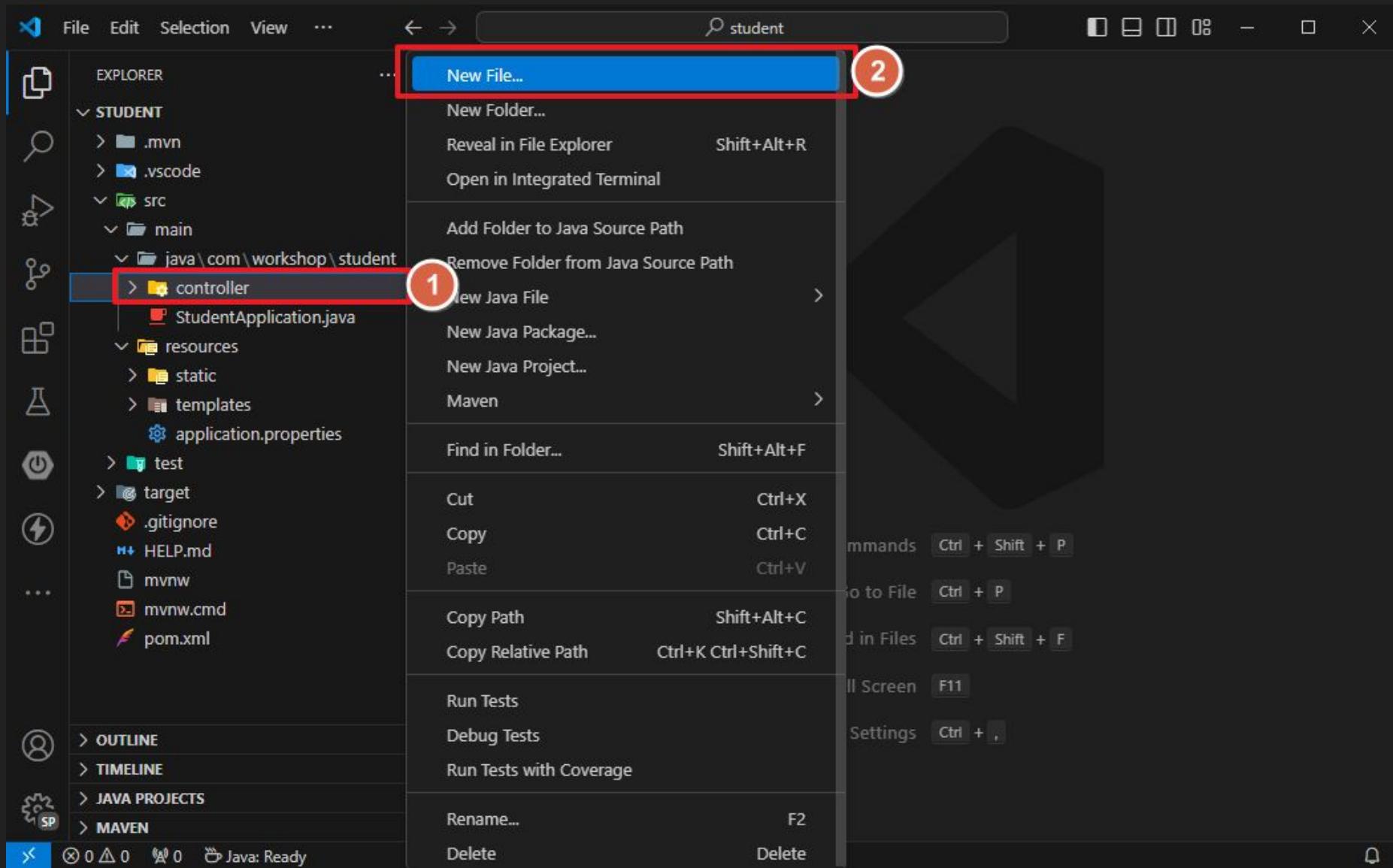
Spring Controller



Spring Controller



Spring Controller



Spring Controller

The screenshot shows a Java file named `TutorialController.java` open in the editor. The code completion dropdown is displayed, highlighting the class definition:

```
1 package com.workshop.student.controller;
2
3 public class TutorialController {
4
5 }
```

The dropdown menu lists several options related to the class, with the first item, "class TutorialController", highlighted by a red rectangle.

The left sidebar shows the project structure under the `STUDENT` folder:

- `.mvn`
- `.vscode`
- `src`
 - `main`
 - `java\com\workshop\student`
 - `controller`
 - `TutorialController.java`
 - `StudentApplication.java`
 - `resources`
 - `static`
 - `templates`
 - `application.properties`
 - `test`
 - `target`
 - `.gitignore`
 - `HELP.md`
 - `mvnw`
 - `mvnw.cmd`
 - `pom.xml`

The bottom status bar indicates "Java: Ready".

Spring Controller

The screenshot shows a Java project structure in the Explorer view of VS Code. The project is named 'STUDENT' and contains the following files and folders:

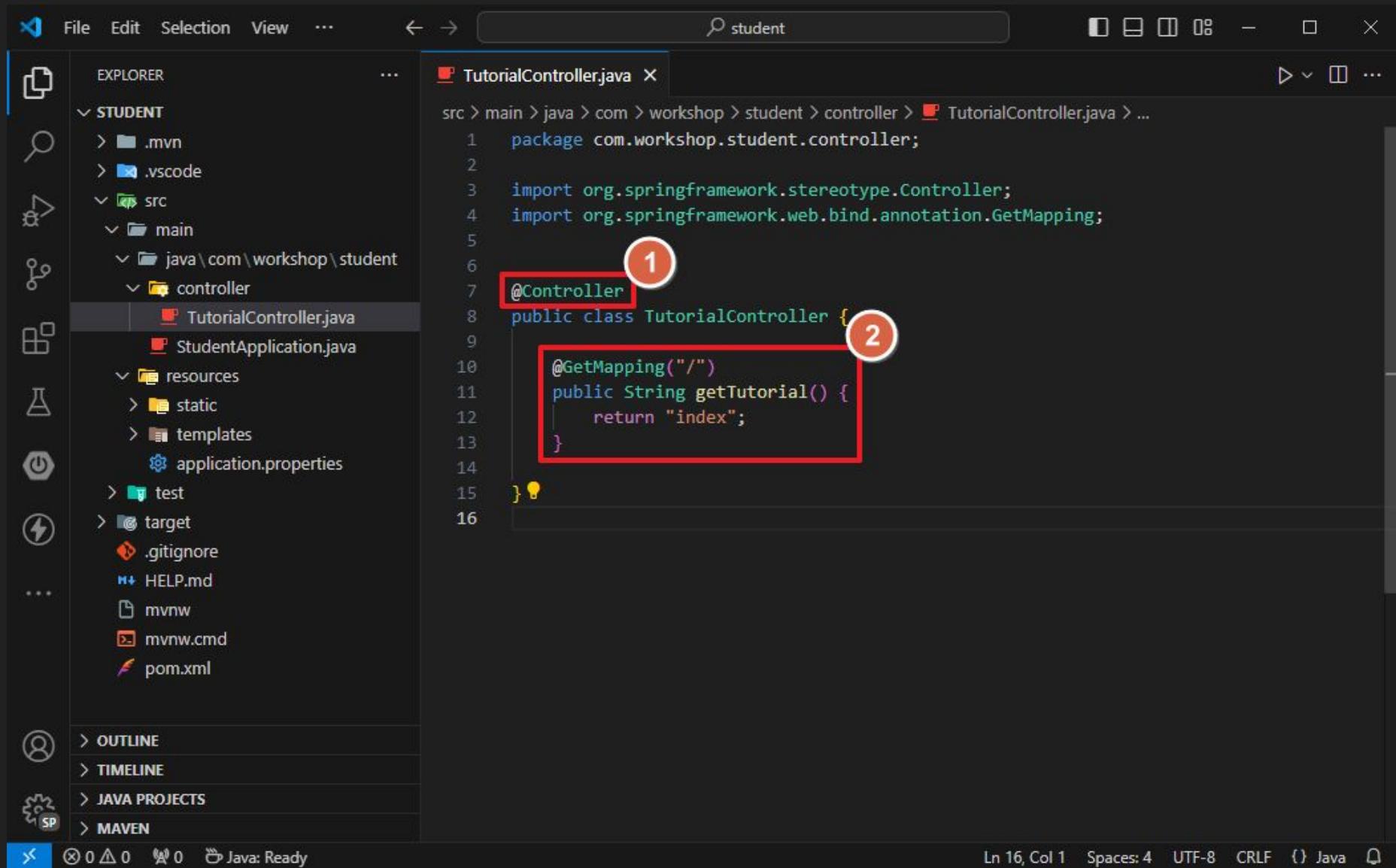
- .mvn
- .vscode
- src
 - main
 - java\com\workshop\student
 - controller
 - TutorialController.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

The 'TutorialController.java' file is open in the editor, showing the following code:

```
1 package com.workshop.student.controller;
2
3 public class TutorialController {
4
5     }
6 }
```

The status bar at the bottom indicates 'Java: Ready'.

Spring Controller - method GET



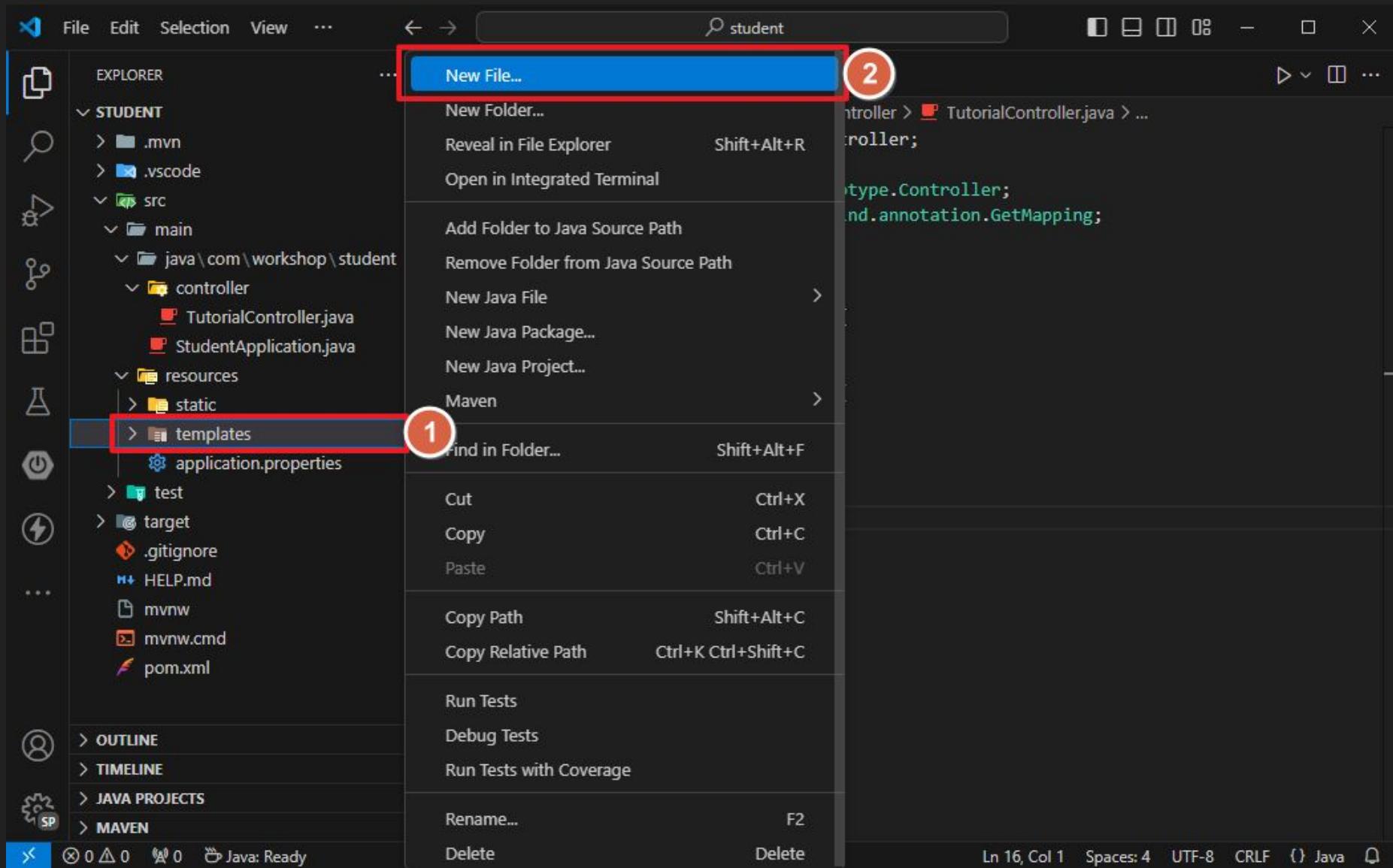
The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the 'STUDENT' folder:
 - .mvn
 - .vscode
 - src
 - main
 - java\com\workshop\student
 - controller (TutorialController.java)
 - StudentApplication.java
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - TutorialController.java (Editor):** The code defines a Spring controller.

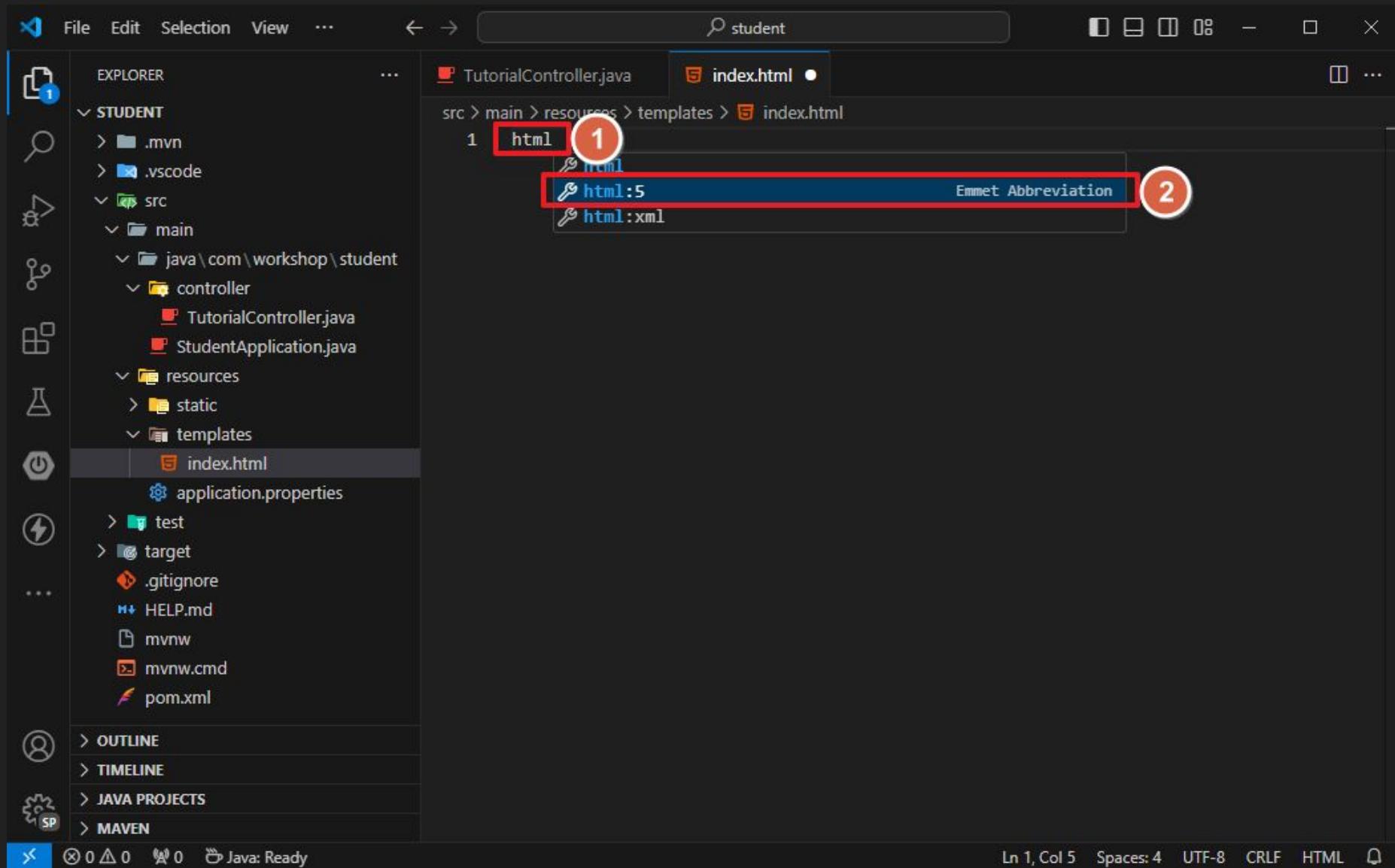
```
1 package com.workshop.student.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5
6
7 @Controller
8 public class TutorialController {
9
10     @GetMapping("/")
11     public String getTutorial() {
12         return "index";
13     }
14
15 }
16
```

Annotations are highlighted with red boxes and numbered circles:
 - Annotation 1:** `@Controller` (line 7)
 - Annotation 2:** `@GetMapping("/")` (line 10)
 - Bottom Status Bar:** Shows build status (0 errors, 0 warnings), Java: Ready, and file information (Ln 16, Col 1, Spaces: 4, UTF-8, CRLF).

Spring Controller - Template File



Spring Controller - Template File



Spring Controller - Template File

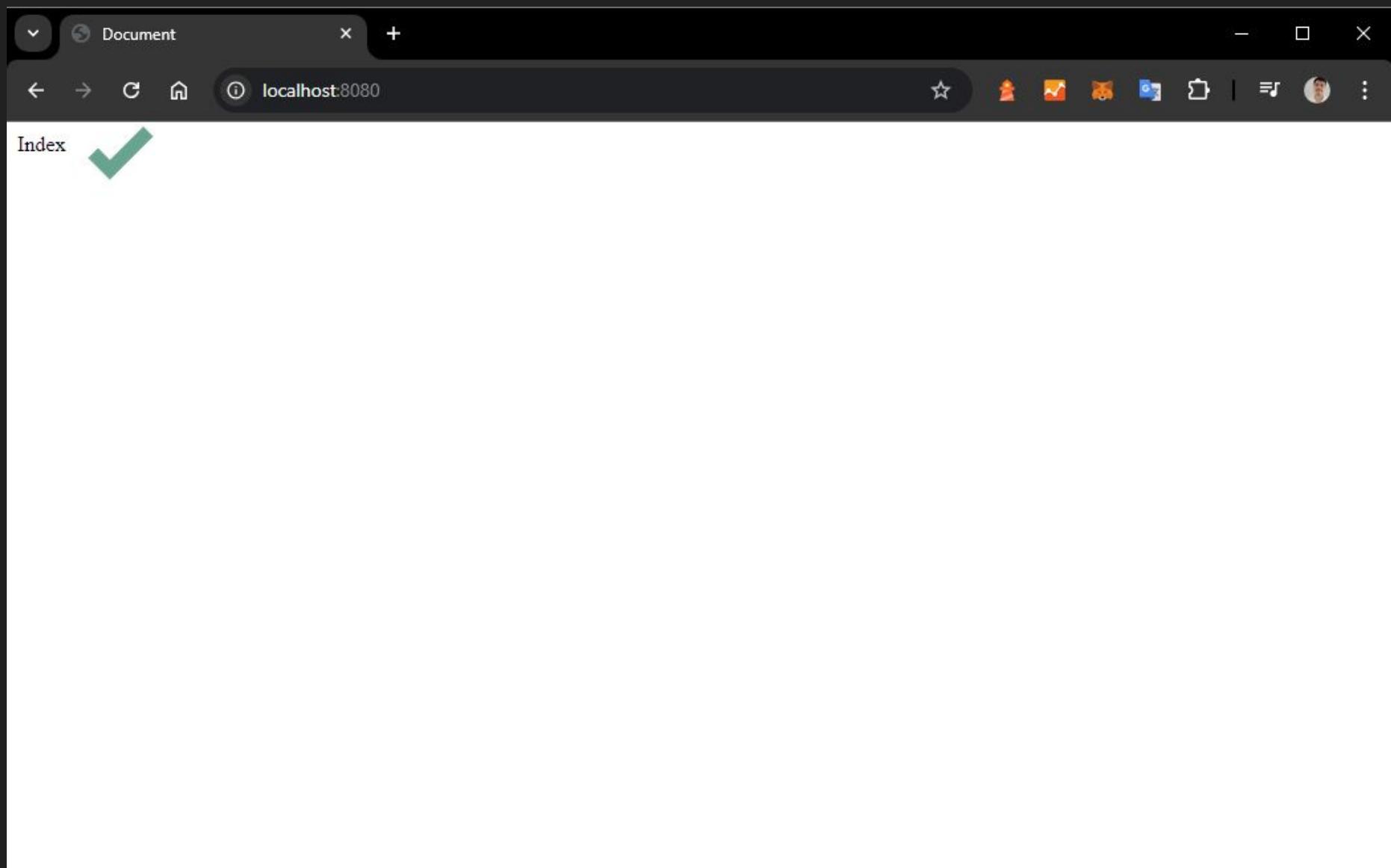
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder:
 - .mvn
 - .vscode
 - src
 - main
 - java\com\workshop\student
 - controller
 - TutorialController.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - index.html
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - Search Bar:** Contains the text "student".
 - Editor (Center):** Displays the content of "index.html".

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    Index
</body>
</html>
```

The word "Index" is highlighted with a red rectangle.
 - Bottom Status Bar:** Shows "Ln 9, Col 10" and "Spaces: 4" along with other status indicators like "Java: Ready".

Spring Controller - Template File



Spring Controller - Template File

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder:
 - .mvn
 - .vscode
 - src
 - main
 - java\com\workshop\student
 - controller
 - TutorialController.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - index.html
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - Code Editor (Center):** Displays `TutorialController.java` with the following code:

```
package com.workshop.student.controller;

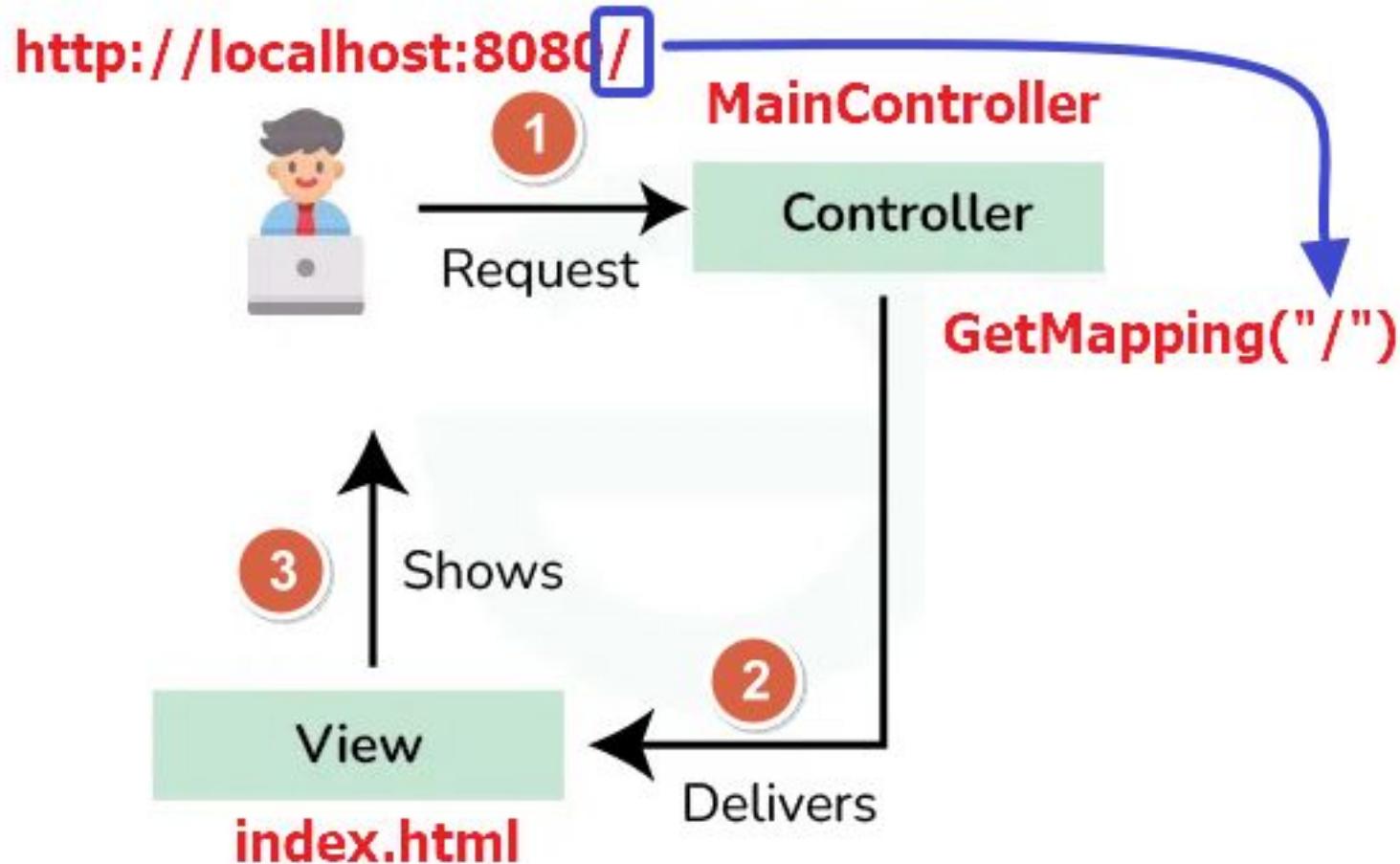
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class TutorialController {

    @GetMapping("/")
    public String getTutorial() {
        return "index";
    }
}
```
 - Terminal (Bottom):** Shows the application's startup logs:

```
dServer      : LiveReload server is running on port 35729
2024-08-20T15:00:10.207+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-08-20T15:00:10.219+07:00 INFO 2928 --- [student] [ restartedMain] c.workshop.student.StudentApplication : Started StudentApplication in 15.628 seconds (process running for 18.171)
2024-08-20T15:00:13.194+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:00:13.195+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:00:13.198+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
```
 - Status Bar (Bottom):** Shows "Java: Ready".

Spring Controller



Spring Controller

@Controller ✓

@RequestMapping

- **@GetMapping ✓**
- **@PostMapping**
- **@PutMapping**
- **@DeleteMapping**

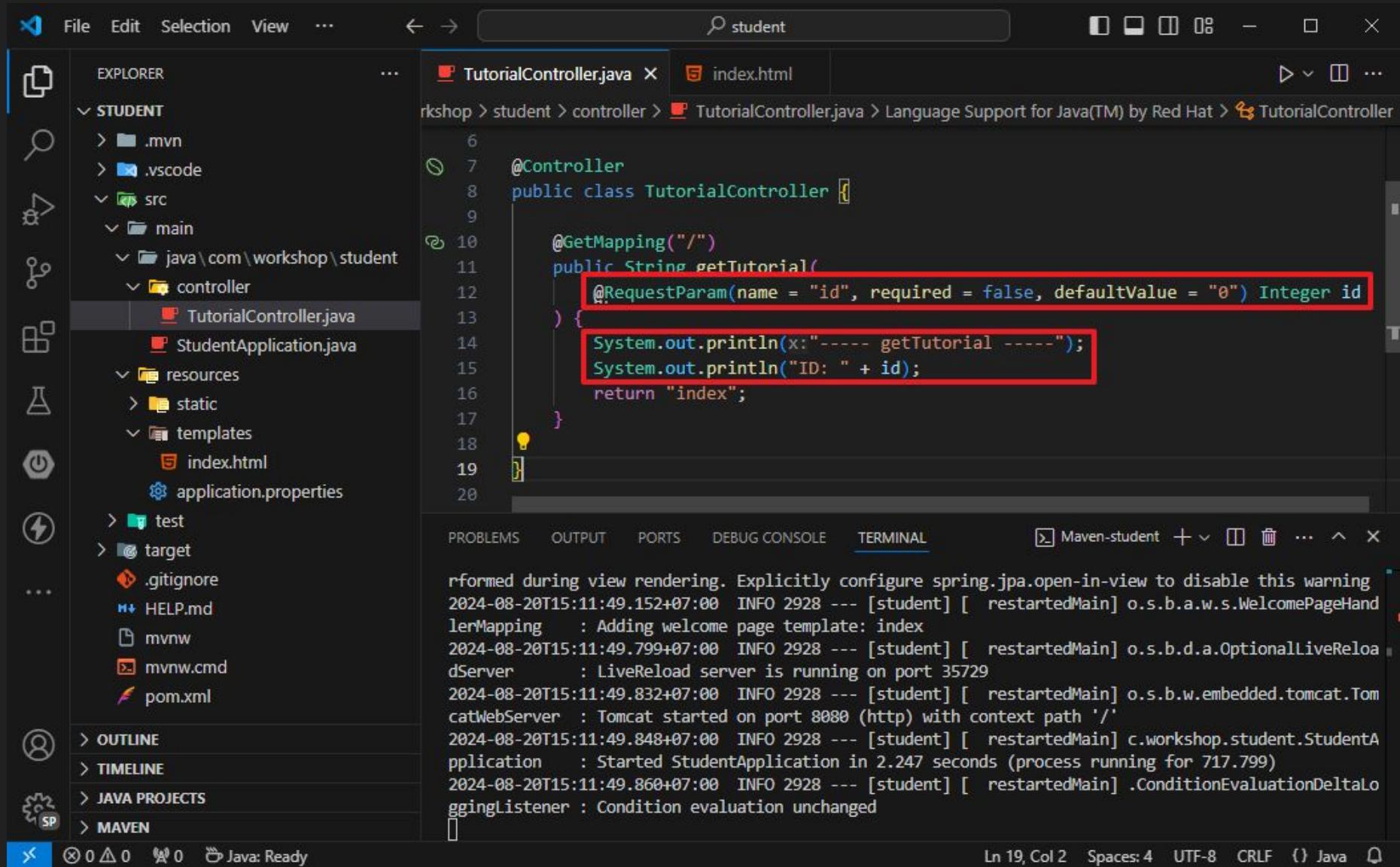
@RequestParam <, @PathVariable <

@RequestBody

@ResponseBody, @ResponseEntity

@RestController

Spring Controller - method GET (Request Param)



The screenshot shows a Java Spring application in a VS Code editor. The code in `TutorialController.java` defines a controller with a method that handles a GET request to the root URL. The method uses a `@RequestParam` annotation to get an optional parameter named `id` with a default value of 0. The annotated code is highlighted with a red box.

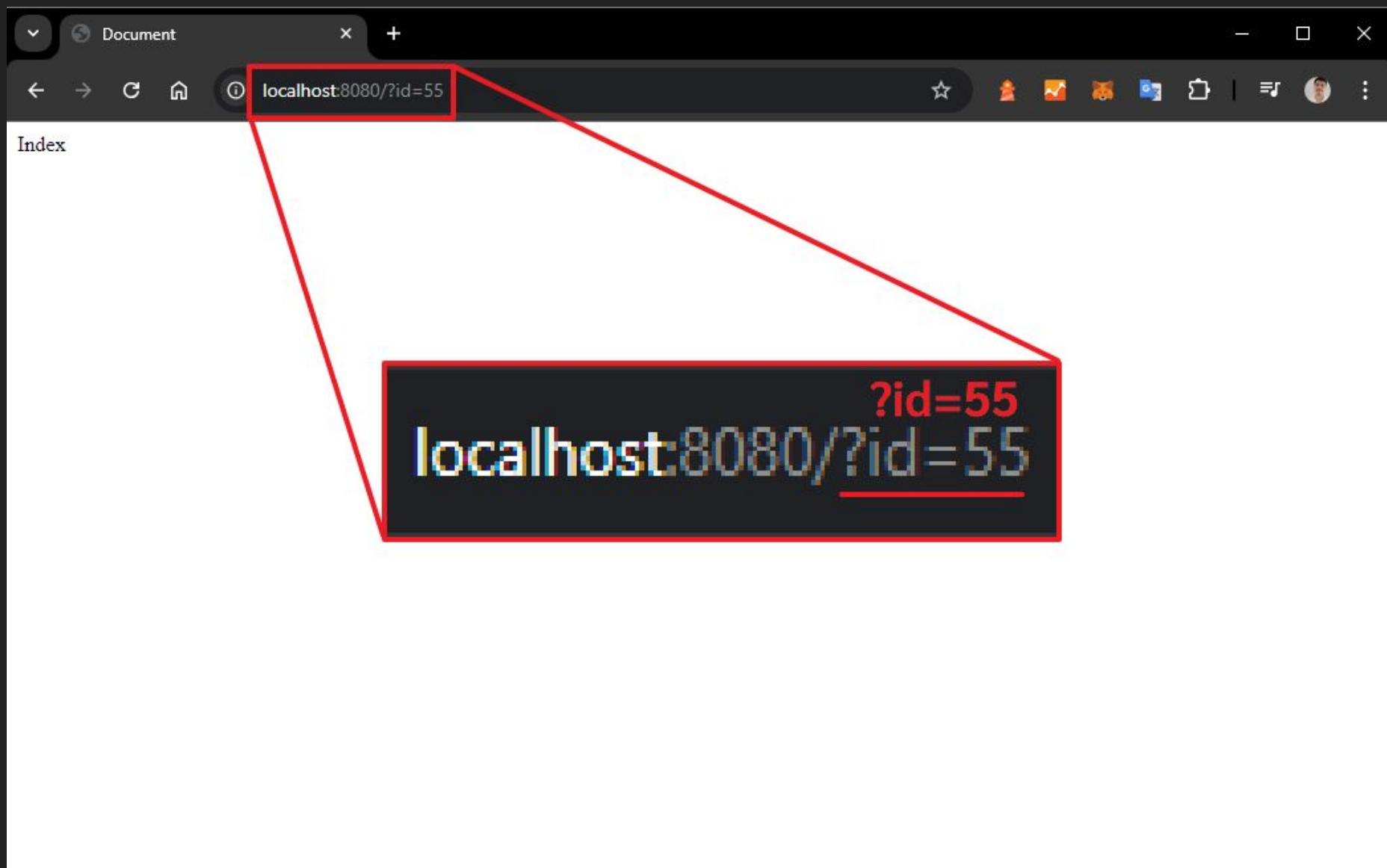
```
6
7  @Controller
8  public class TutorialController {
9
10     @GetMapping("/")
11     public String getTutorial(
12         @RequestParam(name = "id", required = false, defaultValue = "0") Integer id
13     ) {
14         System.out.println("----- getTutorial -----");
15         System.out.println("ID: " + id);
16
17     }
18
19 }
```

The terminal below shows the application's startup logs, indicating it has started successfully and is running on port 35729.

```
reformed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-08-20T15:11:49.152+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index
2024-08-20T15:11:49.799+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-08-20T15:11:49.832+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-08-20T15:11:49.848+07:00 INFO 2928 --- [student] [ restartedMain] c.workshop.student.StudentApplication : Started StudentApplication in 2.247 seconds (process running for 717.799)
2024-08-20T15:11:49.860+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
```

At the bottom, the status bar indicates "Java: Ready".

Spring Controller - method GET (Request Param)



Spring Controller - method GET (Request Param)

The screenshot shows a Java application running in a Spring framework environment. The code in `TutorialController.java` defines a controller method `getTutorial` that prints the value of the `id` parameter to the console. The terminal output shows the printed message and the value `55`.

```
6
7  @Controller
8  public class TutorialController {
9
10     @GetMapping("/")
11     public String getTutorial(
12         @RequestParam(name = "id", required = false, defaultValue = "0") Integer id
13     ) {
14         System.out.println("----- getTutorial -----");
15         System.out.println("ID: " + id);
16         return "index";
17     }
18 }
19
20
```

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

```
application : Started StudentApplication in 2.247 seconds (process running for 717.799)
2024-08-20T15:11:49.860+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLo
ggingListener : Condition evaluation unchanged
2024-08-20T15:12:40.006+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localho
st].[] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:12:40.007+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherS
ervlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:12:40.009+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherS
ervlet : Completed initialization in 2 ms
----- getTutorial -----
ID: 55
```

Ln 19, Col 2 Spaces: 4 UTF-8 CRLF {} Java

Spring Controller - method GET (Request Param)

The screenshot shows the Thunder Client interface for testing a Spring Controller. The URL in the address bar is highlighted with a red box and labeled '1'. The 'Send' button is also highlighted with a red box and labeled '2'.

Activity tab is selected. A recent activity is listed:

- GET localhost:8080 just now
- POST localhost:8080 15 days ago

The 'Query' tab is selected in the request configuration. The URL is set to `localhost:8080/?id=55`. The 'Headers' tab shows two entries: 'Content-Type: application/json' and 'User-Agent: Thunder Client'. The 'Body' tab is empty.

Query Parameters section contains:

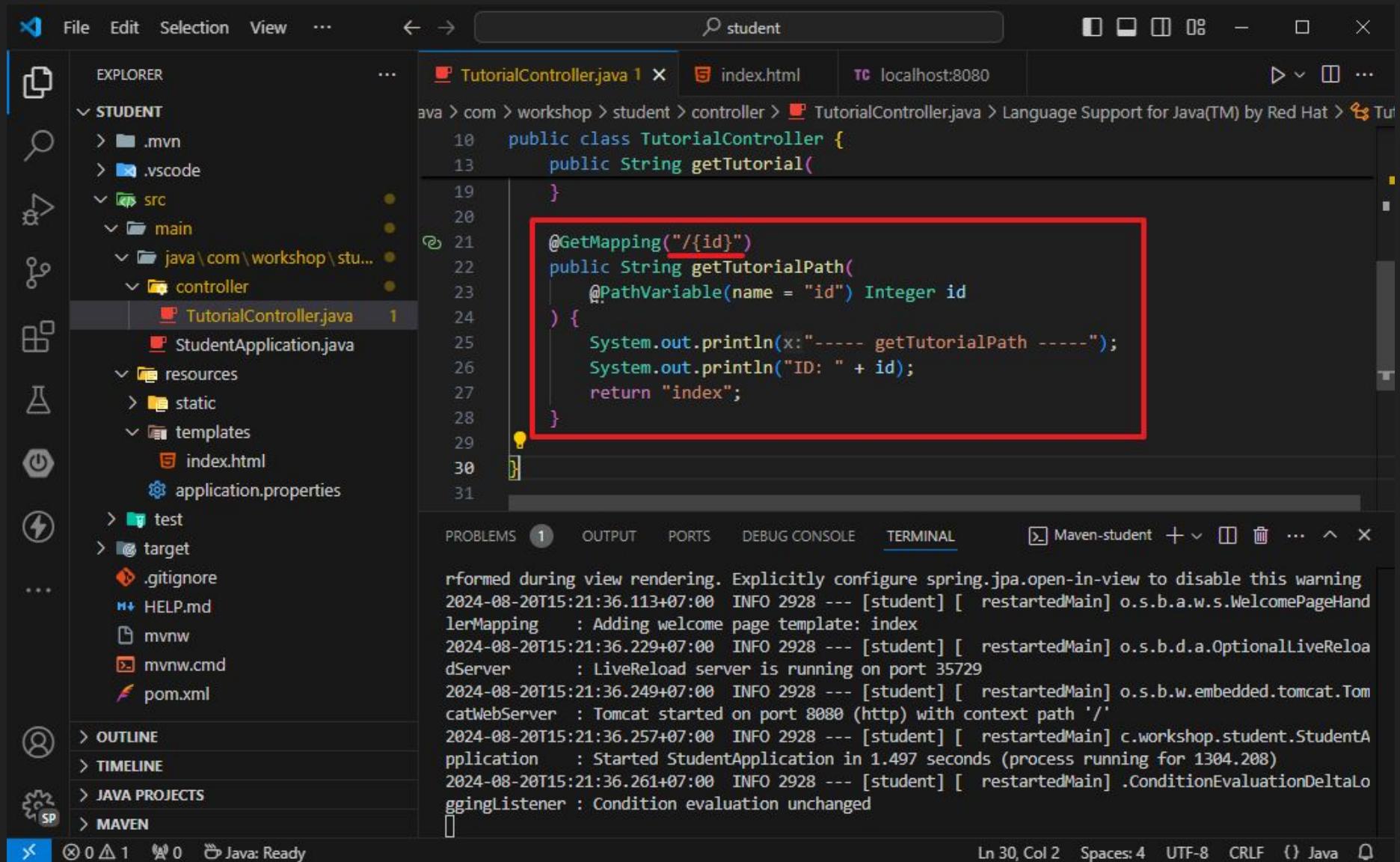
- id: 55
- parameter: value

The response status is **200 OK**, size is **220 Bytes**, and time is **10 ms**. The response body shows the logs from the Spring application, including the successful retrieval of the tutorial with ID 55.

```
ggingListener : Condition evaluation unchanged
2024-08-20T15:12:40.006+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:12:40.007+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:12:40.009+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
----- getTutorial -----
ID: 55
----- getTutorial -----
ID: 55
```

The bottom status bar indicates **Java: Ready**.

Spring Controller - method GET (Path Variable)



The screenshot shows a Java Spring application in a VS Code editor. The code in `TutorialController.java` defines a method `getTutorialPath` with a `@GetMapping` annotation and a `@PathVariable` annotation for the parameter `id`.

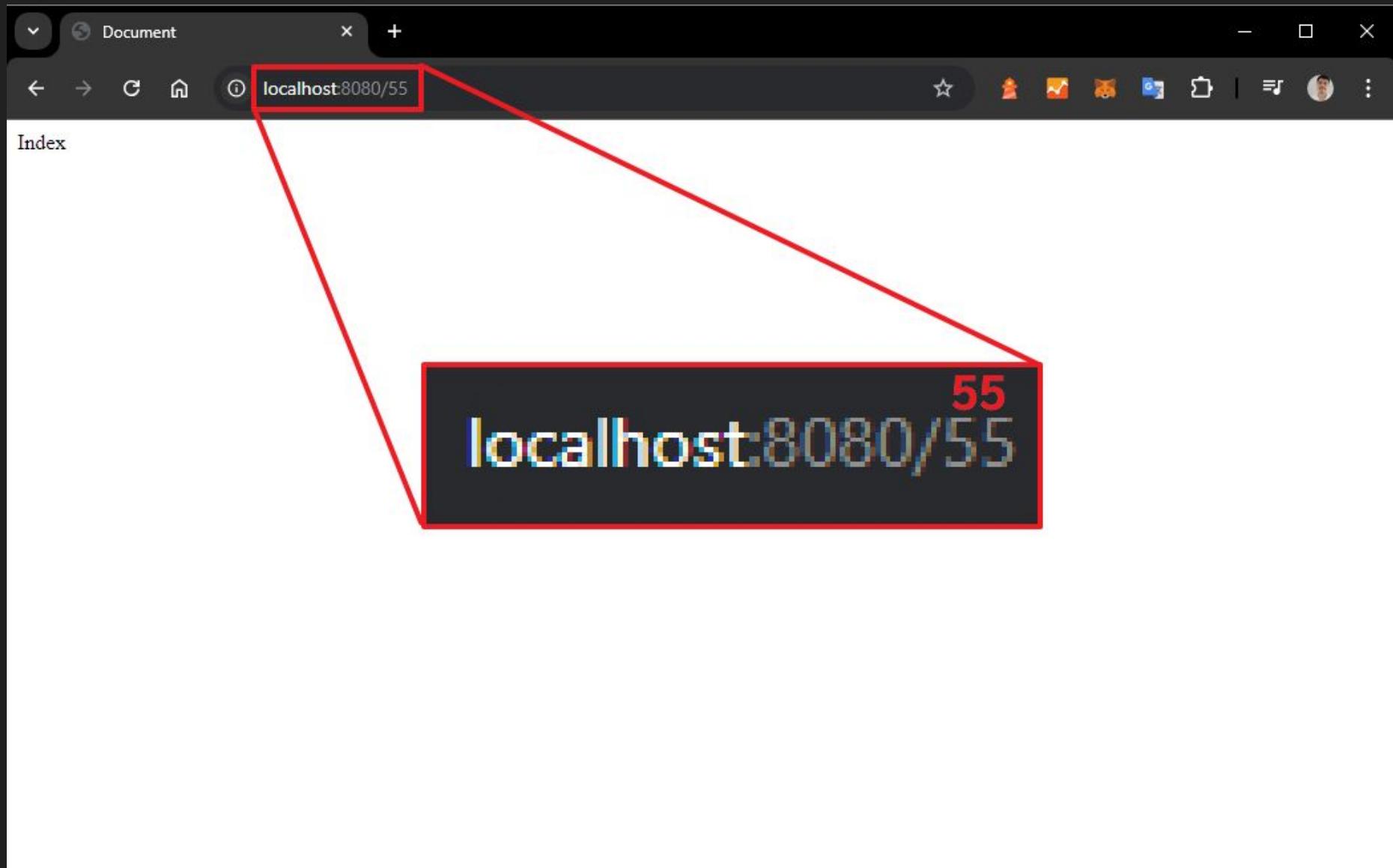
```
public class TutorialController {
    public String getTutorialPath(
        @GetMapping("/{id}")
        @PathVariable(name = "id") Integer id
    ) {
        System.out.println("----- getTutorialPath -----");
        System.out.println("ID: " + id);
        return "index";
    }
}
```

The code editor has a red box highlighting the `@GetMapping` and `@PathVariable` annotations. The terminal below shows the application's startup logs.

```
reformed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-08-20T15:21:36.113+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.a.w.s.WelcomeHandlerMapping : Adding welcome page template: index
2024-08-20T15:21:36.229+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-08-20T15:21:36.249+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-08-20T15:21:36.257+07:00 INFO 2928 --- [student] [ restartedMain] c.workshop.student.StudentApplication : Started StudentApplication in 1.497 seconds (process running for 1304.208)
2024-08-20T15:21:36.261+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
```

VS Code status bar: Ln 30, Col 2 Spaces: 4 UTF-8 CRLF {} Java

Spring Controller - method GET (Path Variable)



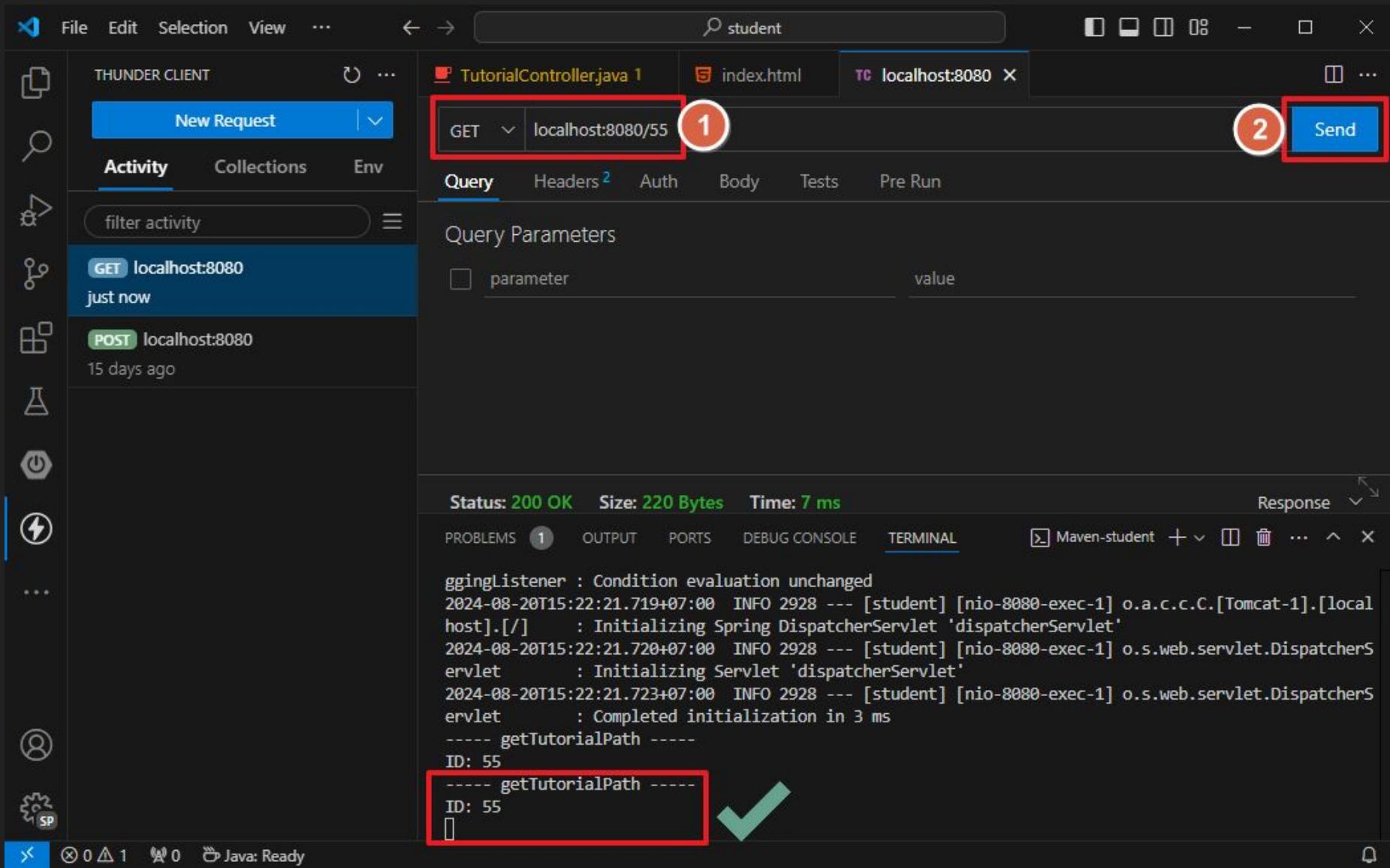
Spring Controller - method GET (Path Variable)

The screenshot shows a Java application running in a Spring framework environment. The code in `TutorialController.java` defines a method `getTutorialPath` that prints the value of the `id` path variable to the console. The terminal output shows the application starting and the specific log entry for this method execution.

```
public class TutorialController {
    public String getTutorialPath(
        @PathVariable(name = "id") Integer id
    ) {
        System.out.println("----- getTutorialPath -----");
        System.out.println("ID: " + id);
        return "index";
    }
}
```

```
application : Started StudentApplication in 1.497 seconds (process running for 1304.208)
2024-08-20T15:21:36.261+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
2024-08-20T15:22:21.19+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:22:21.720+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:22:21.723+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
----- getTutorialPath -----
ID: 55
```

Spring Controller - method GET (Path Variable)



Spring Controller

@Controller ✓

@RequestMapping

- **@GetMapping ✓**
- **@PostMapping**
- **@PutMapping**
- **@DeleteMapping**

@RequestParam ✓ , @PathVariable ✓

@RequestBody

@ResponseBody, @ResponseEntity

@RestController

Spring Controller - method POST

The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (Explorer):**
 - STUDENT** folder:
 - .mvn
 - .vscode
 - src** folder
 - main
 - java\com\workshop\student
 - controller
 - TutorialController.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - index.html
 - application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
- Central Area (Code Editor):** TutorialController.java (2 tabs)

```
public class TutorialController {  
    public String getTutorialPath()  
        return "index";  
    }  
  
    @PostMapping("/")  
    public String postTutorial(  
        @RequestParam(name = "id", required = false, defaultValue = "0") Integer id  
    ) {  
        System.out.println("---- postTutorial ----");  
        System.out.println("ID: " + id);  
        return "index";  
    }  
}
```
- Bottom Status Bar:** Ln 42, Col 2 Spaces: 4 UTF-8 CRLF {} Java
- Terminal:** Maven-student output showing application startup logs.

Spring Controller - method POST

The screenshot shows the Thunder Client interface for making a POST request to the endpoint `localhost:8080`. The request body is set to `Form` and contains a field `id` with the value `55`. The response status is `200 OK`, and the output window shows the logs of the Spring application starting and the successful execution of the `postTutorial` method.

1. Target endpoint: `localhost:8080`
2. Request Body type: `Form`
3. Form field: `id` with value `55`
4. Response status: `200 OK`
5. Send button

```
application : Started StudentApplication in 1.302 seconds (process running for 1753.278)
2024-08-20T15:29:05.331+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
2024-08-20T15:29:43.642+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:29:43.642+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:29:43.645+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
----- postTutorial -----
ID: 55
```

Spring Controller - method POST (Request Param) multiple value by Map

The screenshot shows a Java Spring application in VS Code. The code editor displays `TutorialController.java` with the following content:

```
public class TutorialController {
    ...
    @PostMapping("/")
    public String postTutorial(
        @RequestParam() Map<String, String> param
    ) {
        System.out.println("---- postTutorial ----");
        System.out.println("ID: " + param.get(key:"id"));
        System.out.println("Code: " + param.get(key:"code"));
        return index ;
    }
}
```

The code editor highlights the `@RequestParam()` annotation and the block of code that prints the parameters. The terminal below shows the application output:

```
reformed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-08-20T15:33:37.988+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index
2024-08-20T15:33:38.162+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-08-20T15:33:38.176+07:00 INFO 2928 --- [student] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-08-20T15:33:38.182+07:00 INFO 2928 --- [student] [ restartedMain] c.workshop.student.StudentApplication : Started StudentApplication in 1.586 seconds (process running for 2026.133)
2024-08-20T15:33:38.185+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
```

The status bar at the bottom indicates "Java: Ready".

Spring Controller - method POST (Request Param) multiple value by Map

The screenshot shows the Thunder Client interface for making a POST request to the endpoint `localhost:8080`. The request body is defined as a Form with two fields: `id` (value 55) and `code` (value 650). The response status is 200 OK, and the output shows the successful execution of the `postTutorial` method with the provided parameters.

1. Method selection: POST

2. Request type: Body

3. Content type: Form

4. Form fields: id (55), code (650)

5. Send button

Status: 200 OK
Size: 220 Bytes
Time: 274 ms

```
2024-08-20T15:33:38.185+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
2024-08-20T15:34:11.162+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:34:11.165+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:34:11.168+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
---- postTutorial ----
ID: 55
Code: 650
```

Spring Controller

@Controller ✓

@RequestMapping <

- @GetMapping ✓
- @PostMapping ✓
- @PutMapping
- @DeleteMapping

@RequestParam ✓ , @PathVariable ✓

@RequestBody

@ResponseBody, @ResponseEntity

@RestController

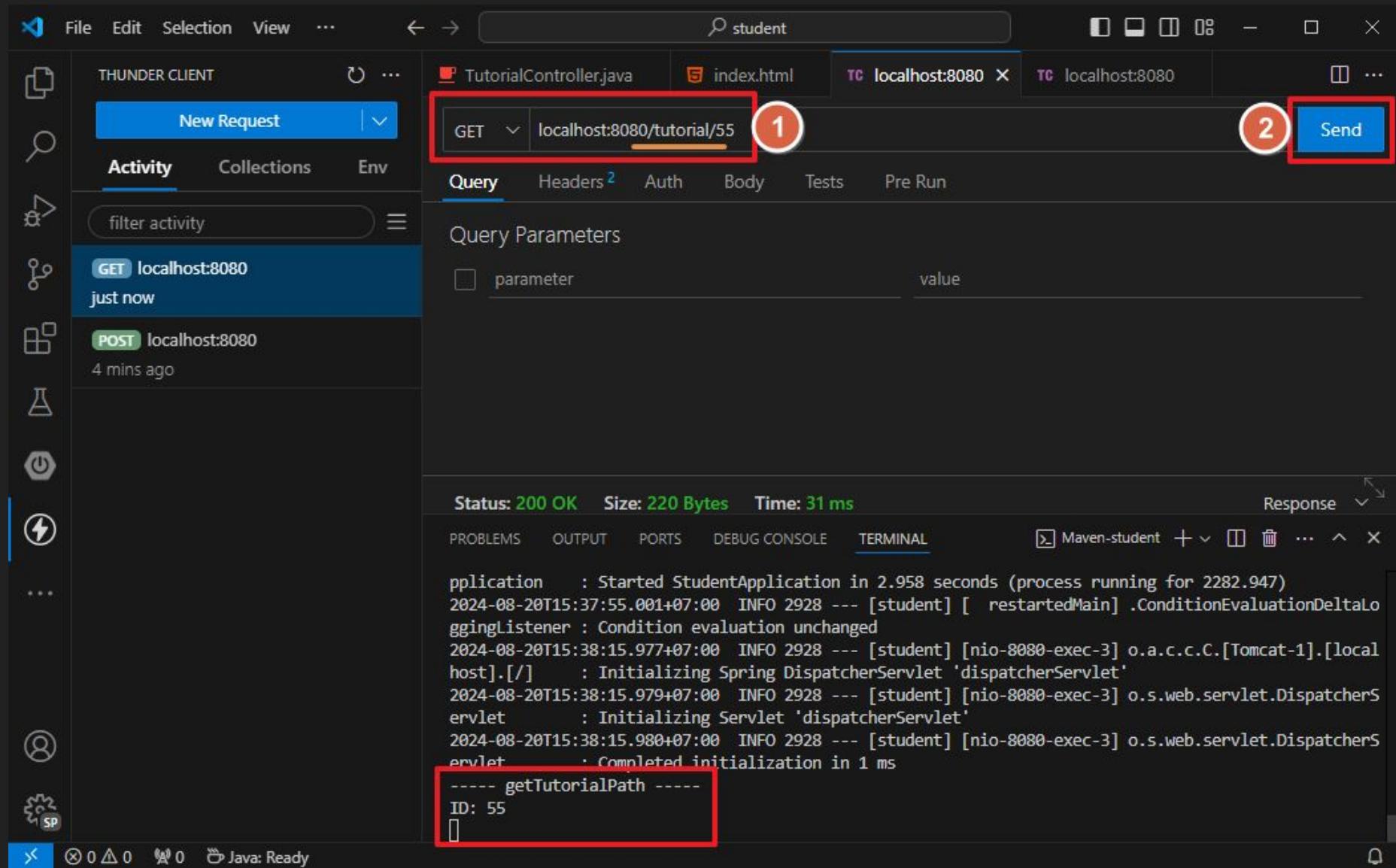
Spring Controller @RequestMapping

The screenshot shows a Java Spring application structure in VS Code. The project tree on the left shows a package named com.workshop.student.controller containing a file named TutorialController.java. The code editor displays the following Java code:

```
4  
5 import org.springframework.stereotype.Controller;  
6 import org.springframework.web.bind.annotation.GetMapping;  
7 import org.springframework.web.bind.annotation.PathVariable;  
8 import org.springframework.web.bind.annotation.RequestParam;  
9 import org.springframework.web.bind.annotation.PostMapping;  
10 import org.springframework.web.bind.annotation.RequestMapping;  
11  
12  
13 @Controller  
14 @RequestMapping("/tutorial")  
15 public class TutorialController {  
16  
17     @GetMapping("/")  
18     public String getTutorial(  
19         @RequestParam(name = "id", required = false, defaultValue = "0") Integer id  
20     ) {  
21         System.out.println("---- getTutorial ----");  
22         System.out.println("ID: " + id);  
23         return "index";  
24     }  
25  
26     @GetMapping("/{id}")  
27     public String getTutorialPath(  
28         @PathVariable(name = "id") Integer id  
29     ) {  
30         System.out.println("---- getTutorialPath ----");  
31         System.out.println("ID: " + id);  
32     }  
33 }
```

The line `@RequestMapping("/tutorial")` is highlighted with a red rectangle. The status bar at the bottom indicates "Java: Ready".

Spring Controller @RequestMapping



Spring Controller

@Controller ✓

@RequestMapping ✓

- **@GetMapping ✓**
- **@PostMapping ✓**
- **@PutMapping**
- **@DeleteMapping**

@RequestParam ✓ , @PathVariable ✓

@RequestBody

@ResponseBody, @ResponseEntity

@RestController

Spring Controller Workshop

CRUD

API - Faculty

(1) getAll

method **GET** - /faculty

READ

(2) getById

method **GET** - /faculty/{faculty-id}

(3) insertFaculty, updateFaculty

method **POST**

- insert - /faculty/
- update - /faculty/{faculty-id}

CREATE & UPDATE

(4) deleteFaculty

method **GET** - /faculty/delete/{faculty-id}

DELETE

Spring Controller - Create Controller Faculty

The screenshot shows a dark-themed interface of the Visual Studio Code (VS Code) code editor. In the top left, the menu bar includes File, Edit, Selection, View, and others. A search bar labeled "student" is positioned above the tabs. The tabs at the top show TutorialController.java, FacultyController.java (which is the active tab), and index.html.

The left sidebar, titled "EXPLORER", displays the project structure under "STUDENT". It includes a ".mvn" folder, a ".vscode" folder, and a "src" folder containing "main" and "resources". Inside "main", there is a "java\com\workshop\student" package with a "controller" folder. Inside "controller", three files are listed: FacultyController.java (highlighted with a red box and circled with a red number 1), TutorialController.java, and StudentApplication.java. Below "controller", there are "resources" and "static" folders, and a "templates" folder containing "index.html" and "application.properties".

The main code editor area shows the beginning of a Java class:

```
1 package com.workshop.student.controller;
2
3 public class FacultyController {
4
5 }
6
```

A red box highlights the entire code block, and a red circle with the number 2 points to the closing brace "}" on line 5. The status bar at the bottom right shows "Ln 6, Col 1" and other details like "Spaces: 4", "UTF-8", "CRLF", and "Java".

Spring Controller - Create Method

The screenshot shows the VS Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close, etc.
- Explorer:** Shows the project structure under STUDENT:
 - .mvn
 - .vscode
 - src
 - main
 - java\com\workshop\stu...
 - controller
 - FacultyController.java
 - TutorialController.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - index.html
 - application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN
 - Bottom Status Bar: L 0 Col 6 Spaces: 4 UTF-8 CRLF {} Java

The code editor displays `FacultyController.java` with the following content:

```
11  @Controller
12  @RequestMapping("/faculty")
13  public class FacultyController {
14
15      @GetMapping("", "/")
16      public String getAll() {
17          System.out.println("----- getAll() -----");
18          return "index";
19      }
20
21      @GetMapping("/{faculty-id}")
22      public String getById(
23          @PathVariable(name = "faculty-id") Integer facultyId
24      ) {
25          System.out.println("----- getById() -----");
26          System.out.println("faculty-id: " + facultyId);
27          return "index";
28      }
29
30      @GetMapping("/delete/{faculty-id}")
31      public String getDeleteById(
32          @PathVariable(name = "faculty-id") Integer facultyId
33      ) {
34          System.out.println("----- getDeleteById() -----");
35          System.out.println("faculty-id: " + facultyId);
36          return "index";
37      }
38 }
```

Three sections of the code are highlighted with red boxes:

 - Line 11: `@Controller` and `@RequestMapping("/faculty")`
 - Lines 15-19: The `getAll()` method, which prints "----- getAll() -----" and returns "index".
 - Lines 21-27: The `getById()` method, which prints "----- getById() -----" and "faculty-id: " + facultyId, then returns "index".

Spring Controller - Create Method (Test)

The screenshot shows the Thunder Client interface for testing a Spring Controller. The top bar has tabs for 'TutorialController.java' and 'FacultyController.java 1'. The main area shows a 'New Request' dialog with a red box around the URL input field containing 'localhost:8080/faculty' (labeled 1). To the right is a 'Send' button with a red box and a circled '2'. Below the URL input are tabs for 'Query', 'Headers 2', 'Auth', 'Body', 'Tests', and 'Pre Run'. The 'Query' tab is selected. Under 'Query Parameters', there is a table with one row: 'parameter' and 'value'. At the bottom, the 'Response' tab is active, displaying the server logs. A red box highlights the log entry for the 'getAll()' method (labeled 3), which is preceded by a green checkmark icon.

Status: 200 OK Size: 220 Bytes Time: 35 ms

```
2024-08-20T15:53:00.282+07:00 INFO 2928 --- [student] [ restartedMain] c.workshop.student.StudentApplication : Started StudentApplication in 1.663 seconds (process running for 3188.232)
2024-08-20T15:53:00.284+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
2024-08-20T15:53:05.850+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:53:05.850+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:53:05.852+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
----- getAll() -----
```

Java: Ready

Spring Controller - Create Method (Test)

The screenshot shows the Thunder Client interface for testing a Spring Controller. The top navigation bar includes File, Edit, Selection, View, and a search bar for 'student'. Below the bar, tabs for TutorialController.java, FacultyController.java 1, and localhost:8080 are visible. A sidebar on the left lists activity logs: a recent GET request to localhost:8080 and a POST request from 19 mins ago.

In the main area, a 'New Request' dialog is open. The URL field contains 'localhost:8080/faculty/55' (marked with a red box and circled with a red number 1). To the right of the URL is a 'Send' button (marked with a red box and circled with a red number 2).

The 'Query' tab is selected. Below it, the 'Query Parameters' section shows a parameter named 'faculty-id' with a value of '55'.

The status bar at the bottom indicates a 200 OK response with 220 Bytes and 11 ms duration. The terminal pane shows Spring application logs:

```
2024-08-20T15:53:00.284+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
2024-08-20T15:53:05.850+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:53:05.850+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:53:05.852+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
----- getAll()
----- getById()
faculty-id: 55
```

A large green checkmark is overlaid on the terminal output for the 'getById()' line.

The bottom status bar shows Java: Ready.

Spring Controller - Create Method (Test)

The screenshot shows the Thunder Client interface for testing a Spring Controller. The top bar displays the search term "student" and several tabs: "TutorialController.java", "FacultyController.java 1", and "localhost:8080".

The main area shows a "New Request" dialog with a red box around the URL input field containing "localhost:8080/faculty/delete/55" and a red circle with the number "1" next to it. To the right of the URL is a "Send" button with a red box and a red circle with the number "2" next to it.

The "Activity" tab is selected, showing a list of recent requests:

- GET localhost:8080** just now
- POST localhost:8080** 19 mins ago

The "Query" tab is active, showing "Query Parameters" with a parameter named "faculty-id" set to "55".

The "Response" tab at the bottom shows the server logs and the response body, which includes a green checkmark icon. The response body contains the following text:

```
2024-08-20T15:53:05.850+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T15:53:05.850+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T15:53:05.852+07:00 INFO 2928 --- [student] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
----- getAll() -----
----- getById() -----
faculty-id: 55
----- getDeleteById() -----
faculty-id: 55
[]
```

The status bar at the bottom indicates "Java: Ready".

Spring Controller - Create Method

The screenshot shows a Java application named "student" in a Spring Boot environment. The code editor displays `FacultyController.java`. A red box highlights the `postInsertAndUpdate` method, which handles POST requests to the root URL. The method uses `@RequestParam` to extract parameters from the request body.

```
public class FacultyController {  
    public String getById(  
    ) {  
        System.out.println(x:"----- getById() -----");  
        System.out.println("faculty-id: " + facultyId);  
        return "index";  
    }  
  
    @GetMapping("/delete/{faculty-id}")  
    public String getDeleteById(  
        @PathVariable(name = "faculty-id") Integer facultyId  
    ) {  
        System.out.println(x:"----- getDeleteById() -----");  
        System.out.println("faculty-id: " + facultyId);  
        return "index";  
    }  
  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        @RequestParam() Map<String, String> param  
    ) {  
        System.out.println(x:"----- postInsertAndUpdate() -----");  
        System.out.println("faculty-id: " + param.get(key:"faculty-id"));  
        System.out.println("faculty-name: " + param.get(key:"faculty-name"));  
        return "index";  
    }  
}
```

The status bar at the bottom indicates "Java: Ready".

Spring Controller - Create Method

The screenshot shows the Thunder Client interface for testing a Spring Controller. The request path is set to `localhost:8080/faculty/`. The `Body` tab is selected, and the `Form` sub-tab is chosen. Two form fields are present: `faculty-id` with value `55` and `faculty-name` with value `Computer`. The `Send` button is highlighted with a red circle labeled `5`. The response shows a `200 OK` status with a log message indicating a successful `postInsertAndUpdate()` call.

THUNDER CLIENT

New Request

Activity Collections Env

filter activity

GET localhost:8080
11 mins ago

POST localhost:8080
just now

student

TutorialController.java FacultyController.java 1 localhost:8080 localhost:8080

POST localhost:8080/faculty/ 1

Query Headers 2 Auth Body 1 2 Pre Run

JSON XML Text Form 3 Form-encode GraphQL Binary

faculty-id 55

faculty-name Computer

field name value

Status: 200 OK Size: 220 Bytes Time: 52 ms

PROBLEMS 1 OUTPUT PORTS DEBUG CONSOLE TERMINAL Maven-student

```
2024-08-20T16:07:56.379+07:00 INFO 2928 --- [student] [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
2024-08-20T16:08:16.235+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-08-20T16:08:16.236+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-08-20T16:08:16.238+07:00 INFO 2928 --- [student] [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
----- postInsertAndUpdate() -----
faculty-id: 55
faculty-name: Computer
```

Java: Ready

API - Student

(1) getAll

method **GET** - /student

(2) getById

method **GET** - /student/{student-id}

(3) insertStudent, updateStudent

method **POST**

- insert - /student/
- update - /student/{student-id}

(4) deleteStudent

method **GET** - /student/delete/{student-id}

Spring Controller - Create Controller Student

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the 'STUDENT' folder. The 'controller' subfolder contains four files: CourseController.java, EnrollController.java, FacultyController.java, and StudentController.java, which is highlighted with a red box.
- Search Bar (Top):** Contains the text "student".
- Code Editor (Center):** Displays the code for StudentController.java. The code defines a controller for the "/student" path, mapping three methods: getAll(), getById(), and getDeleteById(). Each method prints the student ID to the console before returning "index".

```
12 @Controller
13 @RequestMapping("/student")
14 public class StudentController {
15
16     @GetMapping({"", "/"})
17     public String getAll() {
18         System.out.println("----- StudentController getAll() -----");
19         return "index";
20     }
21
22     @GetMapping("/{student-id}")
23     public String getById(
24         @PathVariable(name = "student-id") Integer studentId
25     ) {
26         System.out.println("----- StudentController getById() -----");
27         System.out.println("student-id: " + studentId);
28         return "index";
29     }
30
31     @GetMapping("/delete/{student-id}")
32     public String getDeleteById(
33         @PathVariable(name = "student-id") Integer studentId
34     ) {
35         System.out.println("----- StudentController getDeleteById() -----");
36         System.out.println("student-id: " + studentId);
37         return "index";
38     }
39 }
```
- Bottom Status Bar:** Shows build status (0 errors, 0 warnings), Java: Ready, and file statistics (Ln 41, Col 39, Spaces: 4, UTF-8, CRLF).

Spring Controller - Create Controller Student

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder. The "src/main/java/com/workshop/student/controller" package contains five files: CourseController.java, EnrollController.java, FacultyController.java, StudentController.java (highlighted with a red border), and TutorialController.java.
- Search Bar (Top):** Contains the text "student".
- Code Editor (Center):** Displays the "StudentController.java" file content. A red box highlights the code block starting at line 40, which contains the `@PostMapping("/")` annotation and its associated logic.
- Status Bar (Bottom):** Shows "Ln 50, Col 2" and other status indicators like "Java: Ready".

```
public class StudentController {  
    public String getById(  
        System.out.println("student-id: " + studentId);  
        return "index";  
    }  
  
    @GetMapping("/delete/{student-id}")  
    public String getDeleteById(  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        System.out.println(x:"---- StudentController getDeleteById() ----");  
        System.out.println("student-id: " + studentId);  
        return "index";  
    }  
  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        @RequestParam() Map<String, String> param  
    ) {  
        System.out.println(x:"---- StudentController postInsertAndUpdate() ----");  
        System.out.println("student-id: " + param.get(key:"student-id"));  
        System.out.println("student-name: " + param.get(key:"student-name"));  
        return "index";  
    }  
}
```

API - Course

(1) getAll

method **GET** - /course

(2) getById

method **GET** - /course/{course-id}

(3) insertCourse, updateCourse

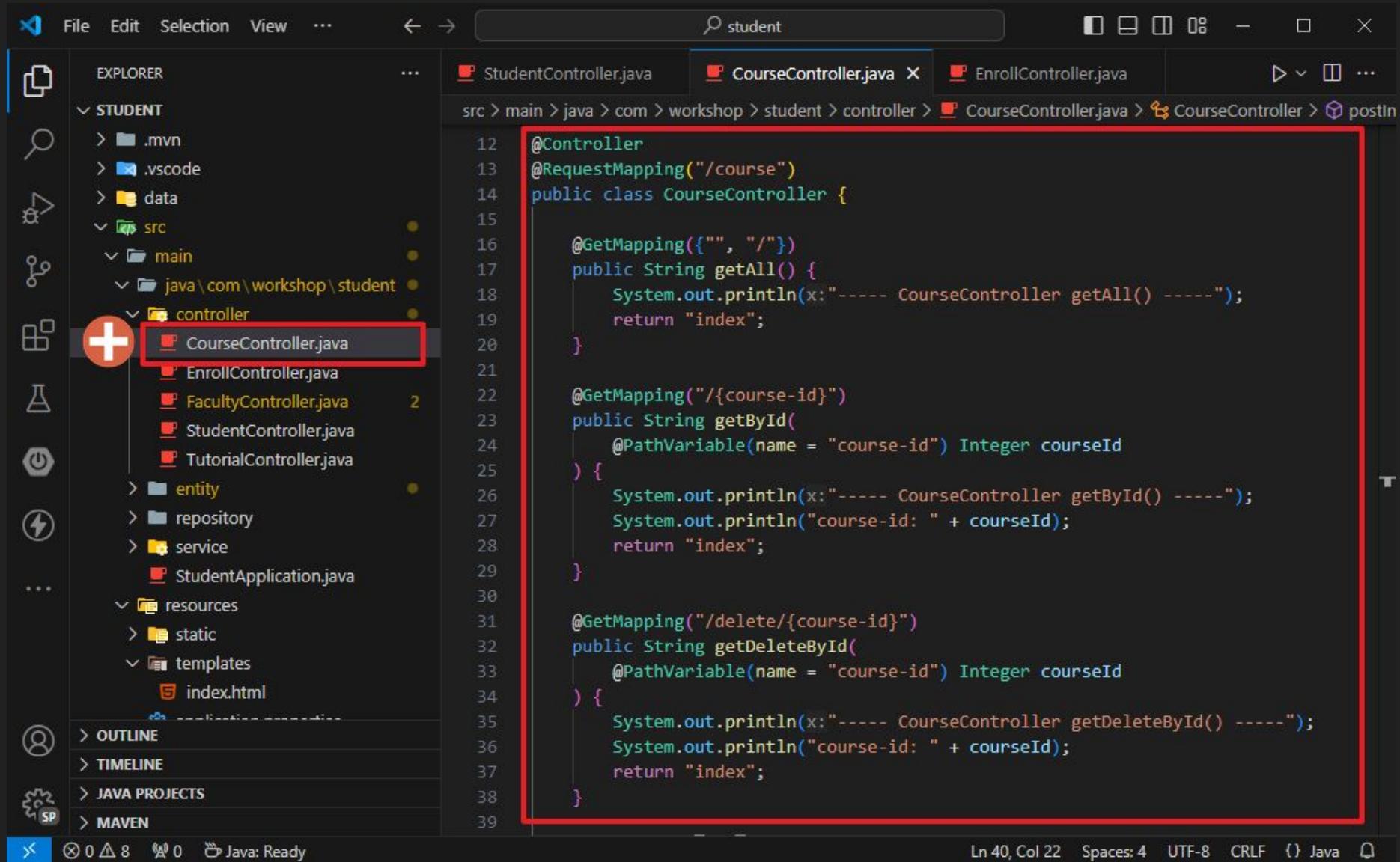
method **POST**

- insert - /course/
- update - /course/{course-id}

(4) deleteCourse

method **GET** - /course/delete/{course-id}

Spring Controller - Create Controller Course



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the 'STUDENT' folder. The 'CourseController.java' file is selected and highlighted with a red border.
- Search Bar:** Contains the text 'student'.
- Editor Area:** Displays the code for 'CourseController.java'. The code defines a controller for managing courses with three methods: getAll(), getById(), and getDeleteById(). Each method prints a message to the console and returns 'index'.
- Bottom Status Bar:** Shows build statistics (0 errors, 0 warnings), Java status ('Java: Ready'), and file information ('Ln 40, Col 22').

```
@Controller  
@RequestMapping("/course")  
public class CourseController {  
  
    @GetMapping({ "", "/" })  
    public String getAll() {  
        System.out.println(x: "---- CourseController getAll() ----");  
        return "index";  
    }  
  
    @GetMapping("/{course-id}")  
    public String getById(  
        @PathVariable(name = "course-id") Integer courseId  
    ) {  
        System.out.println(x: "---- CourseController getById() ----");  
        System.out.println("course-id: " + courseId);  
        return "index";  
    }  
  
    @GetMapping("/delete/{course-id}")  
    public String getDeleteById(  
        @PathVariable(name = "course-id") Integer courseId  
    ) {  
        System.out.println(x: "---- CourseController getDeleteById() ----");  
        System.out.println("course-id: " + courseId);  
        return "index";  
    }  
}
```

Spring Controller - Create Controller Course

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT**: .mvn, .vscode, data, src (selected), main, java\com\workshop\student, controller (selected), CourseController.java (highlighted)
 - entity, repository, service, StudentApplication.java
 - resources, static, templates, index.html
- Center Area:** CourseController.java (highlighted)

```
public class CourseController {  
    public String getById(  
        System.out.println("course-id: " + courseId);  
        return "index";  
    }  
  
    @GetMapping("/delete/{course-id}")  
    public String getDeleteById(  
        @PathVariable(name = "course-id") Integer courseId  
    ) {  
        System.out.println(x:"---- CourseController getDeleteById() ----");  
        System.out.println("course-id: " + courseId);  
        return "index";  
    }  
  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        @RequestParam() Map<String, String> param  
    ) {  
        System.out.println(x:"---- CourseController postInsertAndUpdate() ----");  
        System.out.println("course-id: " + param.get(key:"course-id"));  
        System.out.println("course-name: " + param.get(key:"course-name"));  
        return "index";  
    }  
}
```
- Bottom Status Bar:** Ln 50, Col 2 Spaces: 4 UTF-8 CRLF {} Java

API - Enroll

(1) getAll

method **GET** - /enroll

(2) getById

method **GET** - /enroll/{enroll-id}

(3) insertEnroll, updateEnroll

method **POST**

- insert - /enroll/
- update - /enroll/{enroll-id}

(4) deleteEnroll

method **GET** - /enroll/delete/{enroll-id}

Spring Controller - Create Controller Enroll

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT:** .mvn, .vscode, data, src (selected), main, java\com\workshop\student, controller (selected), CourseController.java, EnrollController.java (highlighted with a red box), FacultyController.java, StudentController.java, TutorialController.java.
 - entity, repository, service, StudentApplication.java.
 - resources, static, templates (selected), index.html.
- Central Area:** Code editor showing EnrollController.java. The code defines a Spring controller for handling enrollment requests. It includes methods for getting all enrollments, getting an enrollment by ID, and deleting an enrollment by ID. A red box highlights the entire code block.
- Bottom Status Bar:** Line 40, Column 22, Spaces: 4, UTF-8, CRLF, Java.

```
@Controller
@RequestMapping("/enroll")
public class EnrollController {

    @GetMapping("", "/")
    public String getAll() {
        System.out.println("----- EnrollController getAll() -----");
        return "index";
    }

    @GetMapping("/{enroll-id}")
    public String getById(
        @PathVariable(name = "enroll-id") Integer enrollId
    ) {
        System.out.println("----- EnrollController getById() -----");
        System.out.println("enroll-id: " + enrollId);
        return "index";
    }

    @GetMapping("/delete/{enroll-id}")
    public String getDeleteById(
        @PathVariable(name = "enroll-id") Integer enrollId
    ) {
        System.out.println("----- EnrollController getDeleteById() -----");
        System.out.println("enroll-id: " + enrollId);
        return "index";
    }
}
```

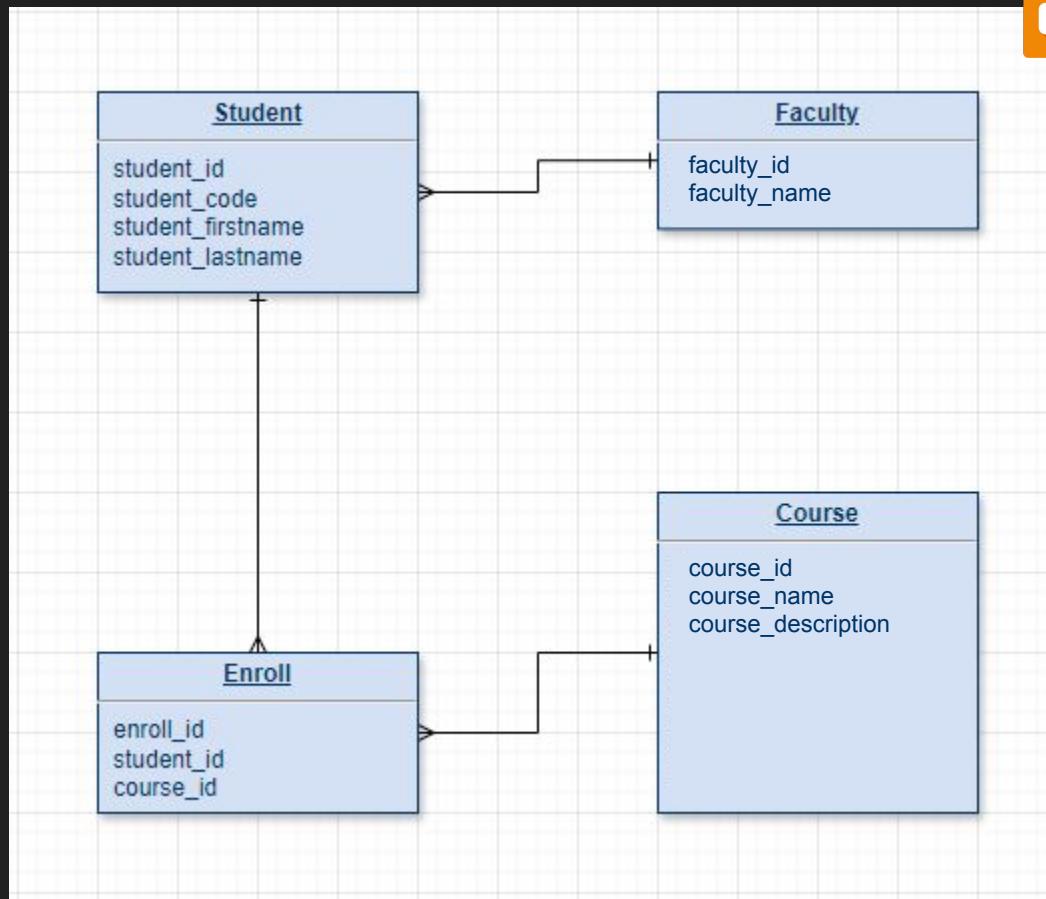
Spring Controller - Create Controller Enroll

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Explorer:** Shows the project structure under STUDENT:
 - .mvn
 - .vscode
 - data
 - src
 - main
 - java\com\workshop\student
 - controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - index.html
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Editor:** The file EnrollController.java is open, showing Java code for a REST controller. A red box highlights the `@PostMapping("/")` annotated method. Another red box highlights the entire code block from line 40 to line 51.
- Bottom Status Bar:** Ln 50, Col 2 Spaces: 4 UTF-8 CRLF {} Java

Database

ER Diagram (Entity Relationship)



Use H2 Database

The screenshot shows the 'Add Dependencies' dialog in IntelliJ IDEA. The 'SQL' tab is selected. A green box highlights the 'H2 Database' entry, which is described as providing fast in-memory database access with a small footprint and supporting embedded and server modes. Other listed databases include IBM DB2 Driver, Apache Derby Database, HyperSQL Database, MariaDB Driver, MS SQL Server Driver, and MySQL Driver.

initializr

Web, Security, JPA, Actuator, Devtools...

Press Ctrl for multiple adds

SQL

IBM DB2 Driver
A JDBC driver that provides access to IBM DB2.

Apache Derby Database
An open source relational database implemented entirely in Java.

H2 Database
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. ↵
Supports embedded and server modes as well as a browser based console application.

HyperSQL Database
Lightweight 100% Java SQL Database Engine.

MariaDB Driver
MariaDB JDBC and R2DBC driver.

MS SQL Server Driver
A JDBC and R2DBC driver that provides access to Microsoft SQL Server and Azure SQL Database from any Java application.

MySQL Driver
MySQL JDBC driver.

3.4.0 (M1)

3.2.8

ple

ect for Spring Boot

ple.demo

○ War

ADD D

Use Spring Data JPA

The screenshot shows the Initializr web interface for creating a Spring Boot project. The URL in the address bar is <https://start.spring.io/>. The page displays various dependency categories and their descriptions. A red box highlights the **Spring Data JPA** section, which is described as persisting data in SQL stores using Java Persistence API and Hibernate. Other visible sections include **SECURITY** (Okta), **SQL** (JDBC API), **Spring Data JDBC**, **Spring Data R2DBC**, and **MyBatis Framework**.

Initializr

Web, Security, JPA, Actuator, Devtools...

Press Ctrl for multiple adds

SECURITY

Okta
Okta specific configuration for Spring Security/Spring Boot OAuth2 features. Enable your Spring Boot application to work with Okta via OAuth 2.0/OIDC.

SQL

JDBC API
Database Connectivity API that defines how a client may connect and query a database.

Spring Data JPA
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Data JDBC
Persist data in SQL stores with plain JDBC using Spring Data.

Spring Data R2DBC
Provides Reactive Relational Database Connectivity to persist data in SQL stores using Spring Data in reactive applications.

MyBatis Framework
Persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis couples objects with stored procedures or SQL statements using a XML descriptor or annotations.

3.4.0 (M1) 3.2.8

ADD DI

Dependency in pom.xml

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "STUDENT". Key files include .mvn, .vscode, pom.xml, StudentApplication.java, application.properties, and several configuration files like mvnw and mvnw.cmd.
- Search Bar (Top):** Contains the text "student".
- Code Editor (Right):** Displays the content of the pom.xml file. A red box highlights the following section of code, which defines dependencies for H2 database, Spring Boot Starter Data JPA, and Spring Boot Starter Test:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```
- Bottom Status Bar:** Shows "Java: Ready" and other status indicators.

Add config at application.properties

The screenshot shows the VS Code interface with the file `application.properties` open. The code defines properties for a Spring Boot application named "student". It includes configurations for H2 Console and a Datasource.

```
spring.application.name=student

# ----- H2 Console -----
# http://localhost:8080/h2-console/
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.h2.console.settings.trace=false
spring.h2.console.settings.web-allow-others=false

# ----- Datasource -----
# spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.url=jdbc:h2:file:./data/student
spring.datasource.username=sa
spring.datasource.password=password

spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.hibernate.ddl-auto=update
```

Annotations highlight specific sections of the code:

- A red box surrounds the H2 Console configuration (lines 3-8).
- A red box surrounds the Datasource configuration (lines 10-21).
- A blue arrow points from the "Datasource" section to a blue box containing three actions:
 - create-drop
 - update
 - validate

VS Code status bar at the bottom: Ln 21, Col 37 Spaces: 4 UTF-8 LF Spring Boot Properties

Restart project for update properties data

Database file is generated

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "STUDENT". A red box highlights the "data" folder, which contains the "student.mv.db" database file. A green checkmark icon is next to "student.mv.db".
- Code Editor (Center):** Displays the "application.properties" file. A red box highlights the line: `spring.datasource.url=jdbc:h2:file:./data/student`.

```
spring.application.name=student
# ----- H2 Console -----
# http://localhost:8080/h2-console/
spring.h2.console.enabled=true
spring.h2.console.settings.trace=false
spring.h2.console.settings.web-allow-others=false
# ----- Datasource -----
# spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.url=jdbc:h2:file:./data/student
spring.datasource.username=sa
spring.datasource.password=password
```
- Terminal (Bottom):** Shows the logs from the application restart:

```
2024-08-20T16:38:37.258+07:00  WARN 7872 --- [student] [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-08-20T16:38:37.308+07:00  INFO 7872 --- [student] [ restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index
2024-08-20T16:38:38.924+07:00  INFO 7872 --- [student] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-08-20T16:38:39.010+07:00  INFO 7872 --- [student] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-08-20T16:38:39.023+07:00  INFO 7872 --- [student] [ restartedMain] c.workshop.student.StudentApplication : Started StudentApplication in 13.01 seconds (process running for 14.299)
```

Open H2 Console

The screenshot shows a browser window titled "H2 Console" with the URL "localhost:8080/h2-console/" highlighted by a red box and circled with a red number 1. The main content area displays the "localhost:8080/h2-console/" title in red. On the left, there is a "Login" form with the following fields:

- Saved Settings: Generic H2 (Embedded)
- Setting Name: Generic H2 (Embedded) (with Save and Remove buttons)
- Driver Class: org.h2.Driver
- JDBC URL: jdbc:h2:file:./data/student
- User Name: sa
- Password: (redacted)
- Buttons: Connect (highlighted with a red box and circled with a red number 3) and Connection

A red arrow points from the "Connection" text in the code block below to the "Connection" button in the form.

On the right, the configuration code is displayed:

```
# ----- Datasource -----
# spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.url=jdbc:h2:file:./data/student
spring.datasource.username=sa
spring.datasource.password=password

spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Login to H2 Console

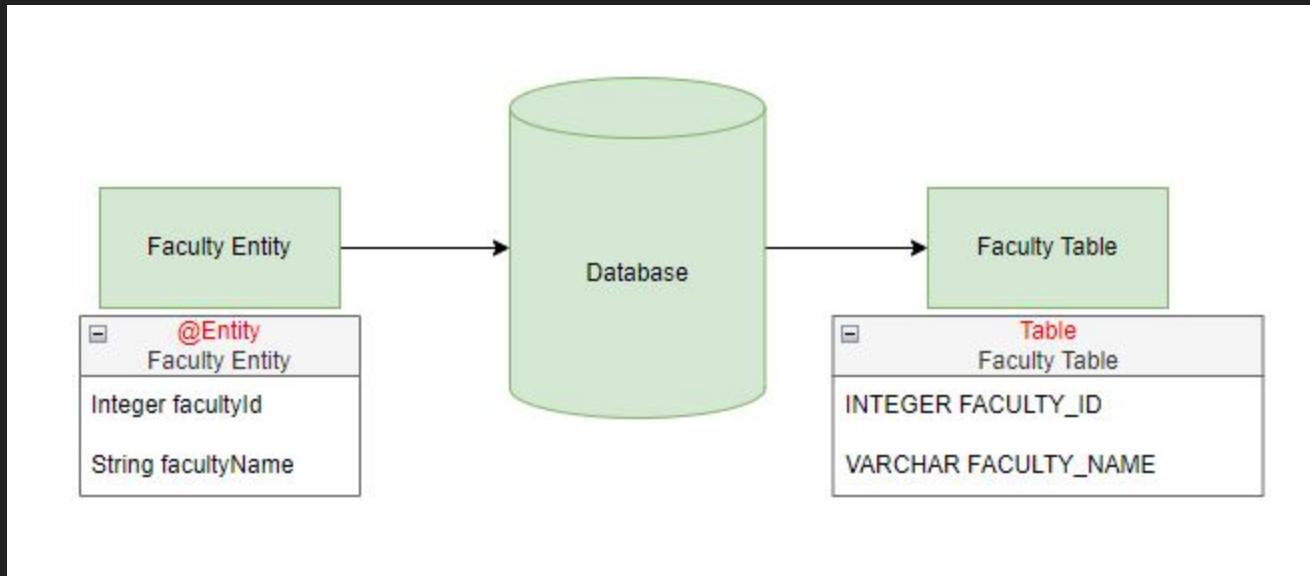
The screenshot shows a web browser window titled "H2 Console" at the URL "localhost:8080/h2-console/login.do?jsessionid=74ffad18233448916e6adf800...". The browser interface includes standard navigation buttons (back, forward, search, etc.) and a toolbar with various icons. On the left, there is a sidebar with database connection details ("jdbc:h2:file:/data/student"), schema information ("INFORMATION_SCHEMA", "Users"), and a version note ("H2 2.2.224 (2023-09-17)"). The main area contains a toolbar with buttons for "Run", "Run Selected", "Auto complete", and "Clear", followed by a text input field labeled "SQL statement:". Below this is a large empty workspace. At the bottom, there are two sections: "Important Commands" (with a table of keyboard shortcuts) and "Sample SQL Script" (with a table showing a script to create a table and insert data).

Icon	Command	Description
?		Displays this Help Page
↶		Shows the Command History
▶	Ctrl+Enter	Executes the current SQL statement
▶	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
🔌		Disconnects from the database

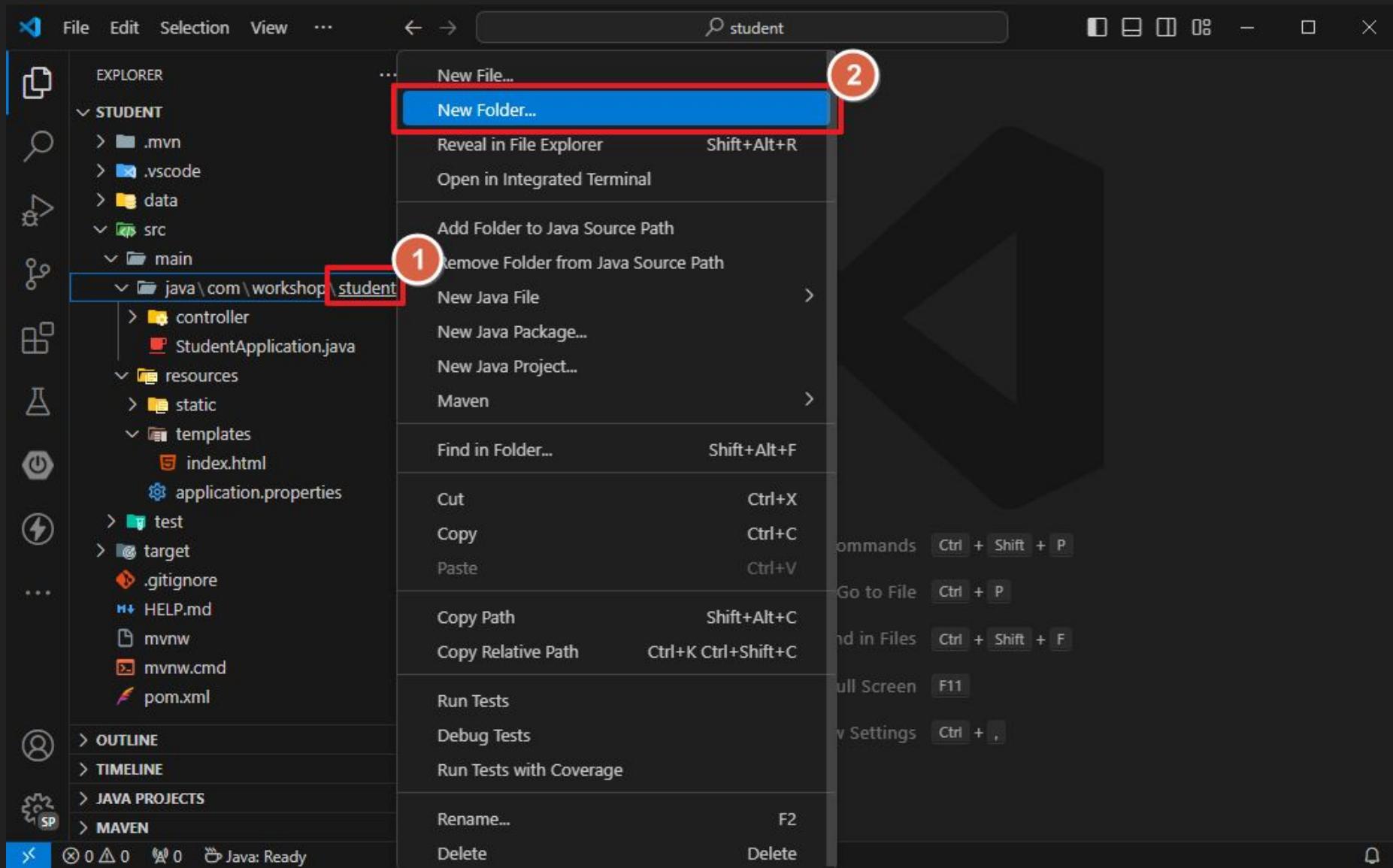
Action	SQL Statement
Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');

Entity, Repository And Service

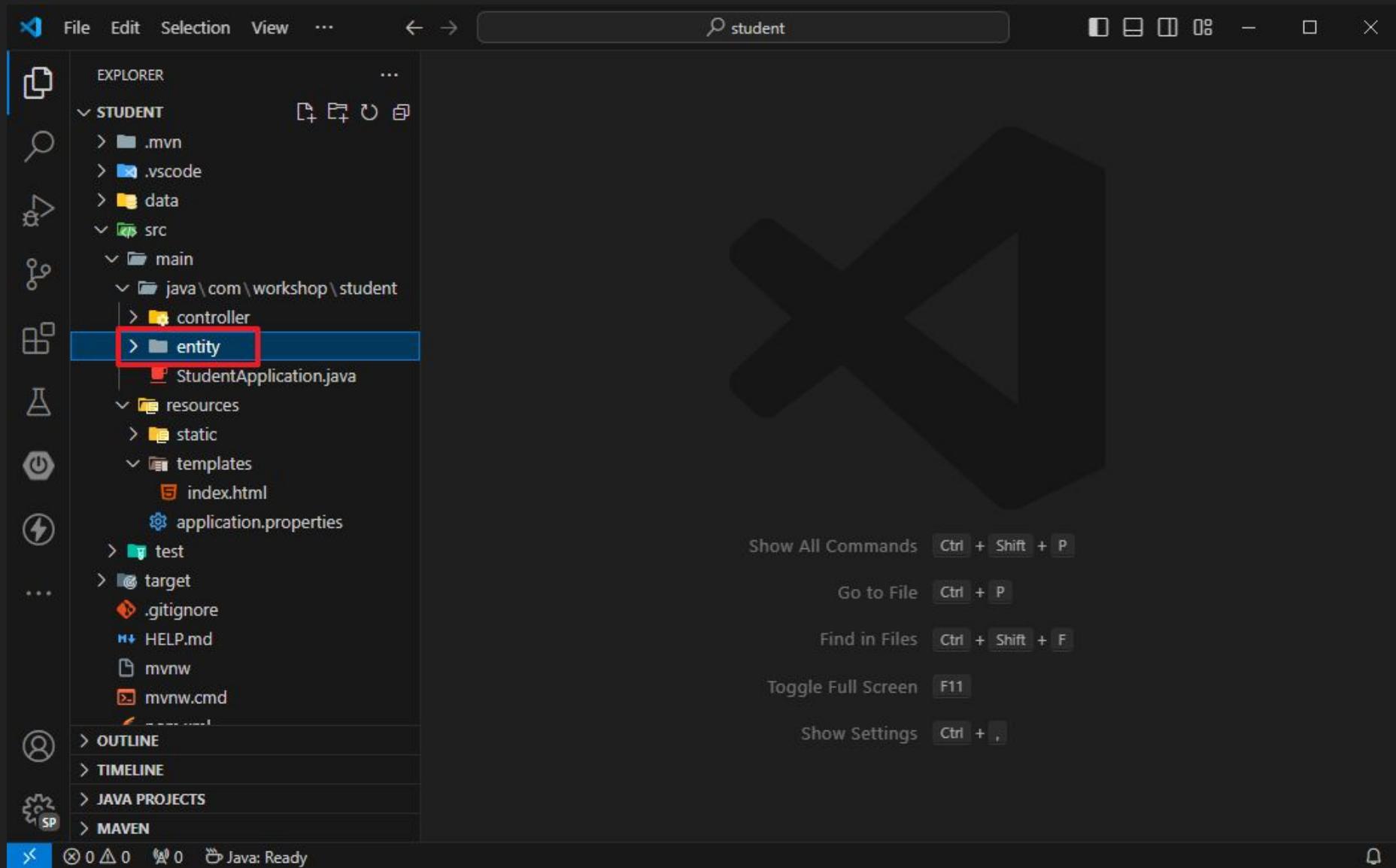
Entity, Repository and Service



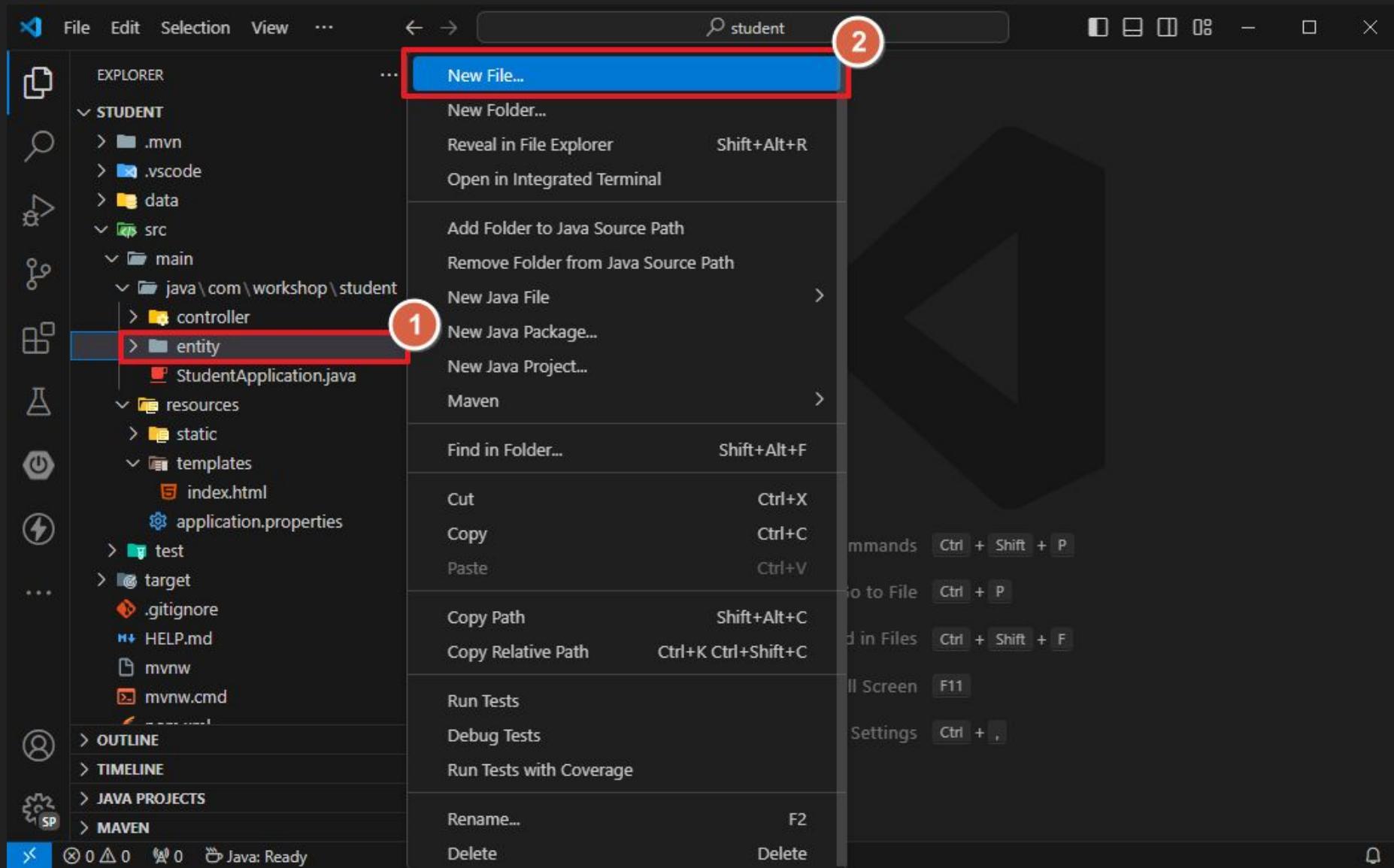
Create folder “*Entity*” and create FacultyEntity.java



Create folder “*Entity*” and create FacultyEntity.java



Create folder “*Entity*” and create FacultyEntity.java



Create folder *Entity* and create FacultyEntity.java

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Explorer:** Shows the project structure under STUDENT:
 - .mvn
 - .vscode
 - data
 - src
 - main
 - java\com\workshop\student
 - controller
 - entity
 - FacultyEntity.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - index.html
 - application.properties
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Editor:** FacultyEntity.java (highlighted)
- Status Bar:** Ln 5, Col 2 Spaces: 4 UTF-8 CRLF () Java

Create folder “*Entity*” and create FacultyEntity.java

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder. The "entity" folder contains "FacultyEntity.java" and "StudentApplication.java".
- Code Editor (Center):** Displays the content of "FacultyEntity.java". A red box highlights the annotations and class definition.

```
src > main > java > com > workshop > student > entity > FacultyEntity.java > ...
1 import lombok.AllArgsConstructor;
2 import lombok.Builder;
3 import lombok.Getter;
4 import lombok.NoArgsConstructor;
5 import lombok.Setter;
6 import lombok.ToString;
7
8 @Getter
9 @Setter
10 @AllArgsConstructor
11 @NoArgsConstructor
12 @Builder
13
14 @Entity
15 @Table(name = "faculty")
16 public class FacultyEntity {
17
18     @Id
19     @GeneratedValue
20     private Integer facultyId;
21     private String facultyName;
22 }
```
- Bottom Status Bar:** Shows "Java: Ready" and other status indicators.
- Bottom Right:** Includes file navigation icons and status text like "Ln 32, Col 1" and "Spaces: 4".

Check result on H2 Console

The screenshot shows the H2 Console interface running in a browser window. The URL is `localhost:8080/h2-console/login.do?jsessionid=a5b21ba0ef218f109bb48aa27...`. The left sidebar shows the database structure:

- jdbc:h2:file:/data/student** (selected)
- FACULTY** (expanded):
 - FACULTY_ID**
 - FACULTY_NAME**
- Indexes**
- INFORMATION_SCHEMA**
- Sequences**
- Users**

A red box highlights the **FACULTY** table. A green checkmark icon is positioned next to the table name in the sidebar.

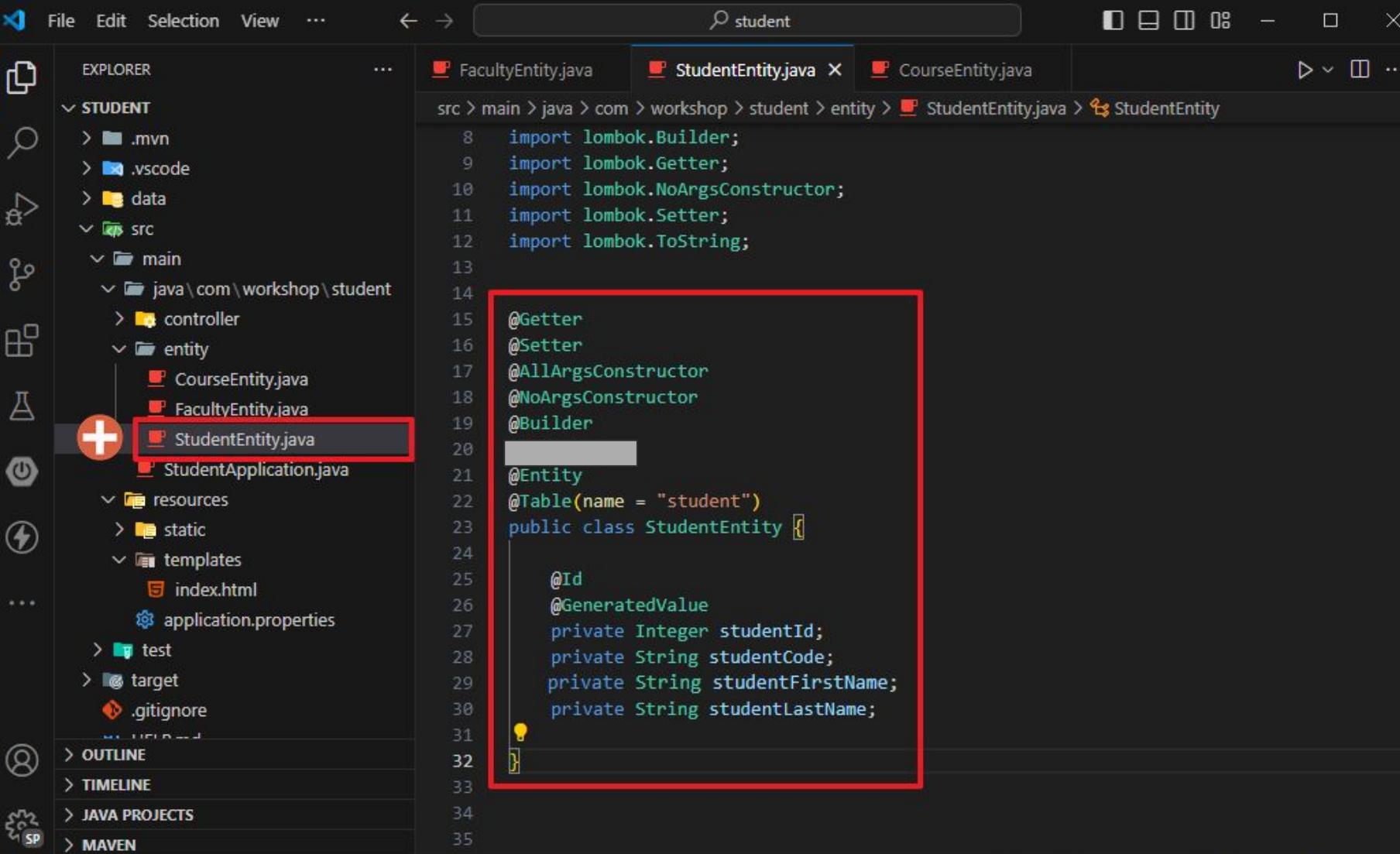
Important Commands

	Displays this Help Page
	Shows the Command History
	Ctrl+Enter Executes the current SQL statement
	Shift+Enter Executes the SQL statement defined by the text selection
	Ctrl+Space Auto complete
	Disconnects from the database

Sample SQL Script

Delete the table if it exists	<code>DROP TABLE IF EXISTS TEST;</code>
Create a new table with ID and NAME columns	<code>CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));</code>
Add a new row	<code>INSERT INTO TEST VALUES(1, 'Hello');</code>
Add another row	<code>INSERT INTO TEST VALUES(2, 'World');</code>

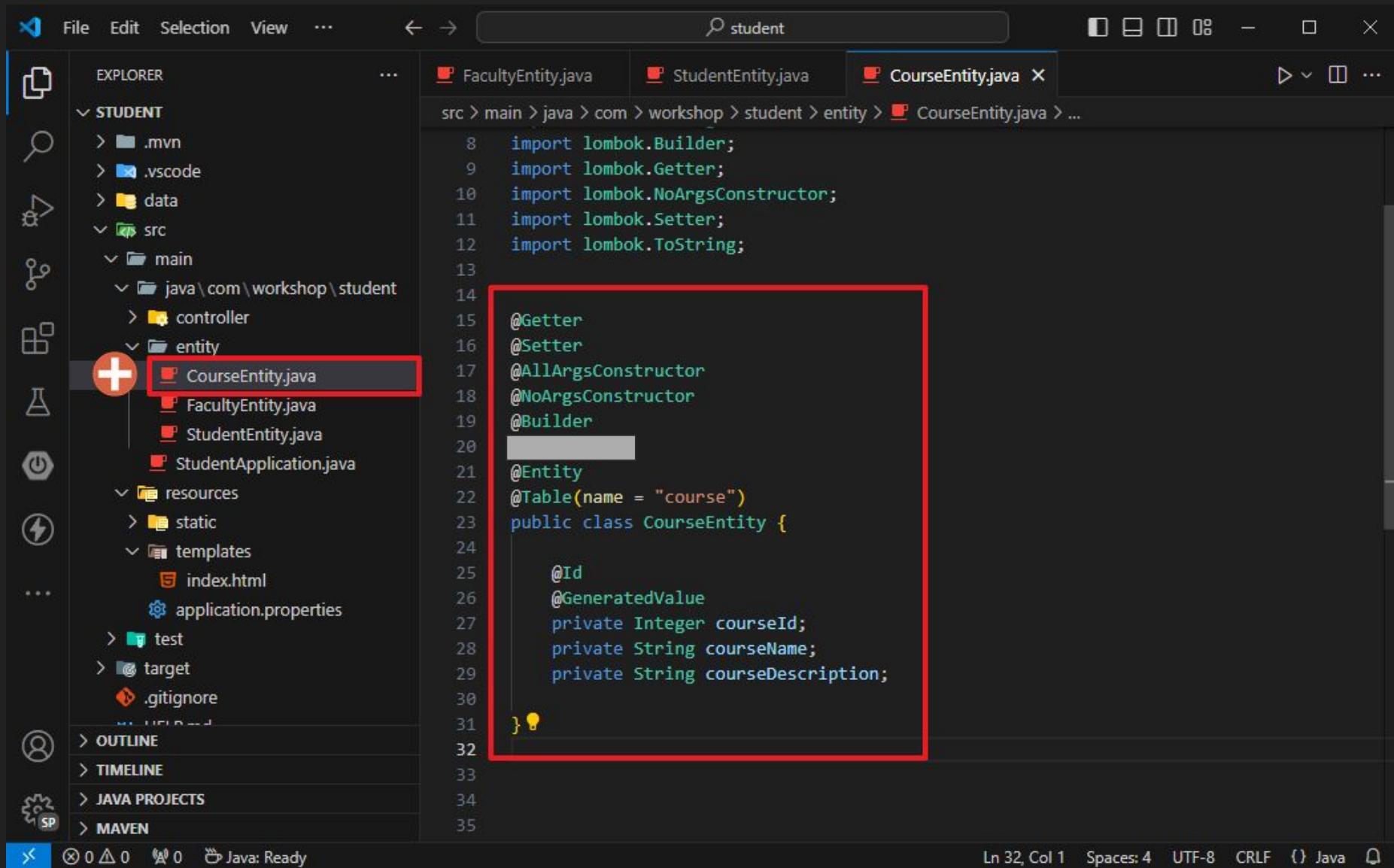
Create StudentEntity.java



The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the **STUDENT** folder. The **StudentEntity.java** file is selected and highlighted with a red box.
- Editor (Center):** Displays the **StudentEntity.java** code. The code includes imports for `lombok.Builder`, `lombok.Getter`, `lombok.NoArgsConstructor`, `lombok.Setter`, and `lombok.ToString`. It also features Lombok annotations (`@Getter`, `@Setter`, `@AllArgsConstructor`, `@NoArgsConstructor`, `@Builder`) and JPA annotations (`@Entity`, `@Table(name = "student")`). The class definition starts with `public class StudentEntity {` and ends with `}`.
- Search Bar (Top):** Contains the text `student`.
- Toolbar (Top):** Includes standard icons for file operations like Open, Save, and Close, as well as a search icon.
- Bottom Status Bar:** Shows the current file is `StudentEntity.java`, line 32, column 2, with 4 spaces, and the encoding is UTF-8 CRLF. It also indicates Java is ready.

Create CourseEntity.java



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a project structure under "STUDENT". The "entity" folder contains "CourseEntity.java", which is selected and highlighted with a red box. Other files in the "entity" folder include "FacultyEntity.java", "StudentEntity.java", and "StudentApplication.java".
- Editor (Center):** Displays the code for "CourseEntity.java". The code uses Lombok annotations and includes fields for courseId, courseName, and courseDescription.
- Search Bar:** Contains the text "student".
- Bottom Status Bar:** Shows "Java: Ready" and other status indicators like "Ln 32, Col 1" and "Spaces: 4".

```
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "course")
public class CourseEntity {

    @Id
    @GeneratedValue
    private Integer courseId;
    private String courseName;
    private String courseDescription;
}
```

Check result on H2 Console

The screenshot shows the H2 Console interface running in a browser window. The title bar says "H2 Console". The left sidebar displays the database schema:

- COURSE**: Contains columns COURSE_ID, COURSE_DESCRIPTION, COURSE_NAME, and Indexes.
- FACULTY**
- STUDENT**: Contains columns STUDENT_ID, STUDENT_CODE, STUDENT_FIRST_NAME, STUDENT_LAST_NAME, and Indexes.
- INFORMATION_SCHEMA**
- Sequences**
- Users**

Two sections of the schema are highlighted with red boxes: **COURSE** and **STUDENT**. The bottom left of the sidebar shows the version: H2 2.2.224 (2023-09-17).

The main area contains the following sections:

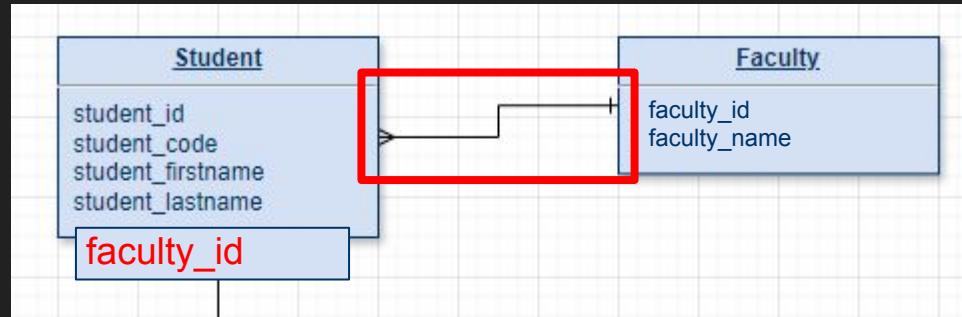
- Important Commands** table:

	Displays this Help Page
	Shows the Command History
	Ctrl+Enter Executes the current SQL statement
	Shift+Enter Executes the SQL statement defined by the text selection
	Ctrl+Space Auto complete
	Disconnects from the database
- Sample SQL Script** table:

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');

Entity Relationship (@ManyToOne)

@ManyToOne > M Student To 1 Faculty



StudentEntity.java

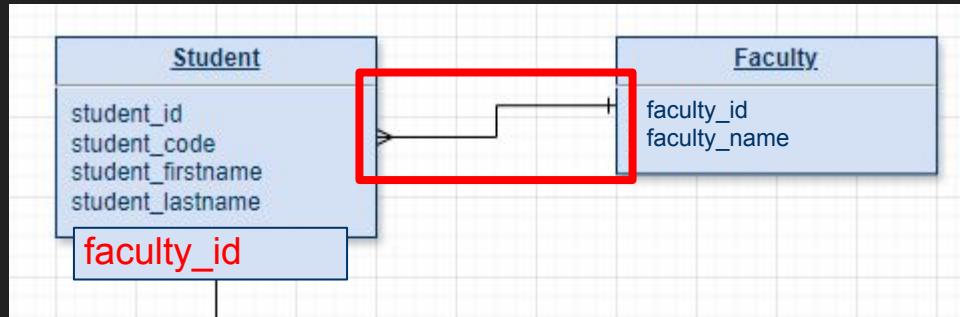
```
src > main > java > com > workshop > student > entity > StudentEntity.java > StudentEntity > faculty
```

```
25  public class StudentEntity {  
26  
27      @Id  
28      @GeneratedValue  
29      private Integer studentId;  
30      private String studentCode;  
31      private String studentFirstName;  
32      private String studentLastName;  
33  
34      @ManyToOne  
35      @JoinColumn(name = "faculty_id")  
36      private FacultyEntity faculty;  
37  
38  }
```

Ln 34, Col 42 Spaces: 4 UTF-8 CRLF {} Java

Entity Relationship (@OneToMany)

@OneToMany > 1 Faculty To M Student



The screenshot shows the `FacultyEntity.java` file in an IDE. The code defines a class `FacultyEntity` with attributes `facultyId` and `facultyName`. It includes a `@OneToMany` annotation with the `mappedBy` attribute set to "faculty", which maps to the `students` field, represented by a red box. The code is as follows:

```
26 public class FacultyEntity {  
27  
28     @Id  
29     @GeneratedValue  
30     private Integer facultyId;  
31     private String facultyName;  
32  
33     @OneToMany(mappedBy = "faculty")  
34     private List<StudentEntity> students;  
35  
36 }
```

1st @ManyToOne - 2nd @OneToMany

The screenshot shows a Java application structure in VS Code. The **EXPLORER** sidebar lists the project structure under the **STUDENT** folder. The **FacultyEntity.java** file is open in the editor, and the **StudentEntity.java** file is shown in a preview pane below it.

FacultyEntity.java:

```
public class FacultyEntity {  
    @Id  
    @GeneratedValue  
    private Integer facultyId;  
    private String facultyName;  
    @OneToMany(mappedBy = "faculty")  
    private List<StudentEntity> students;  
}
```

StudentEntity.java:

```
public class StudentEntity {  
    @Id  
    @GeneratedValue  
    private Integer studentId;  
    private String studentCode;  
    private String studentFirstName;  
    private String studentLastName;  
    @ManyToOne  
    @JoinColumn(name = "faculty_id")  
    private FacultyEntity faculty;  
}
```

A red circle with the number **1** highlights the `@ManyToOne` annotation and its associated `JoinColumn` annotation in the `StudentEntity` class. A red circle with the number **2** highlights the `@OneToMany` annotation and its associated `mappedBy` attribute in the `FacultyEntity` class. An orange arrow points from the `mappedBy` attribute in the `FacultyEntity` class to the `faculty` field in the `StudentEntity` class.

@JoinColumn (custom name)

The screenshot shows the VS Code interface with two open files: `FacultyEntity.java` and `StudentEntity.java`.

FacultyEntity.java:

```
public class FacultyEntity {  
    @Id  
    @GeneratedValue  
    private Integer facultyId;  
    private String facultyName;  
  
    @OneToMany(mappedBy = "faculty")  
    private List<StudentEntity> students;  
}
```

StudentEntity.java:

```
public class StudentEntity {  
    @Id  
    @GeneratedValue  
    private Integer studentId;  
    private String studentCode;  
    private String studentFirstName;  
    private String studentLastName;  
  
    @ManyToOne  
    @JoinColumn(name = "faculty_id")  
    private FacultyEntity faculty;  
}
```

The `@JoinColumn(name = "faculty_id")` annotation in the `StudentEntity` class is highlighted with a red rectangle.

VS Code status bar at the bottom: Ln 34, Col 42 Spaces: 4 UTF-8 CRLF {} Java

@JoinColumn (custom name)

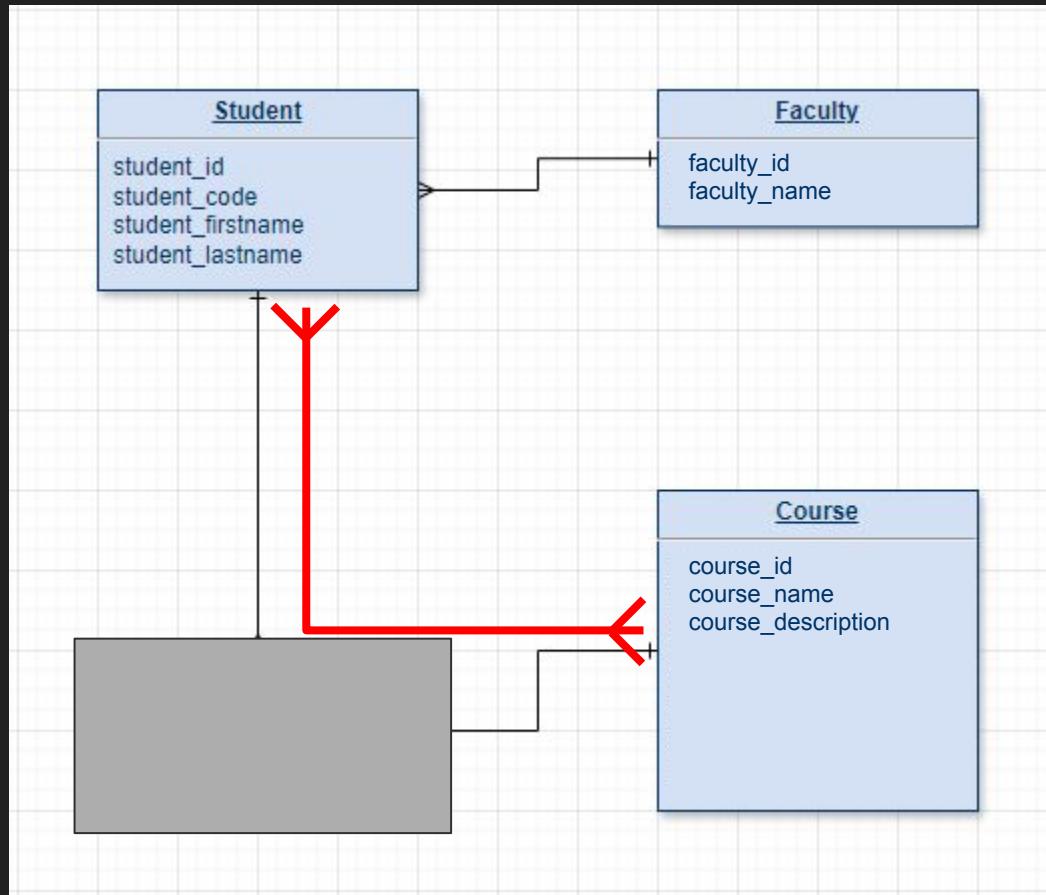
The screenshot shows the H2 Console interface with the following details:

- Database Connection:** jdbc:h2:file:/data/student
- Tables:** COURSE, FACULTY, STUDENT, INFORMATION_SCHEMA, Sequences, Users.
- STUDENT Table:** Faculty_ID, STUDENT_ID, STUDENT_CODE, STUDENT_FIRST_NAME, STUDENT_LAST_NAME.
- Faculty_ID Column:** This column is highlighted with a red box.
- Important Commands:** A table listing keyboard shortcuts and their functions.
- Sample SQL Script:** A table showing how to delete a table, create a new table, add rows, and insert values.

Command	Description
?	Displays this Help Page
!	Shows the Command History
Ctrl+Enter	Executes the current SQL statement
Shift+Enter	Executes the SQL statement defined by the text selection
Ctrl+Space	Auto complete
Disconnect icon	Disconnects from the database

Action	SQL Statement
Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');

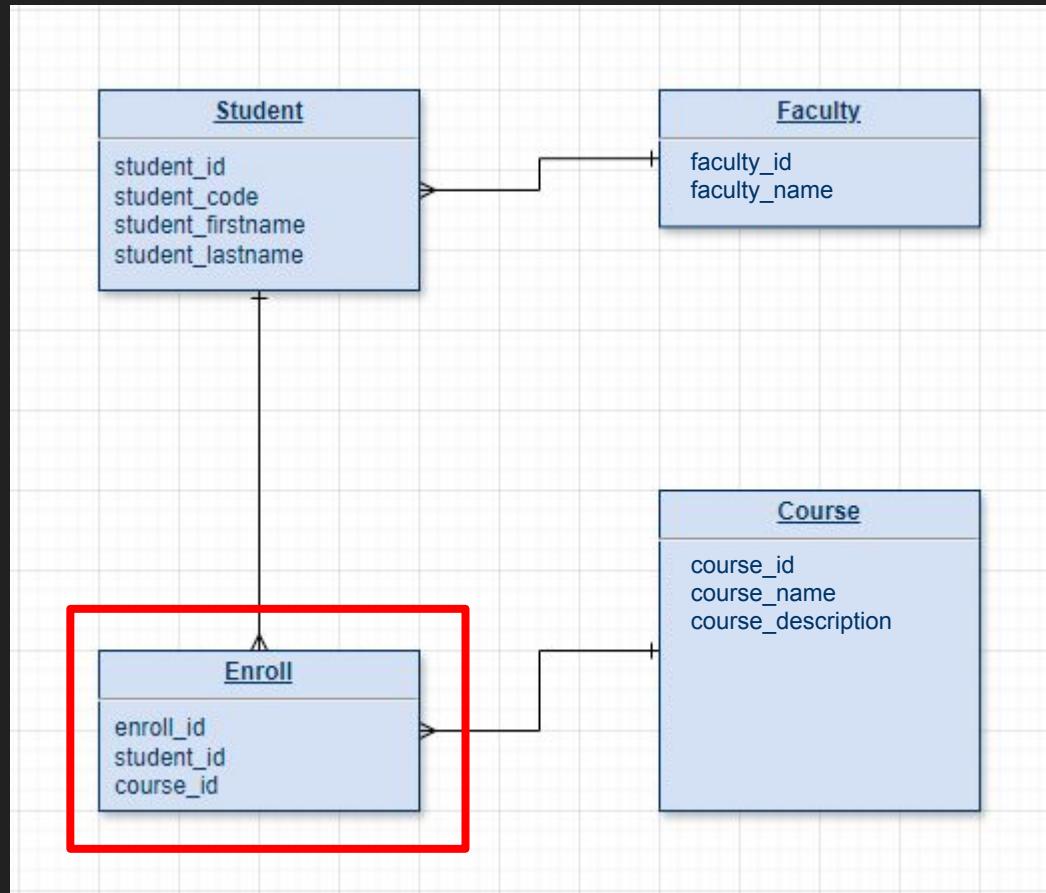
Entity Relationship (@ManyToMany)



Entity Relationship (@ManyToOne)

@ManyToOne > M Enroll To 1 Student

@ManyToOne > M Enroll To 1 Course



Entity Relationship (@ManyToOne)

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under 'STUDENT'. The 'src' folder contains 'main', 'java', 'entity', and 'resources'. 'entity' contains 'CourseEntity.java', 'EnrollEntity.java' (highlighted with a red border), 'FacultyEntity.java', and 'StudentEntity.java'. 'resources' contains 'static' and 'templates'.
- Code Editor (Right):** Displays the 'EnrollEntity.java' code. The code defines a class 'EnrollEntity' with annotations for Lombok and JPA. It includes fields for 'enrollId' (auto-generated integer) and two many-to-one relationships: 'student' (with student_id as join column) and 'course' (with course_id as join column). A red box highlights the entire code block in the editor.
- Bottom Status Bar:** Shows build errors (8), Java status ('Java: Ready'), and other system information like line and column numbers (Ln 40, Col 1).

```
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@NoArgsConstructor
@Builder
@Entity
@Table(name = "enroll")
public class EnrollEntity {

    @Id
    @GeneratedValue
    private Integer enrollId;

    @ManyToOne
    @JoinColumn(name = "student_id")
    private StudentEntity student;

    @ManyToOne
    @JoinColumn(name = "course_id")
    private CourseEntity course;
}
```

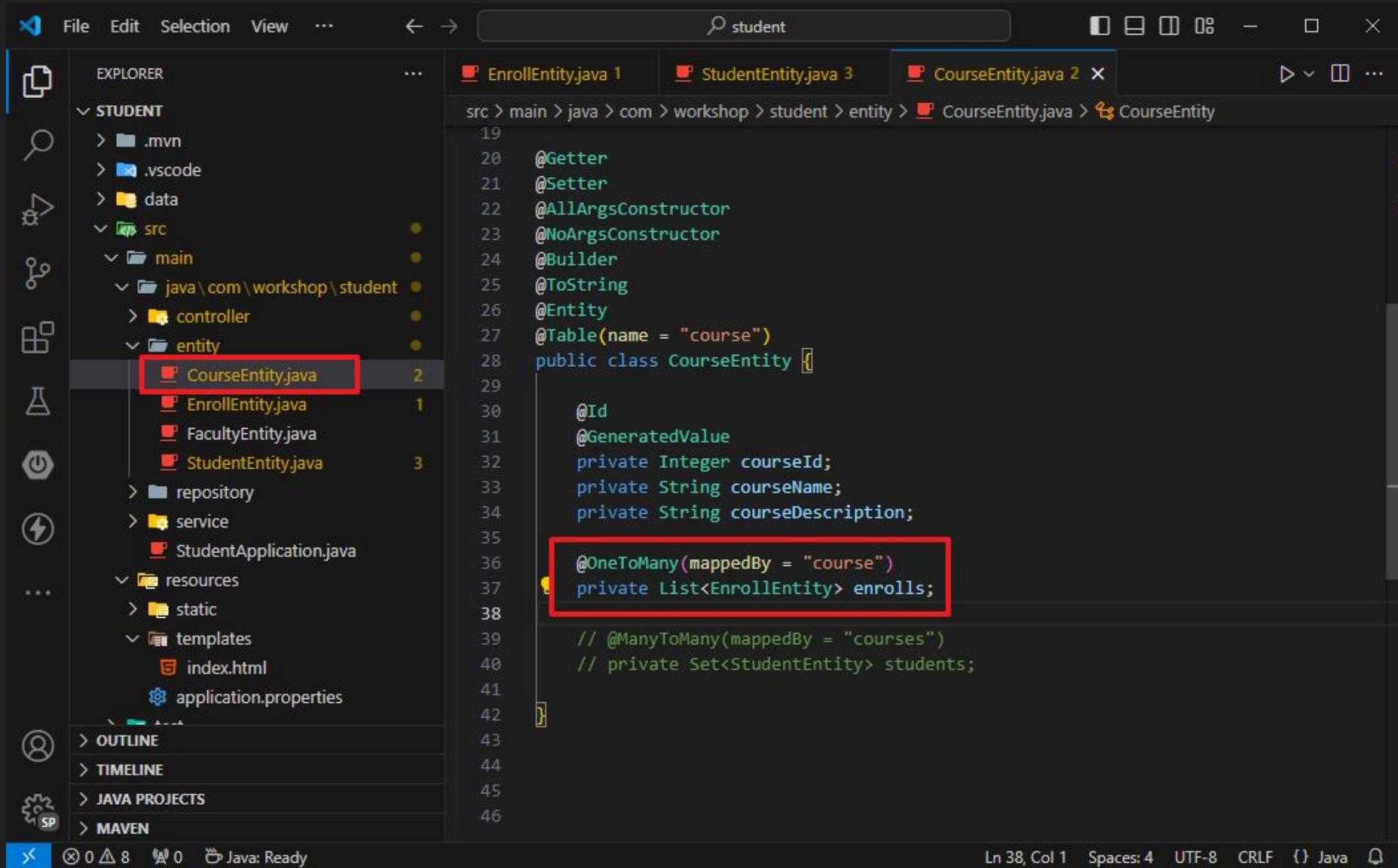
Entity Relationship (@ManyToOne)

The screenshot shows a Java code editor in VS Code displaying the `StudentEntity.java` file. The code defines a student entity with attributes for student ID, code, first name, last name, and a many-to-one relationship to a faculty entity. A specific line of code for the many-to-one relationship is highlighted with a red box.

```
22  @Getter  
23  @Setter  
24  @AllArgsConstructor  
25  @NoArgsConstructor  
26  @Builder  
27  @ToString  
28  @Entity  
29  @Table(name = "student")  
30  public class StudentEntity {  
31  
32      @Id  
33      @GeneratedValue  
34      private Integer studentId;  
35      private String studentCode;  
36      private String studentFirstName;  
37      private String studentLastName;  
38  
39      @ManyToOne  
40      @JoinColumn(name = "faculty_id")  
41      private FacultyEntity faculty;  
42  
43      @OneToMany(mappedBy = "student")  
44      private List<EnrollEntity> enrolls;  
45  
46      // @ManyToMany  
47      // @JoinTable(  
48      //     name = "enroll",  
49
```

The code editor interface includes a top bar with file navigation, a search bar containing "student", and a status bar at the bottom indicating "Java: Ready". The left sidebar shows the project structure under the "STUDENT" folder, with the `StudentEntity.java` file selected and highlighted with a red border.

Entity Relationship (@ManyToOne)

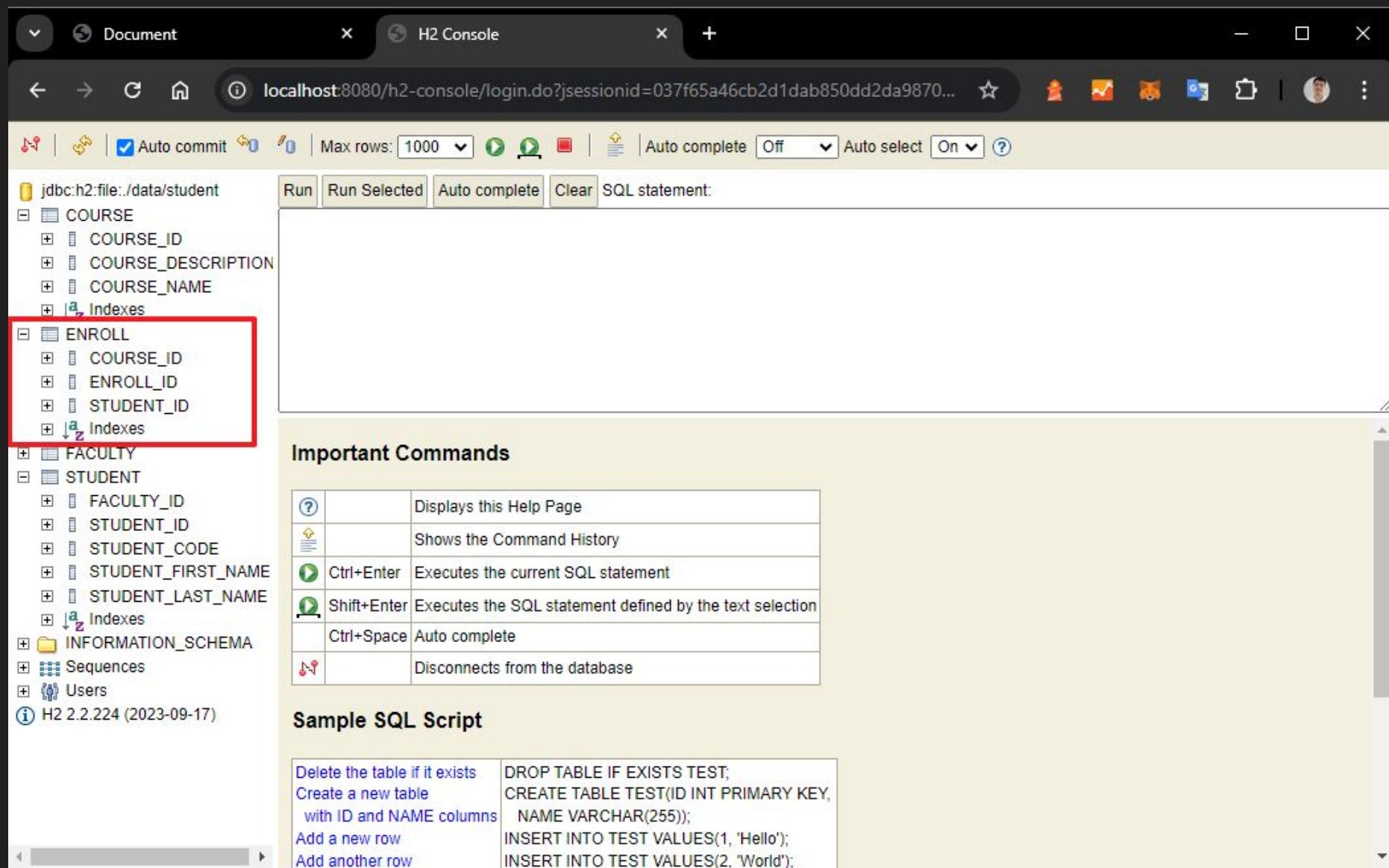


The screenshot shows a Java code editor in VS Code with the following details:

- File Explorer:** Shows a project structure under the **STUDENT** folder. The **entity** package contains **CourseEntity.java**, **EnrollEntity.java**, **FacultyEntity.java**, and **StudentEntity.java**. The **CourseEntity.java** file is currently open.
- Search Bar:** Contains the text "student".
- Code Editor:** Displays the **CourseEntity.java** code. A red box highlights the **@ManyToOne** annotation on line 37.

```
19  
20     @Getter  
21     @Setter  
22     @AllArgsConstructor  
23     @NoArgsConstructor  
24     @Builder  
25     @ToString  
26     @Entity  
27     @Table(name = "course")  
28     public class CourseEntity {  
  
29         @Id  
30         @GeneratedValue  
31         private Integer courseId;  
32         private String courseName;  
33         private String courseDescription;  
  
34         @OneToMany(mappedBy = "course")  
35         private List<EnrollEntity> enrolls;  
  
36         // @ManyToMany(mappedBy = "courses")  
37         // private Set<StudentEntity> students;  
38     }  
39  
40  
41  
42  
43  
44  
45  
46
```
- Bottom Status Bar:** Shows build errors: 8 errors and 0 warnings. It also indicates "Java: Ready".
- Bottom Right:** Includes status information: Line 38, Column 1, Spaces: 4, UTF-8, CRLF, and Java.

Entity Relationship (@ManyToOne)



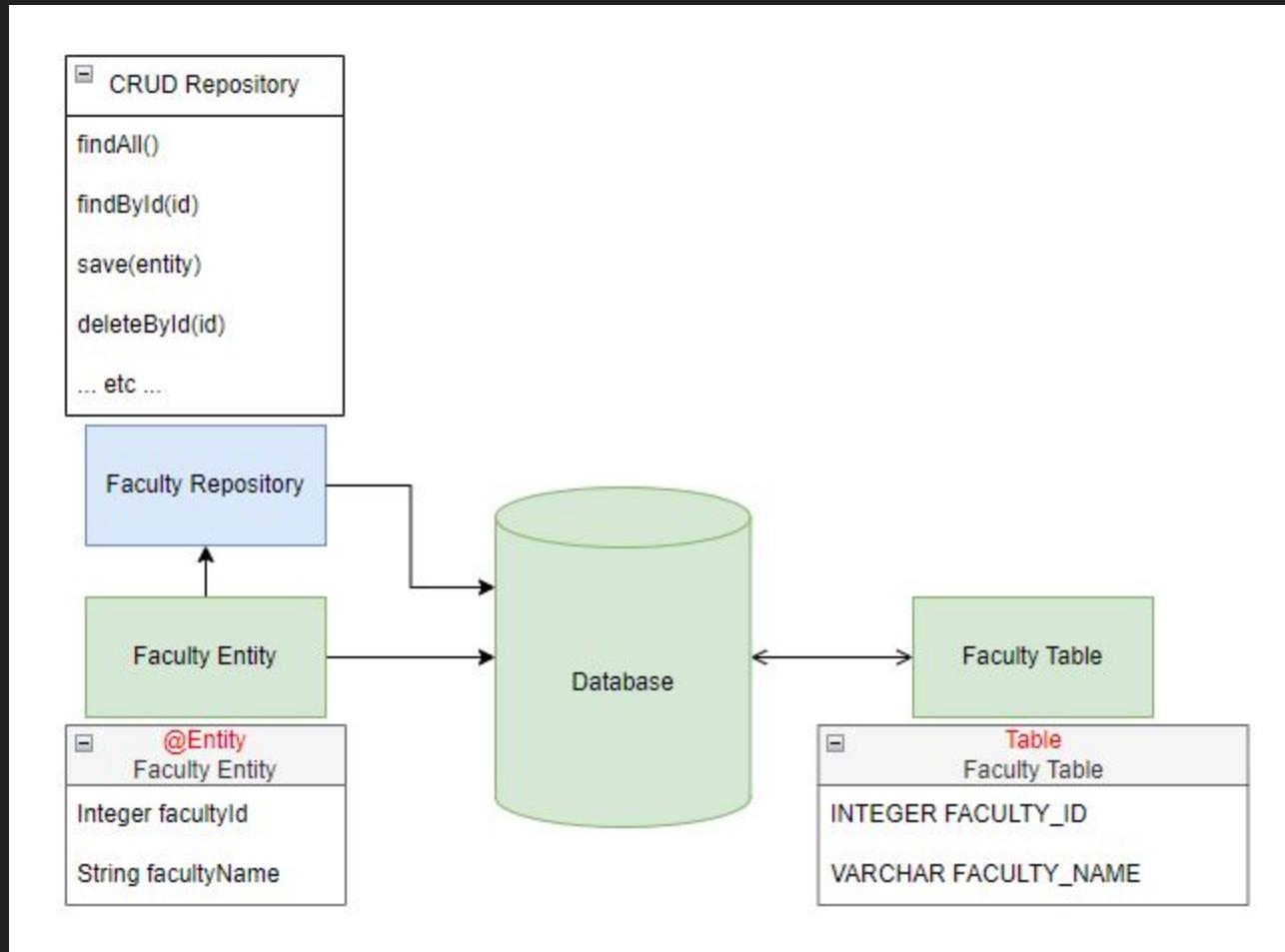
The screenshot shows the H2 Console interface with the following details:

- Toolbar:** Includes Document, H2 Console, and a URL bar pointing to `localhost:8080/h2-console/login.do?jsessionid=037f65a46cb2d1dab850dd2da9870...`.
- Left Sidebar (Database Structure):**
 - jdbc:h2:file:/data/student** (selected)
 - COURSE**: Contains COURSE_ID, COURSE_DESCRIPTION, COURSE_NAME, and Indexes.
 - ENROLL** (highlighted with a red box): Contains COURSE_ID, ENROLL_ID, STUDENT_ID, and Indexes.
 - FACULTY**
 - STUDENT**: Contains FACULTY_ID, STUDENT_ID, STUDENT_CODE, STUDENT_FIRST_NAME, STUDENT_LAST_NAME, and Indexes.
 - INFORMATION_SCHEMA**
 - Sequences**
 - Users**
- Central Area:** Shows the SQL statement input field with "SQL statement:" placeholder and several execution buttons: Run, Run Selected, Auto complete, and Clear.
- Important Commands Table:**

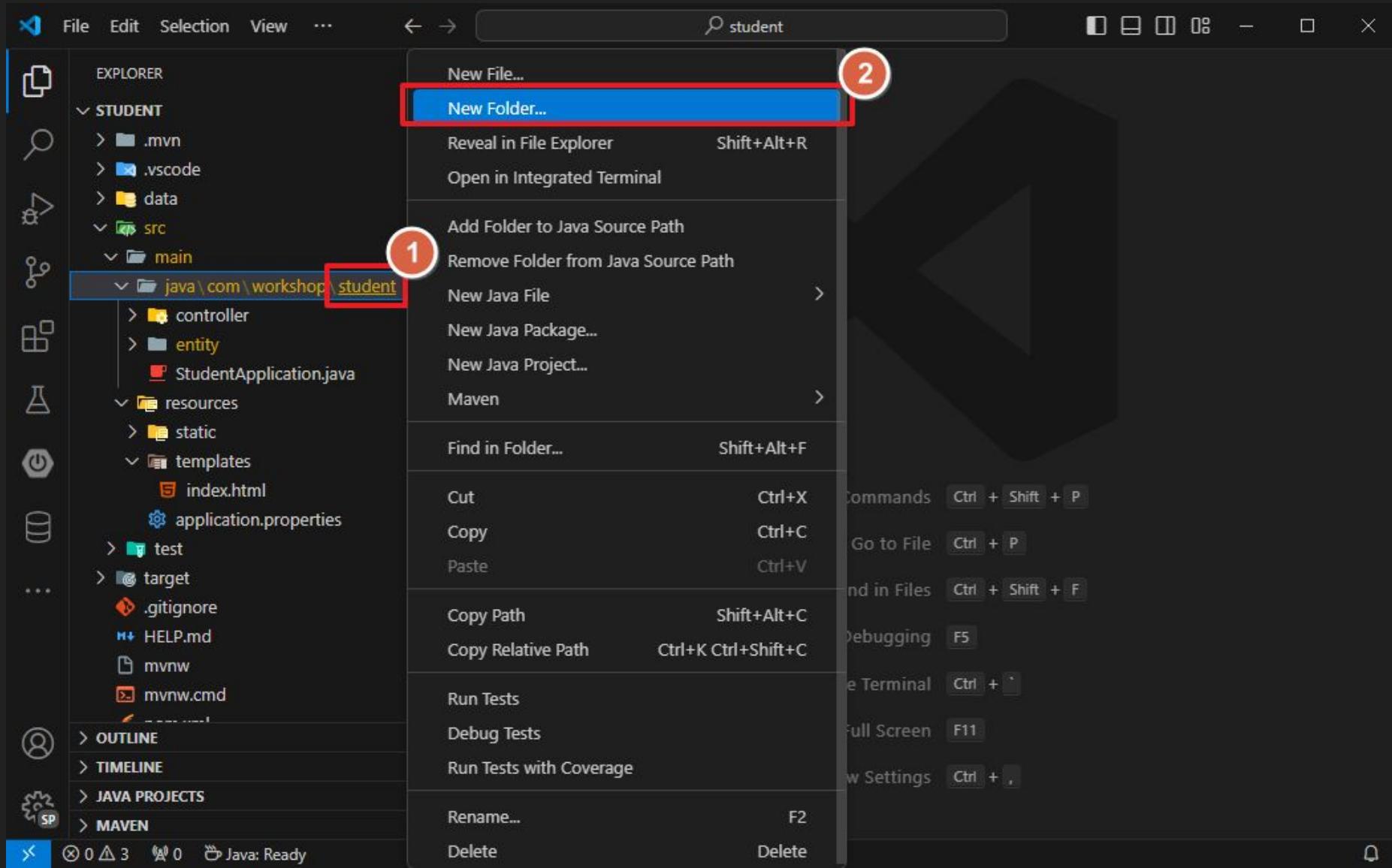
	Displays this Help Page
	Shows the Command History
	Ctrl+Enter Executes the current SQL statement
	Shift+Enter Executes the SQL statement defined by the text selection
	Ctrl+Space Auto complete
	Disconnects from the database
- Sample SQL Script:**

Delete the table if it exists	<code>DROP TABLE IF EXISTS TEST;</code>
Create a new table with ID and NAME columns	<code>CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));</code>
Add a new row	<code>INSERT INTO TEST VALUES(1, 'Hello');</code>
Add another row	<code>INSERT INTO TEST VALUES(2, 'World');</code>

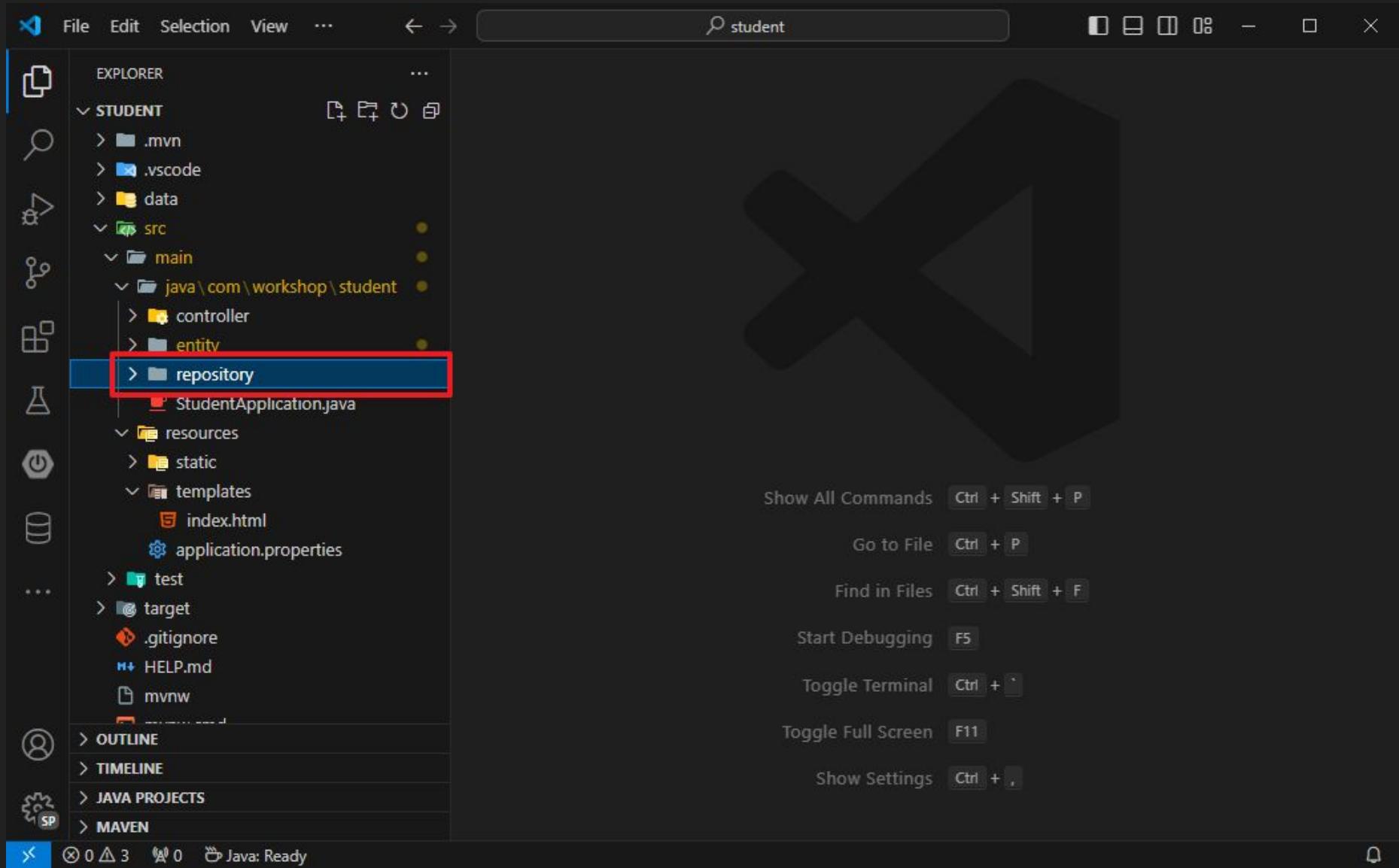
Entity, *Repository* and Service



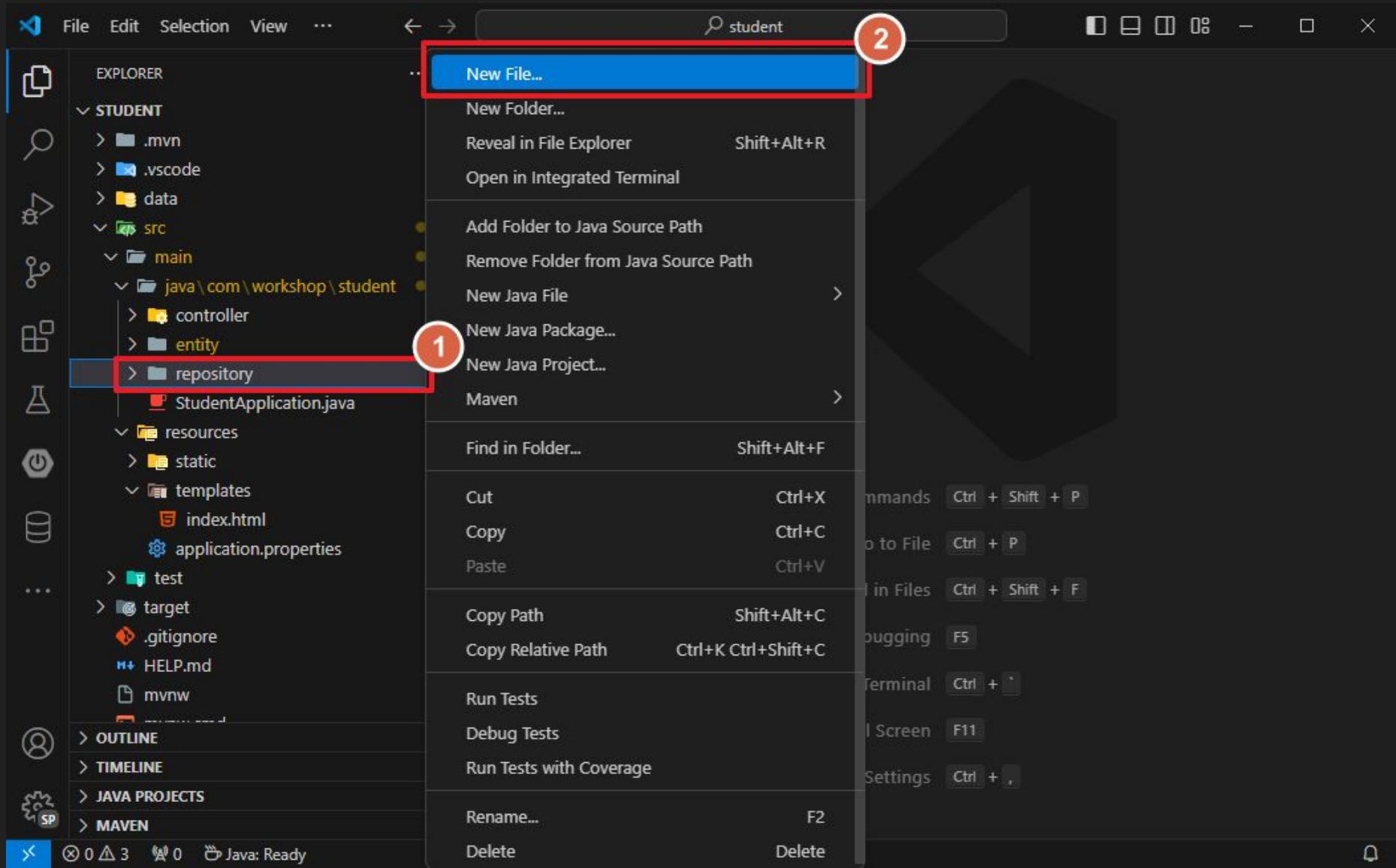
Create folder “Repository” and create FacultyRepository.java



Create folder “Repository” and create FacultyRepository.java



Create folder “Repository” and create FacultyRepository.java



Create folder “Repository” and create FacultyRepository.java

The screenshot shows a Java project structure in the Explorer sidebar under the 'STUDENT' folder. The 'repository' package contains a file named 'FacultyRepository.java'. The code for this interface is displayed in the main editor area:

```
1 package com.workshop.student.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import com.workshop.student.entity.FacultyEntity;
6
7 public interface FacultyRepository extends JpaRepository<FacultyEntity, Integer>{
8
9 }
10
```

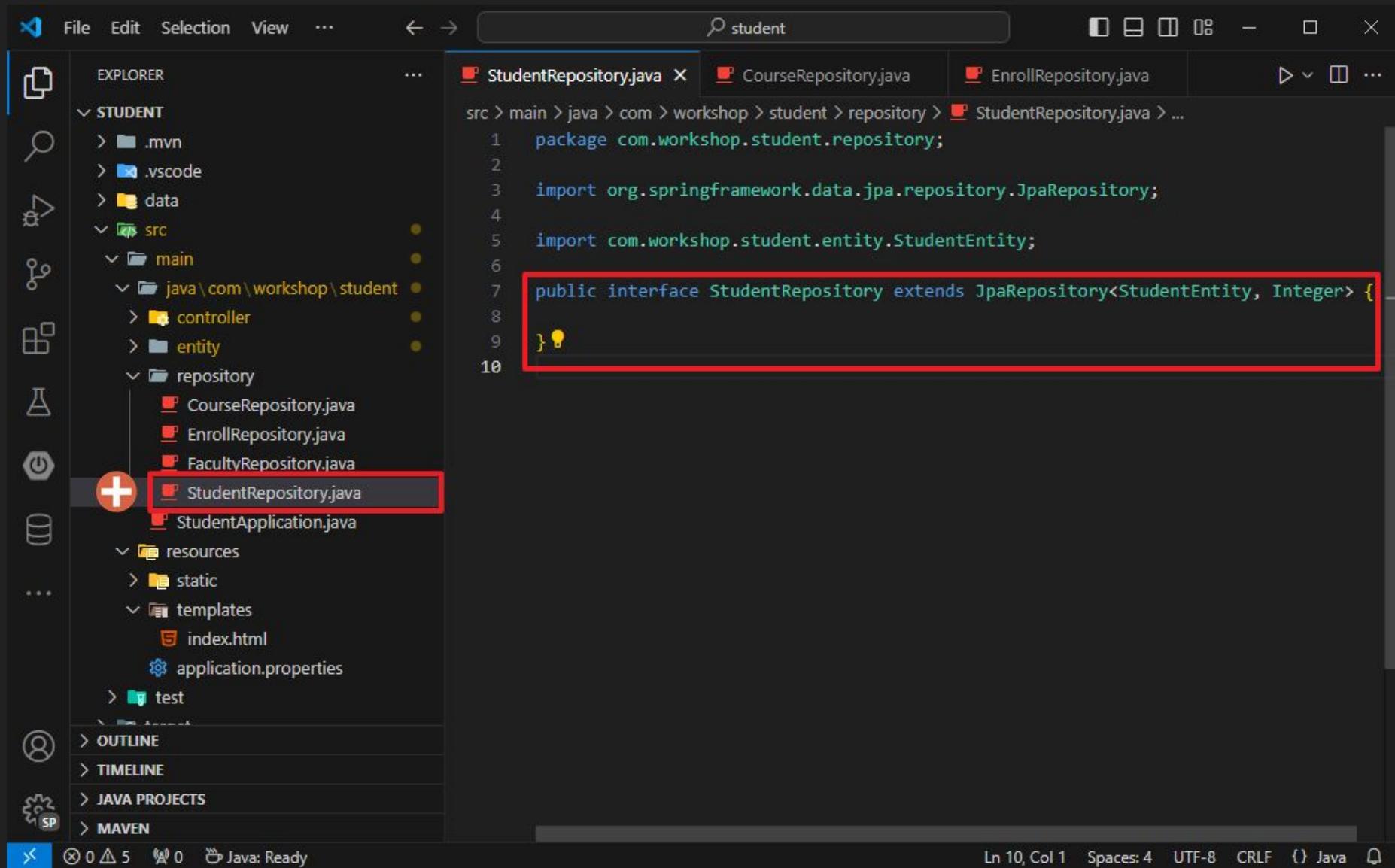
A red box highlights the entire interface definition. Below it, a larger orange box highlights the implementation of the 'FacultyEntity' class:

```
@Table(name = "faculty")
public class FacultyEntity {
    @Id
    @GeneratedValue
    private Integer facultyId;
    private String facultyName;

    @OneToMany(mappedBy = "faculty")
    private List<StudentEntity> students;
}
```

The status bar at the bottom indicates 'Java: Ready'.

Create StudentRepository.java



File Edit Selection View ... ⟲ ⟳ 🔎 student 📁 STUDENT .mvn .vscode data src main java\com\workshop\student controller entity repository CourseRepository.java EnrollRepository.java FacultyRepository.java StudentRepository.java StudentApplication.java resources static templates index.html application.properties test OUTLINE TIMELINE JAVA PROJECTS MAVEN ✘ 0 △ 5 ⚡ 0 Java: Ready Ln 10, Col 1 Spaces: 4 UTF-8 CRLF {} Java

```
1 package com.workshop.student.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import com.workshop.student.entity.StudentEntity;
6
7 public interface StudentRepository extends JpaRepository<StudentEntity, Integer> {
8
9 }
10 }
```

Create CourseRepository.java

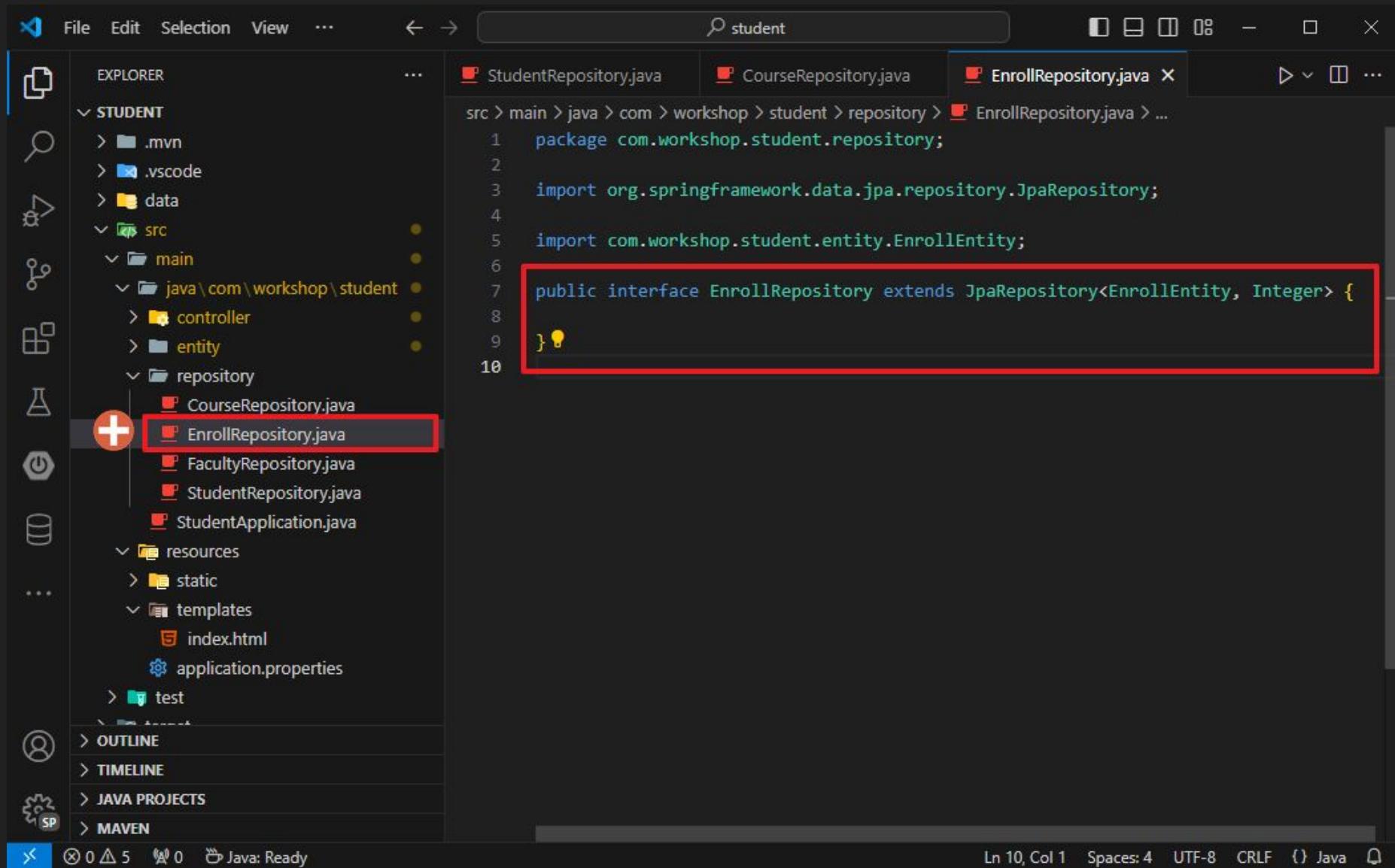
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the 'STUDENT' folder. The 'repository' folder contains 'CourseRepository.java', which is highlighted with a red border.
- Editor (Center):** Displays the code for 'CourseRepository.java'. The code is as follows:

```
1 package com.workshop.student.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import com.workshop.student.entity.CourseEntity;
6
7 public interface CourseRepository extends JpaRepository<CourseEntity, Integer> {
8
9 }
```

A red box highlights the entire code block.
- Bottom Status Bar:** Shows file statistics (0 0 5 0), Java status ('Java: Ready'), and terminal status (Ln 10, Col 1 Spaces: 4 UTF-8 CRLF {} Java).

Create EnrollRepository.java

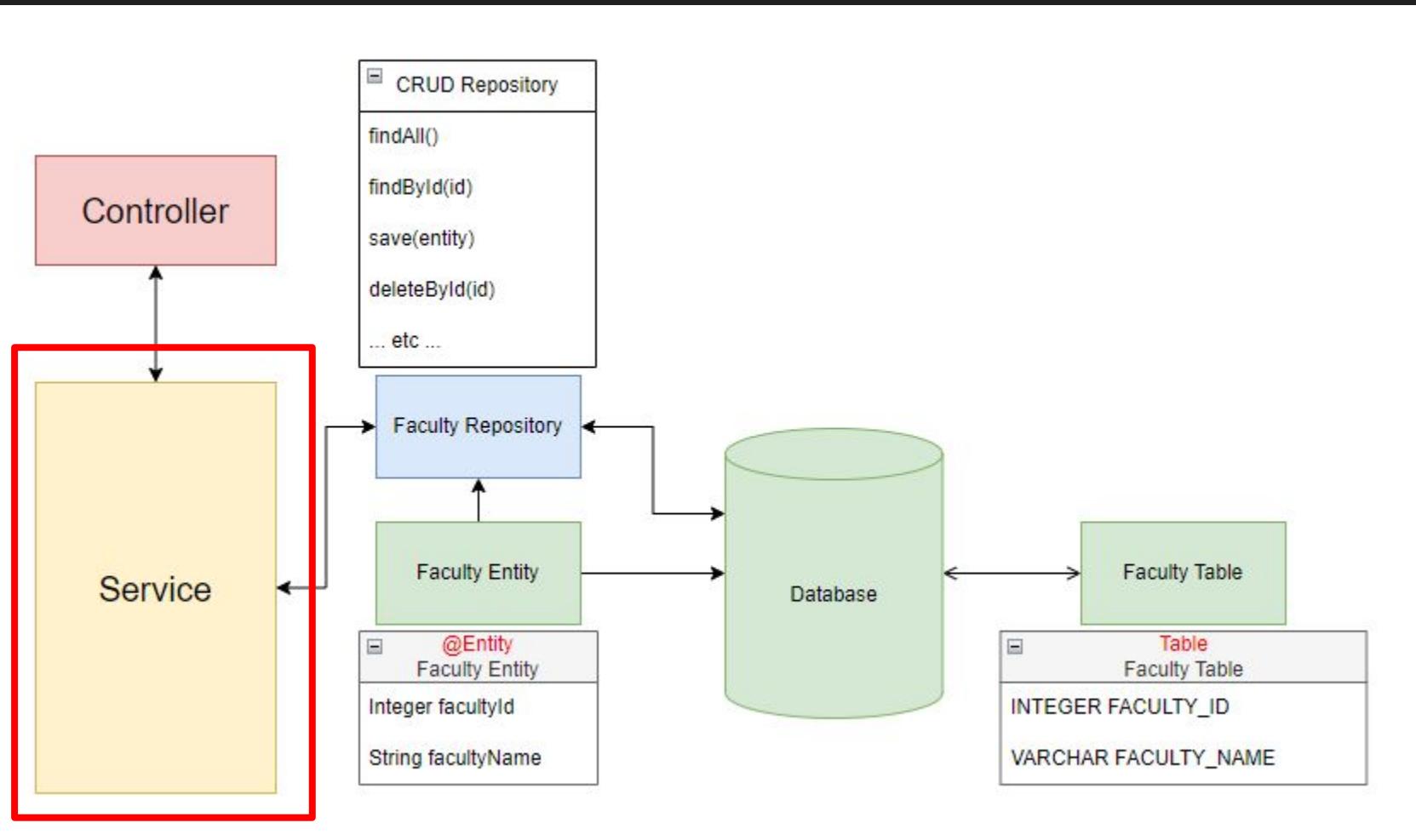


The screenshot shows a dark-themed interface of the Visual Studio Code code editor. On the left, the Explorer sidebar displays the project structure under the 'STUDENT' folder. The 'repository' folder contains several Java files: CourseRepository.java, EnrollRepository.java (which is currently selected and highlighted with a red box), FacultyRepository.java, StudentRepository.java, and StudentApplication.java. Below the repository folder are resources and templates, including static files like index.html and configuration files like application.properties. A large red box highlights the 'REPOSITORY' icon in the Explorer sidebar. In the center, the code editor shows the 'EnrollRepository.java' file with the following content:

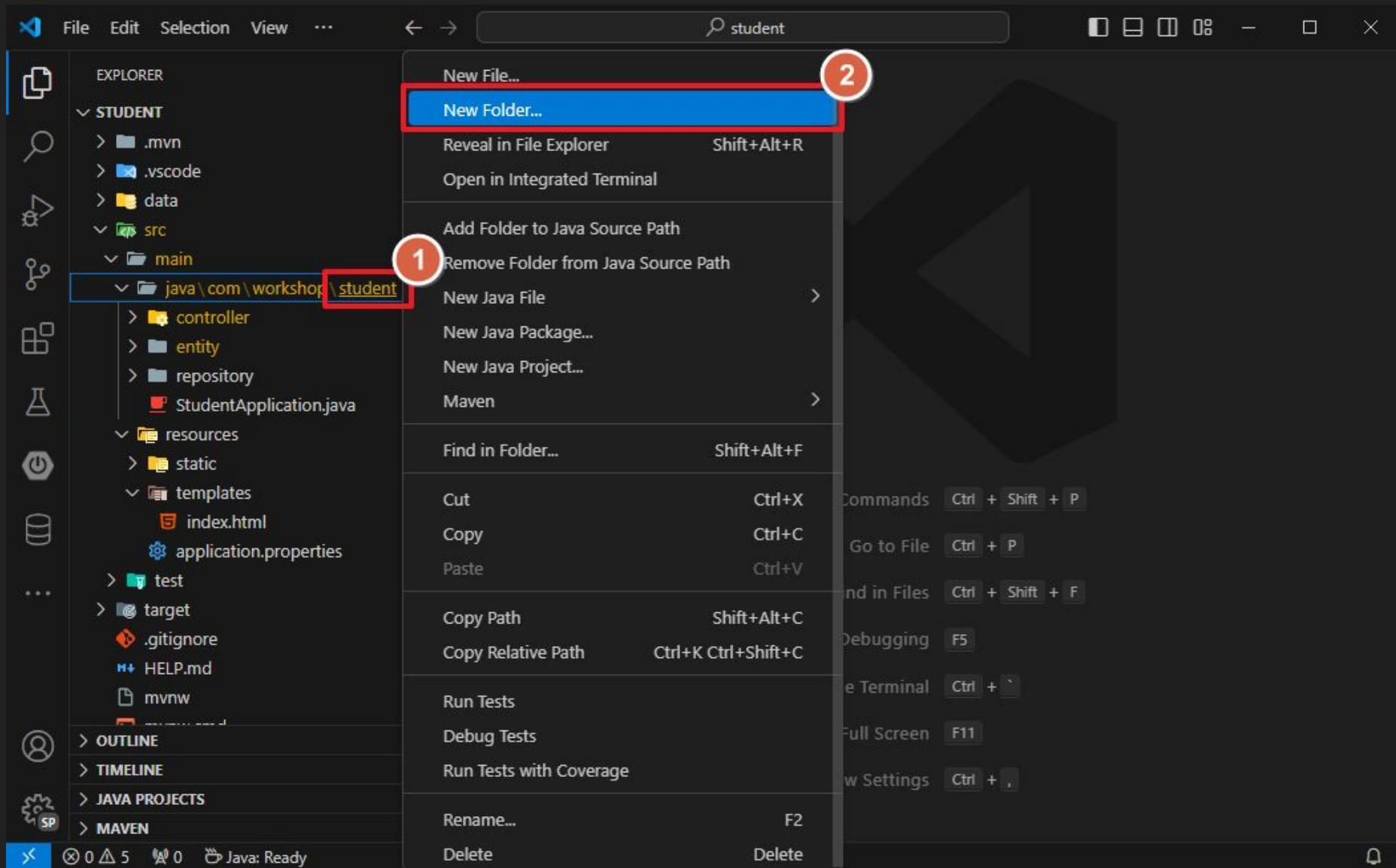
```
1 package com.workshop.student.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import com.workshop.student.entity.EnrollEntity;
6
7 public interface EnrollRepository extends JpaRepository<EnrollEntity, Integer> {
8
9 }
10
```

The status bar at the bottom indicates 'Java: Ready'.

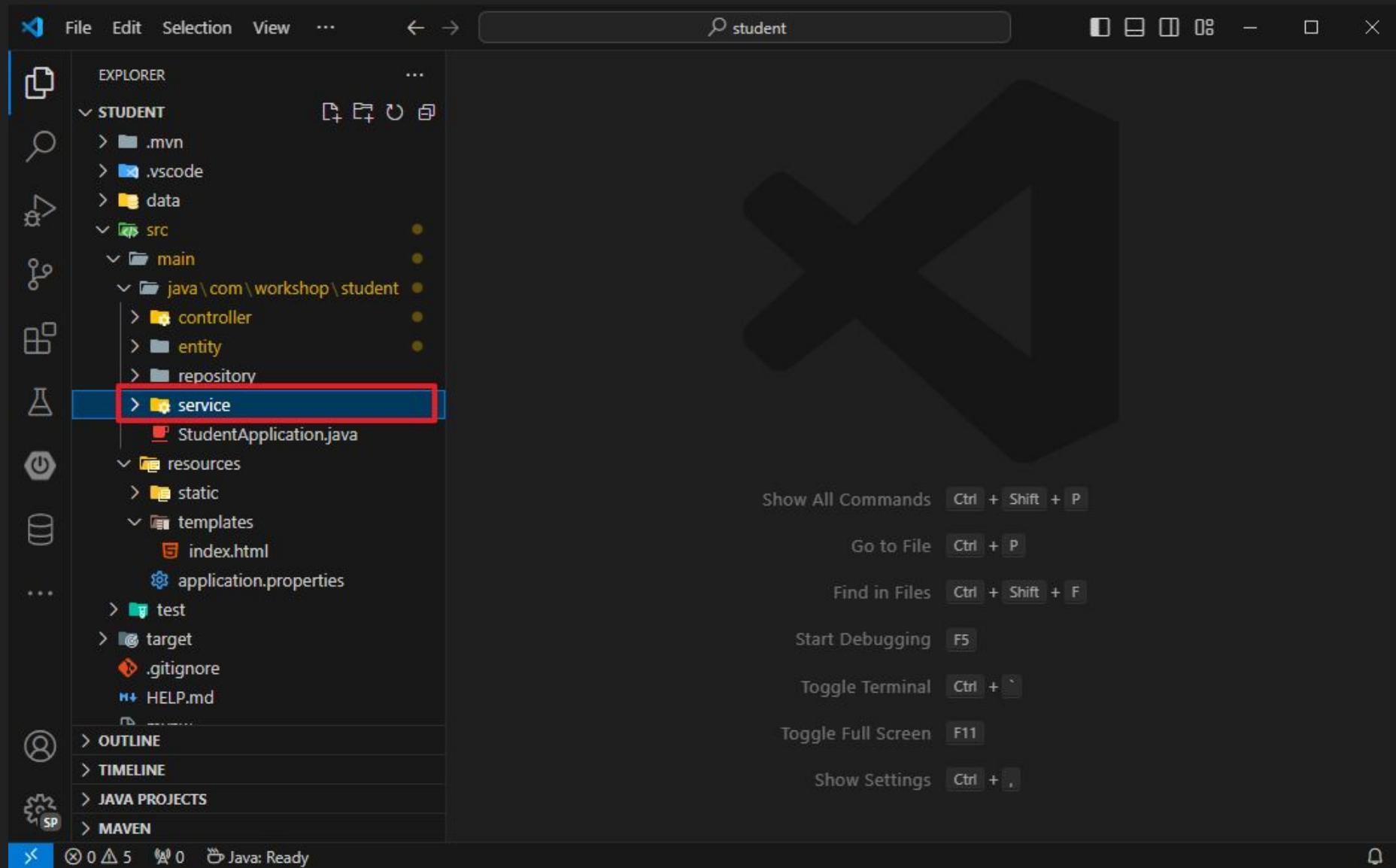
Entity, Repository and Service



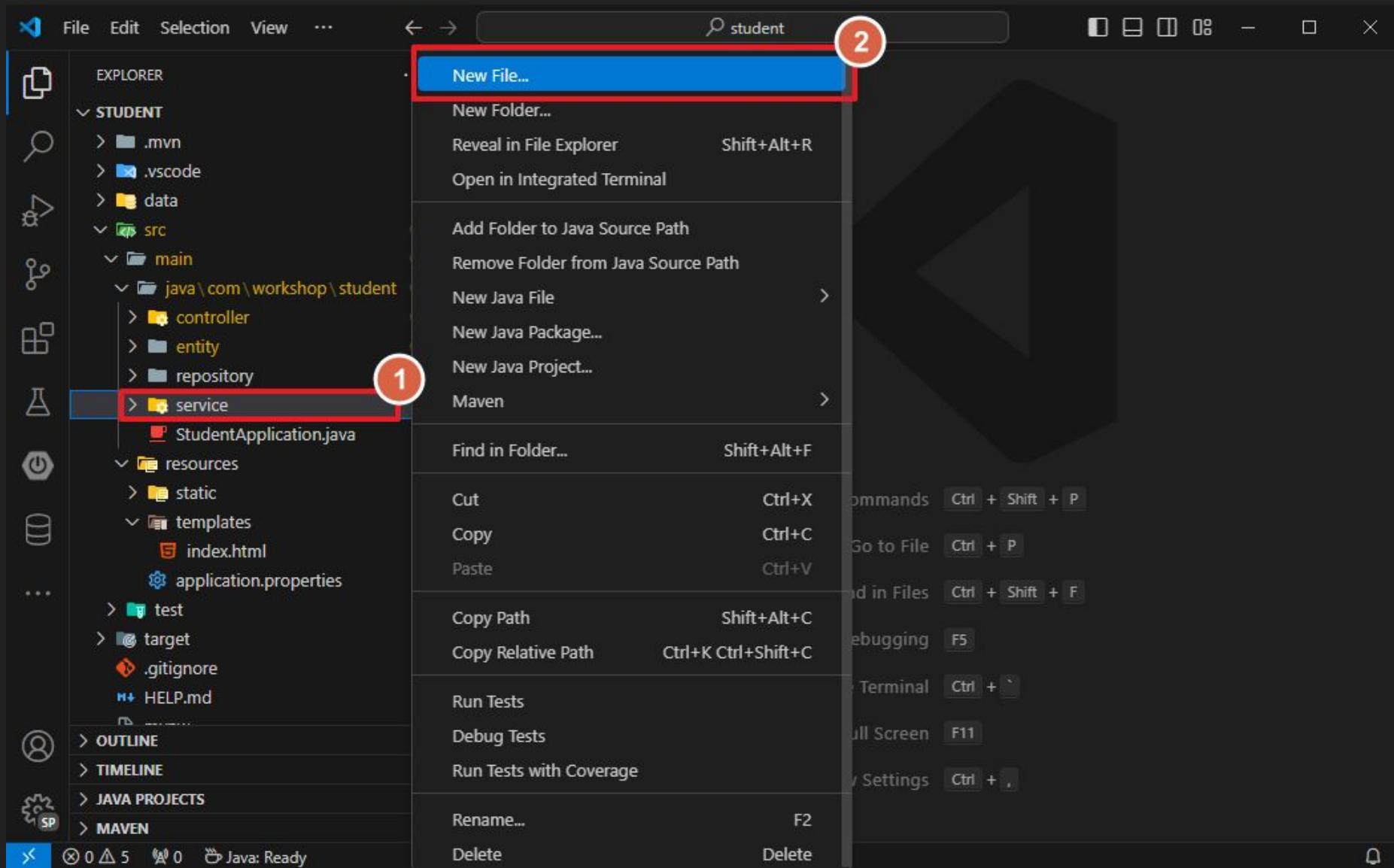
Create folder “Service” and FacultyService.java



Create folder “Service” and FacultyService.java



Create folder “Service” and FacultyService.java



Create folder “Service” and FacultyService.java

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the 'STUDENT' folder. The 'src/main/java/com/workshop/student/service' package contains 'FacultyService.java' and 'StudentApplication.java'. A red box highlights 'FacultyService.java' in the file list.
- Editor (Right):** Displays the content of 'FacultyService.java'. The code is as follows:

```
1 package com.workshop.student.service;
2
3 public class FacultyService {
4
5 }
```

A red box highlights the entire class definition from line 3 to line 6.
- Status Bar (Bottom):** Shows 'Ln 2, Col 1' and other status information.

FacultyService.java

The screenshot shows the VS Code interface with the file `FacultyService.java` open. The code implements a service for managing faculty entities. Annotations numbered 1 through 6 highlight specific parts of the code:

- Annotation 1: `@Service` - A Java annotation indicating that this class is a service.
- Annotation 2: `@Autowired` - A Java annotation used to automatically inject the `FacultyRepository`.
- Annotation 3: `getFacultyAll()` - A method that returns a list of all faculty entities.
- Annotation 4: `getFacultyById()` - A method that retrieves a faculty entity by its ID, returning `null` if it is not present.
- Annotation 5: `saveFaculty()` - A method that saves a new or updated faculty entity.
- Annotation 6: `deleteFacultyById()` - A method that deletes a faculty entity by its ID.

```
12  @Service 1
13  public class FacultyService {
14
15      @Autowired
16      private FacultyRepository facultyRepository;
17
18      public List<FacultyEntity> getFacultyAll() {
19          return facultyRepository.findAll();
20      }
21
22      public FacultyEntity getFacultyById(Integer facultyId) {
23          Optional<FacultyEntity> faculty = facultyRepository.findById(facultyId);
24          if(faculty.isPresent()) {
25              return faculty.get();
26          }
27          return null;
28      }
29
30      public FacultyEntity saveFaculty(FacultyEntity facultyEntity) {
31          FacultyEntity faculty = facultyRepository.save(facultyEntity);
32          return faculty;
33      }
34
35      public void deleteFacultyById(Integer facultyId) {
36          facultyRepository.deleteById(facultyId);
37      }
38
39 }
```

VS Code status bar: Ln 39, Col 2 Spaces: 4 UTF-8 CRLF {} Java

Create StudentService.java

The screenshot shows a Java project structure in the Explorer sidebar under the STUDENT folder. The StudentService.java file is selected and highlighted with a red border. The code editor displays the following Java code:

```
12  @Service
13  public class StudentService {
14
15      @Autowired
16      private StudentRepository studentRepository;
17
18      public List<StudentEntity> getStudentAll() {
19          return studentRepository.findAll();
20      }
21
22      public StudentEntity getStudentById(Integer studentId) {
23          Optional<StudentEntity> student = studentRepository.findById(studentId);
24          if(student.isPresent()) {
25              return student.get();
26          }
27          return null;
28      }
29
30      public StudentEntity saveStudent(StudentEntity studentEntity) {
31          StudentEntity student = studentRepository.save(studentEntity);
32          return student;
33      }
34
35      public void deleteStudentById(Integer studentId) {
36          studentRepository.deleteById(studentId);
37      }
38
39 }
```

The status bar at the bottom indicates "Java: Ready".

Create CourseService.java

The screenshot shows a Java project structure in the Explorer sidebar. The 'CourseService.java' file is selected and highlighted with a red box. The code editor displays the following Java code:

```
12 @Service
13 public class CourseService {
14
15     @Autowired
16     private CourseRepository courseRepository;
17
18     public List<CourseEntity> getCourseAll() {
19         return courseRepository.findAll();
20     }
21
22     public CourseEntity getCourseById(Integer courseId) {
23         Optional<CourseEntity> course = courseRepository.findById(courseId);
24         if(course.isPresent()) {
25             return course.get();
26         }
27         return null;
28     }
29
30     public CourseEntity saveCourse(CourseEntity courseEntity) {
31         CourseEntity course = courseRepository.save(courseEntity);
32         return course;
33     }
34
35     public void deleteCourseById(Integer courseId) {
36         courseRepository.deleteById(courseId);
37     }
38
39 }
```

The status bar at the bottom indicates 'Java: Ready'.

Create EnrollService.java

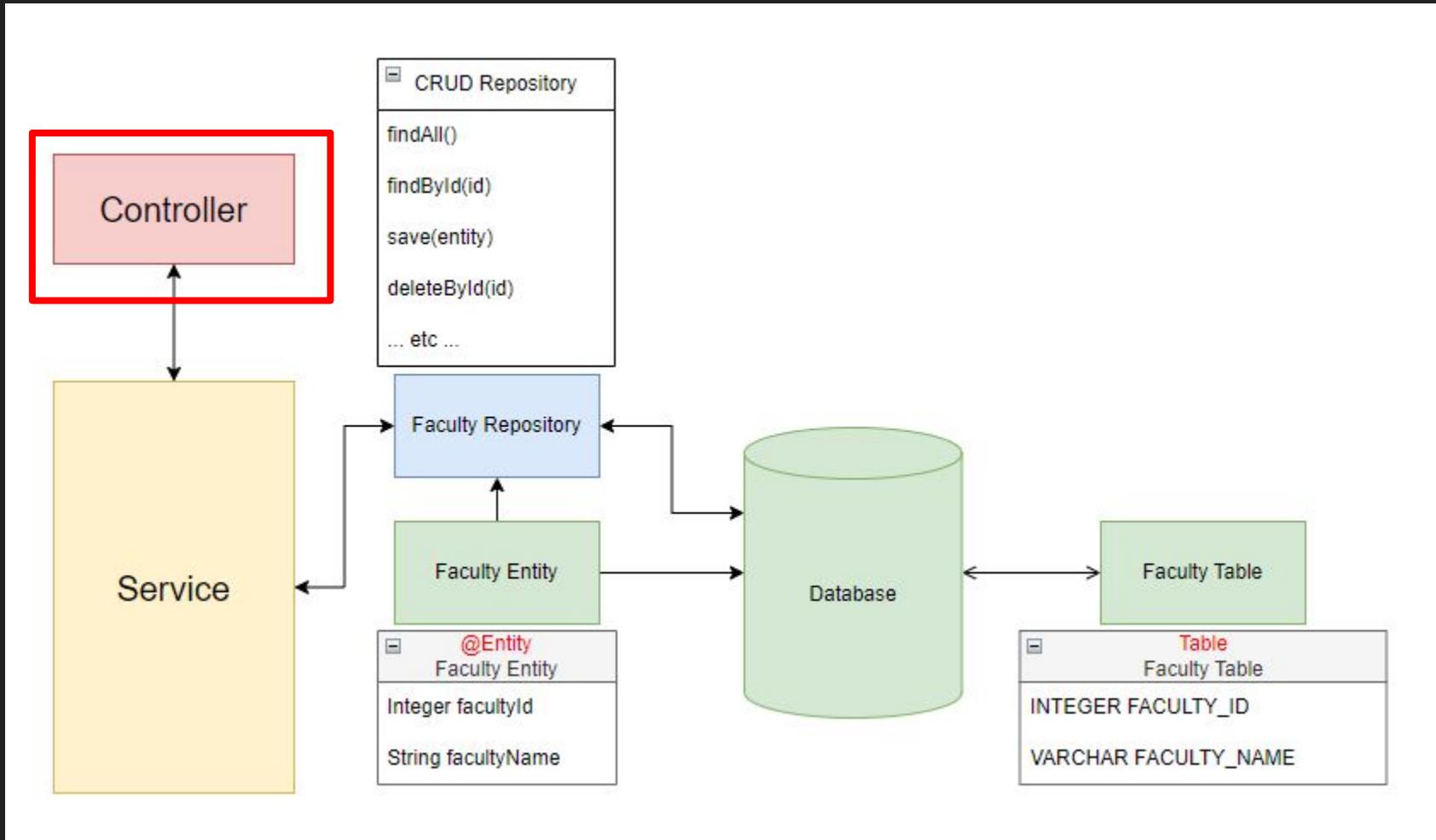
The screenshot shows a Java project structure in the Explorer sidebar under the 'STUDENT' category. The 'src' folder is highlighted with a red box. Inside 'src', there are 'main', 'java', 'com', 'workshop', 'student', 'service', 'CourseService.java', 'EnrollService.java', 'FacultyService.java', 'StudentService.java', and 'StudentApplication.java'. The 'resources' folder contains 'static' and 'templates' with 'index.html' and 'application.properties'. The 'OUTLINE', 'TIMELINE', 'JAVA PROJECTS', and 'MAVEN' sections are also visible.

The code editor displays `EnrollService.java` with the following content:

```
12  @Service
13  public class EnrollService {
14
15      @Autowired
16      private EnrollRepository enrollRepository;
17
18      public List<EnrollEntity> getEnrollAll() {
19          return enrollRepository.findAll();
20      }
21
22      public EnrollEntity getEnrollById(Integer enrollId) {
23          Optional<EnrollEntity> enroll = enrollRepository.findById(enrollId);
24          if(enroll.isPresent()) {
25              return enroll.get();
26          }
27          return null;
28      }
29
30      public EnrollEntity saveEnroll(EnrollEntity enrollEntity) {
31          EnrollEntity course = enrollRepository.save(enrollEntity);
32          return course;
33      }
34
35      public void deleteEnrollById(Integer enrollId) {
36          enrollRepository.deleteById(enrollId);
37      }
38
39 }
```

The status bar at the bottom indicates 'Java: Ready'.

Controller and Service



Controller and Service, Faculty

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "STUDENT". The "controller" folder contains "CourseController.java", "EnrollController.java", "FacultyController.java" (highlighted with a red box), "StudentController.java", and "TutorialController.java". Other folders like ".mvn", ".vscode", "data", "src", "entity", "repository", "service", "resources", "static", and "templates" are also listed.
- Editor (Right):** The file "FacultyController.java" is open. The code defines a controller for the "/faculty" endpoint. It includes an autowired service dependency and two methods: "getAll()" and "getById()".

```
19 @Controller
20 @RequestMapping("/faculty")
21 public class FacultyController {
22
23     @Autowired
24     private FacultyService facultyService;
25
26     @GetMapping("", "/")
27     public String getAll() {
28         System.out.println("---- getAll() ----");
29
30         List<FacultyEntity> faculties = facultyService.getFacultyAll();
31         System.out.println("---- getAll() Result ----");
32         System.out.println("Size: " + faculties.size());
33         return "index";
34     }
35
36     @GetMapping("/{faculty-id}")
37     public String getById(
38         @PathVariable(name = "faculty-id") Integer facultyId
39     ) {
40
41         System.out.println("---- getById() ----");
42         System.out.println("faculty-id: " + facultyId);
43
44         FacultyEntity entity = facultyService.getFacultyById(facultyId);
45         System.out.println("---- getById() Result ----");
46         System.out.println("Faculty Name: " + entity.getFacultyName());
47     }
48 }
```
- Annotations:**
 - Annotation **1** highlights the autowiring line: `@Autowired private FacultyService facultyService;`.
 - Annotation **2** highlights the entire body of the `getAll()` method, including the service call and output statements.
- Status Bar (Bottom):** Shows "Java: Ready", "Ln 25, Col 1", "Spaces: 4", "UTF-8", "CRLF", and file navigation icons.

Controller and Service, Faculty

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder. The "controller" folder contains five files: CourseController.java, EnrollController.java, FacultyController.java (highlighted with a red box), StudentController.java, and TutorialController.java.
- Search Bar (Top):** Contains the text "student".
- Code Editor (Right):** Displays the content of FacultyController.java. The code defines a controller class with methods for getting all faculties, getting a faculty by ID, and deleting a faculty by ID. A specific section of the code for getting a faculty by ID is highlighted with a red box:

```
        FacultyEntity entity = facultyService.getFacultyById(facultyId);
        System.out.println(x:"---- getDeleteById() ----");
        System.out.println("Faculty Name: " + entity.getFacultyName());
```
- Bottom Status Bar:** Shows "Java: Ready" and other status indicators like file counts and Java version.

Controller and Service, Faculty

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Back, Forward, Find, Replace, Copy, Paste, Cut, Delete, Undo, Redo, Minimize, Maximize, Close.
- Left Sidebar (EXPLORER):**
 - STUDENT**: .mvn, .vscode (settings.json), data, src (main, java\com\workshop\stud..., controller, CourseController.java, EnrollController.java, FacultyController.java, StudentController.java, TutorialController.java), entity, repository, service, StudentApplication.java, resources, static, templates.
 - OUTLINE**, **TIMELINE**, **JAVA PROJECTS**, **MAVEN**.
- Code Editor:** FacultyController.java (Language Support for Java(TM) by Red Hat). The code is as follows:

```
public class FacultyController {  
    public String getById(  
        FacultyEntity entity = facultyService.getFacultyById(facultyId);  
        System.out.println("---- getById() Result ----");  
        System.out.println("Faculty Name: " + entity.getFacultyName());  
  
        return "index";  
    }  
  
    @GetMapping("/delete/{faculty-id}")  
    public String getDeleteById(  
        @PathVariable(name = "faculty-id") Integer facultyId  
    ) {  
        System.out.println("---- getDeleteById() ----");  
        System.out.println("faculty-id: " + facultyId);  
  
        System.out.println("---- getDeleteById() Result ----");  
        facultyService.deleteFacultyById(facultyId);  
  
        return "index";  
    }  
  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        @RequestParam() Map<String, String> param  
    ) {  
        System.out.println("---- postInsertAndUpdate() ----");  
        System.out.println("faculty-id: " + param.get(key:"faculty-id"));  
    }  
}
```
- Bottom Status Bar:** Ln 49, Col 1, Spaces: 4, UTF-8, CRLF, {}, Java, Q.

Controller and Service, Faculty

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "STUDENT". The "controller" folder contains five files: CourseController.java, EnrollController.java, FacultyController.java (highlighted with a red box), StudentController.java, and TutorialController.java.
- Editor (Right):** Displays the content of FacultyController.java. A yellow bulb icon is shown at line 61, indicating a potential issue or suggestion.
- Code Content:**

```
21  public class FacultyController {  
22      public String getDeleteById(  
23          @RequestParam("faculty-id") Integer facultyId) {  
24          facultyService.deleteFaculty(facultyId);  
25          return "index";  
26      }  
27  
28      @PostMapping("/")  
29      public String postInsertAndUpdate(  
30          @RequestParam() Map<String, String> param) {  
31          System.out.println("---- postInsertAndUpdate() ----");  
32          System.out.println("faculty-id: " + param.get("faculty-id"));  
33          System.out.println("faculty-name: " + param.get("faculty-name"));  
34  
35          System.out.println("---- postInsertAndUpdate() Result ----");  
36          FacultyEntity entity = new FacultyEntity();  
37          if(null != param.get("faculty-id")) {  
38              entity.setFacultyId(Integer.parseInt(param.get("faculty-id")));  
39          }  
40          entity.setFacultyName(param.get("faculty-name"));  
41          FacultyEntity result = facultyService.saveFaculty(entity);  
42          System.out.println("Faculty ID: " + result.getFacultyId());  
43          System.out.println("Faculty Name: " + result.getFacultyName());  
44  
45          return "index";  
46      }  
47  }
```
- Status Bar (Bottom):** Shows "Java: Ready" and other status indicators.

Controller and Service, Faculty (Test Get All)

The screenshot shows the Thunder Client interface with the following details:

- Request URL:** GET localhost:8080/faculty/ (highlighted with red box #1)
- Action Buttons:** Send button (highlighted with red box #2)
- Activity Tab:** Active tab (highlighted with red box #3)
- Request History:** GET faculty (17 hours ago) and POST faculty (17 hours ago) listed on the left.
- Status Bar:** Status: 200 OK, Size: 293 Bytes, Time: 53 ms
- Terminal Output:** Shows Java code for fetching all faculty records using Hibernate's getAll() method.
- Result Area:** getAll() Result - Size: 0 (highlighted with red box #4)

Controller and Service, Faculty (Test Insert 1)

The screenshot shows the Thunder Client interface with the following steps highlighted:

1. URL: POST localhost:8080/faculty/
2. Body tab selected.
3. Form tab selected.
4. Faculty Name field: checked, value "Computer".
5. Send button.

Below the request, the Response tab displays the generated SQL query and its execution results:

```
select next value for faculty_seq
Hibernate:
    insert
    into
        faculty
        (faculty_name, faculty_id)
    values
        (?, ?)
```

Execution results:

```
Faculty ID: 1
Faculty Name: Computer
```

A green checkmark icon is positioned next to the results.

Controller and Service, Faculty (Test Insert 2)

The screenshot shows the Thunder Client interface with the following steps highlighted:

1. Method dropdown set to POST.
2. Body tab selected.
3. Form tab selected.
4. Faculty Name field filled with "Law".
5. Send button highlighted.

The Response section shows the generated SQL query and the successful insertion of a new faculty record:

```
select next value for faculty_seq
Hibernate:
    insert
    into
        faculty
        (faculty_name, faculty_id)
    values
        (?, ?)
Faculty ID: 2
Faculty Name: Law
```

A green checkmark icon is present next to the response text.

Controller and Service, Faculty (Test Update)

The screenshot shows the Thunder Client interface for testing RESTful APIs. A POST request is being prepared to update a faculty record.

Request Details:

- Method: POST (1)
- URL: localhost:8080/faculty/ (1)
- Body Type: Form (2)
- Body Encoding: JSON (3)

Form Fields:

- faculty-name (checked) (4)
- faculty-id (checked) (4)
- field name (unchecked) (4)

Status and Response:

- Status: 200 OK
- Size: 293 Bytes
- Time: 53 ms
- Response (Terminal):

```
where  
    fe1_0.faculty_id=?  
Hibernate:  
    update  
        faculty  
    set  
        faculty_name=?  
    where  
        faculty_id=?  
Faculty ID: 2  
Faculty Name: Law Edit
```

A large red box highlights the "Body" section, and a green checkmark is placed over the "Faculty Name: Law Edit" entry in the response terminal.

Controller and Service, Faculty (Test Get All)

The screenshot shows the Thunder Client interface with the following details:

- Request URL:** GET localhost:8080/faculty/ (highlighted by a red box labeled 1)
- Send Button:** A blue "Send" button with a red border and a red circle containing the number 2.
- Status Bar:** Status: 200 OK, Size: 293 Bytes, Time: 28 ms
- Terminal Output:**

```
Faculty ID: 2
Faculty Name: Law Edit
----- getAll() -----
Hibernate:
    select
        fe1_0.faculty_id,
        fe1_0.faculty_name
    from
        faculty fe1_0
----- getAll() Result -----
Size: 2
```
- Result Area:** A green checkmark icon is positioned next to the terminal output.

Controller and Service, Faculty (Test Get By Id)

The screenshot shows the Thunder Client interface. On the left, there's a sidebar with various icons and activity logs. The main area has a search bar at the top with the text "student". Below it, there are tabs for "FacultyController.java", "faculty" (selected), and another "faculty" tab. A red box highlights the "GET" method and the URL "localhost:8080/faculty/1" in the request bar, with a red circle containing the number "1" above it. To the right of the URL is a "Send" button with a red box around it and a red circle containing the number "2" above it. The "Query" tab is selected, showing a "Query Parameters" section with a parameter table. Below the request bar, the status is shown as "Status: 200 OK Size: 293 Bytes Time: 18 ms". The "TERMINAL" tab is active, displaying the SQL query used to retrieve the data. The output window shows the result: "----- getById() Result ----- Faculty Name: Computer" with a green checkmark icon.

THUNDER CLIENT

New Request

Activity Collections Env

filter activity

GET faculty just now

POST faculty just now

File Edit Selection View ...

student

FacultyController.java

faculty

faculty

GET localhost:8080/faculty/1

1

Send

Query Headers² Auth Body Tests Pre Run

Query Parameters

parameter value

Status: 200 OK Size: 293 Bytes Time: 18 ms

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL

Maven-student

faculty-id: 1
Hibernate:
 select
 fe1_0.faculty_id,
 fe1_0.faculty_name
 from
 faculty fe1_0
 where
 fe1_0.faculty_id=>
----- getById() Result -----
Faculty Name: Computer

200 OK

293 Bytes

18 ms

getById() Result

Faculty Name: Computer

✓

SP

Java: Ready

Controller and Service, Faculty (Check Database)

The screenshot shows the H2 Console interface running on localhost:8080. The left sidebar lists database objects: COURSE, ENROLL, FACULTY (highlighted with a red box and number 1), STUDENT, INFORMATION_SCHEMA, Sequences, and Users. The main area displays a SQL statement: "SELECT * FROM FACULTY;". The result set shows two rows: (1, Computer) and (2, Law Edit). A large green checkmark is overlaid on the result table. The top navigation bar includes tabs for Auto commit (checked), rows (set to 1000), and various tool icons.

1

2

3

```
SELECT * FROM FACULTY;
```

FACULTY_ID	FACULTY_NAME
1	Computer
2	Law Edit

(2 rows, 0 ms)

Edit

Controller and Service, Faculty (Test Delete)

The screenshot shows the Thunder Client interface with the following details:

- Request URL:** GET localhost:8080/faculty/delete/1 (highlighted with red box #1)
- Send Button:** A blue "Send" button with a red border and a white number "2" (highlighted with red box #2).
- Activity Tab:** The "Activity" tab is selected, showing recent operations: a GET request to "faculty" just now and a POST request to "faculty" just now.
- Response Terminal:** The terminal shows the SQL query generated by Hibernate for the delete operation:

```
fe1_0.faculty_name
from
faculty fe1_0
where
fe1_0.faculty_id=?
```

Hibernate:
delete
from
faculty
where
faculty_id=?

A green checkmark icon is positioned next to the generated SQL code.
- Bottom Status Bar:** Java: Ready

Controller and Service, Faculty (Check Database)

The screenshot shows the H2 Console interface running on localhost:8080. The left sidebar lists database objects: COURSE, ENROLL, FACULTY (selected), STUDENT, INFORMATION_SCHEMA, Sequences, and Users. A red circle labeled '1' is around the FACULTY entry. The main area shows a SQL statement: 'SELECT * FROM FACULTY;'. A red circle labeled '2' is around the SELECT statement. Another red circle labeled '3' is around the 'Run' button. The result set displays one row: FACULTY_ID 2 and FACULTY_NAME Law Edit. A green checkmark icon is positioned next to the result set. The status bar at the bottom shows 'Edit'.

localhost8080

H2 Console

localhost:8080/h2-console/login.do?jsessionid=cf93ce4975a967bda8151b25...

Auto commit

rows: 1000

Auto complete Off

Auto select On

SQL statement:

jdbc:h2:file:/data/student

COURSE

ENROLL

FACULTY 1

STUDENT

INFORMATION_SCHEMA

Sequences

Users

H2 2.2.224 (2023-09-17)

SELECT * FROM FACULTY;

FACULTY_ID FACULTY_NAME

2 Law Edit

(1 row, 2 ms)

Edit

Controller and Service, Student

The screenshot shows the VS Code interface with the file `StudentController.java` open. The code defines a controller for student management, utilizing services for student and faculty operations.

```
18  
19  
20 @Controller  
21 @RequestMapping("/student")  
22 public class StudentController {  
23       
24     @Autowired  
25     private StudentService studentService;  
26       
27     @Autowired  
28     private FacultyService facultyService;  
29       
30     @GetMapping({"", "/"})  
31     public String getAll() {  
32         System.out.println("----- StudentController getAll() -----");  
33           
34         List<StudentEntity> students = studentService.getStudentAll();  
35         System.out.println("----- StudentController getAll() Result -----");  
36         System.out.println("Size: " + students.size());  
37           
38         return "index";  
39     }  
40       
41     @GetMapping("/{student-id}")  
42     public String getById(  
43         @PathVariable(name = "student-id") Integer studentId  
44     ) {  
45         System.out.println("----- StudentController getById() -----");  
46     }  
47 }
```

Annotations:

- Annotation 1 highlights the autowiring of the `StudentService` and `FacultyService` beans.
- Annotation 2 highlights the logic for retrieving all student records and printing the result.

VS Code status bar at the bottom: Ln 40, Col 2 Spaces: 4 UTF-8 CRLF {} Java

Controller and Service, Student

The screenshot shows an IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT:** .mvn, .vscode (with settings.json), data, src (main, java, controller, entity, repository, service, resources, static, templates).
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN:** Standard project management sections.
- Central Area:** Code editor for `StudentController.java`.

```
public class StudentController {  
    public String getAll() {  
        System.out.println("Size: " + students.size());  
        return "index";  
    }  
  
    @GetMapping("/{student-id}")  
    public String getById(  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        System.out.println("----- StudentController getById() -----");  
        System.out.println("student-id: " + studentId);  
  
        StudentEntity entity = studentService.getStudentById(studentId);  
        System.out.println("----- StudentController getById() Result -----");  
        System.out.println("Student First Name: " + entity.getStudentFirstName());  
        System.out.println("Student Last Name: " + entity.getStudentLastName());  
  
        return "index";  
    }  
  
    @GetMapping("/delete/{student-id}")  
    public String getDeleteById(  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        System.out.println("----- StudentController getDeleteById() -----");  
        System.out.println("student-id: " + studentId);  
    }  
}
```
- Bottom Status Bar:** Ln 55, Col 1, Spaces: 4, UTF-8, CRLF, Java, etc.

Controller and Service, Student

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard toolbar icons.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - .mvn
 - .vscode
 - settings.json
 - data
 - src
 - main
 - java\com\workshop\stud...
 - controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java** (highlighted with a red box)
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
 - Central Area:** Code editor showing **StudentController.java**.

```
public class StudentController {  
    public String getById(  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        System.out.println("----- StudentController getById() -----");  
        System.out.println("student-id: " + studentId);  
  
        System.out.println("----- StudentController getById() Result -----");  
        studentService.deleteStudentById(studentId);  
  
        return "index";  
    }  
  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        @RequestParam() Map<String, String> param  
    ) {  
        System.out.println("----- StudentController postInsertAndUpdate() -----");  
        System.out.println("student-id: " + param.get(key:"student-id"));  
        System.out.println("student-code: " + param.get(key:"student-code"));  
        System.out.println("student-fname: " + param.get(key:"student-fname"));  
        System.out.println("student-lname: " + param.get(key:"student-lname"));  
    }  
}
```
 - Bottom Status Bar:** Ls 68, Col 6, Spaces: 4, UTF-8, CRLF, Java

Controller and Service, Student

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Back, Forward, Home, Stop, Refresh, Minimize, Maximize, Close.
- Left Sidebar (EXPLORER):**
 - STUDENT**: .mvn, .vscode (settings.json), data, src (main, java, entity, repository, service, StudentApplication.java), resources (static, templates).
 - OUTLINE**, **TIMELINE**, **JAVA PROJECTS**, **MAVEN**.
- Central Area (Code Editor):** StudentController.java (highlighted with a red border). The code implements a POST method to insert or update a student record. It prints various parameters to the console and uses the FacultyService to get a faculty entity by ID. It then creates a new StudentEntity, sets its properties, and saves it using the studentService. Finally, it prints the student ID, first name, and last name of the saved student.

```
public class StudentController {
    public String postInsertAndUpdate(
        System.out.println("student-code: " + param.get(key:"student-code"));
        System.out.println("student-fname: " + param.get(key:"student-fname"));
        System.out.println("student-lname: " + param.get(key:"student-lname"));

        System.out.println("faculty-id: " + param.get(key:"faculty-id"));

        System.out.println(x:"----- StudentController postInsertAndUpdate() Result -----");
        Integer facultyId = Integer.parseInt(param.get(key:"faculty-id"));
        FacultyEntity facultyEntity = facultyService.getFacultyById(facultyId);
        System.out.println(facultyEntity.getFacultyId());

        StudentEntity entity = new StudentEntity();
        if(null != param.get(key:"student-id")) {
            entity.setStudentId(Integer.parseInt(param.get(key:"student-id")));
        }
        entity.setStudentCode(param.get(key:"student-code"));
        entity.setStudentFirstName(param.get(key:"student-fname"));
        entity.setStudentLastName(param.get(key:"student-lname"));
        entity.setFaculty(facultyEntity);
        StudentEntity result = studentService.saveStudent(entity);
        System.out.println("Student ID: " + result.getStudentId());
        System.out.println("Student First Name: " + result.getStudentFirstName());
        System.out.println("Student Last Name: " + result.getStudentLastName());

        return "index";
    }
}
```
- Bottom Status Bar:** L0 0 △ 0 ⚡ 0 Java: Ready, Ln 68, Col 6, Spaces: 4, UTF-8, CRLF, {}, Java, Q.

Controller and Service, Student (Test)

The screenshot shows the Thunder Client interface with the following steps highlighted:

1. The URL field shows `localhost:8080/faculty/`.
2. The `Body` tab is selected.
3. The `Form` tab is selected.
4. The `faculty-name` field is populated with `Computer`.
5. The `Send` button is highlighted.
6. The terminal output shows the generated `Faculty ID: 1` and `Faculty Name: Computer`.

Activity pane (left):

- `GET faculty` 22 mins ago
- `GET course` 16 mins ago
- `GET student` 7 mins ago
- `POST faculty` just now (highlighted)
- `POST course` 16 mins ago
- `POST student` 5 mins ago

Status bar (bottom):

- `Java: Ready`

Controller and Service, Student (Test)

The screenshot shows the Thunder Client interface with the following steps highlighted:

1. Method: POST, URL: localhost:8080/student/
2. Body tab selected.
3. Form tab selected.
4. Form fields:
 - student-code: 68001
 - student-fname: Student First Name
 - student-lname: Student Last Name
 - faculty-id: 1
5. Send button.

The Response section shows the following output:

```
next value for student_seq
Hibernate:
    insert
    into
        student
        (faculty_id, student_code, student_frist_name, student_last_name, student_id)
    values
        (2, 2, 2, 2, 2)
Student ID: 1
Student First Name: Student First Name
Student Last Name: Student Last Name
```

A green checkmark icon is present next to the response text.

Bottom status bar: Java: Ready

Controller and Service, Student (Test)

The screenshot shows the H2 Console interface running on localhost:8080. The left sidebar lists database objects: COURSE, ENROLL, FACULTY, STUDENT (selected), INFORMATION_SCHEMA, Sequences, and Users. A red circle labeled '1' highlights the 'STUDENT' table. The main area contains a SQL statement: 'SELECT * FROM STUDENT;'. A red box surrounds the result table, which displays one row: Faculty ID 1, Student ID 1, Student Code 68001, Student First Name 'Student First Name', and Student Last Name 'Student Last Name'. A red circle labeled '2' is on the table, and another labeled '3' is on the 'Run' button. A green checkmark icon is positioned next to the result table.

localhost8080 x H2 Console x +

localhost:8080/h2-console/login.do?jsessionid=b5746e6d452b62b9456dbdba... ☆

Auto commit | Max rows: 1000 | Auto complete Off | Auto select On ?

jdbc:h2:file:/data/student

COURSE
ENROLL
FACULTY
STUDENT 1
INFORMATION_SCHEMA
Sequences
Users

H2 2.2.224 (2023-09-17)

Run Selected Auto complete Clear SQL statement:

SELECT * FROM STUDENT;

FACULTY_ID	STUDENT_ID	STUDENT_CODE	STUDENT_FIRST_NAME	STUDENT_LAST_NAME
1	1	68001	Student First Name	Student Last Name

(1 row, 1 ms)

Edit

Controller and Service, Course

File Edit Selection View ... ← → 🔍 student

EXPLORER STUDENT .mvn .vscode settings.json data src main java\com\workshop\stud... controller CourseController.java EnrollController.java FacultyController.java StudentController.java TutorialController.java entity repository service StudentApplication.java resources static templates OUTLINE TIMELINE JAVA PROJECTS MAVEN

workshop > student > controller > CourseController.java > Language Support for Java(TM) by Red Hat > CourseController

```
16
17  @Controller
18  @RequestMapping("/course")
19  public class CourseController {
20
21      @Autowired
22      private CourseService courseService; 1
23
24      @GetMapping("", "/")
25      public String getAll() {
26          System.out.println("----- CourseController getAll() -----");
27
28          List<CourseEntity> courses = courseService.getCourseAll();
29          System.out.println("----- CourseController getAll() Result -----");
30          System.out.println("Size: " + courses.size()); 2
31
32          return "index";
33      }
34
35      @GetMapping("/{course-id}")
36      public String getById(
37          @PathVariable(name = "course-id") Integer courseId
38      ) {
39          System.out.println("----- CourseController getById() -----");
40          System.out.println("course-id: " + courseId);
41
42          CourseEntity entity = courseService.getCourseById(courseId);
43          System.out.println("----- CourseController getById() Result -----");
```

Ln 34, Col 2 Spaces: 4 UTF-8 CRLF {} Java

Controller and Service, Course

The screenshot shows the VS Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Explorer:** Shows the project structure under STUDENT:
 - .mvn
 - .vscode
 - settings.json
 - data
 - src
 - main
 - java\com\workshop\stud...
 - controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Editor:** FacultyController.java, CourseController.java (active), course, course, course.
- Code:** CourseController.java content (lines 19-55). The code handles HTTP requests for courses. A red box highlights the section from line 42 to 62, which prints the course entity's name.

```
public class CourseController {  
    public String getAll() {  
        System.out.println("Size: " + courses.size());  
        return "index";  
    }  
  
    @GetMapping("/{course-id}")  
    public String getById(  
        @PathVariable(name = "course-id") Integer courseId  
    ) {  
        System.out.println("----- CourseController getById() -----");  
        System.out.println("course-id: " + courseId);  
  
        CourseEntity entity = courseService.getCourseById(courseId);  
        System.out.println("----- CourseController getById() Result -----");  
        System.out.println("Course Name: " + entity.getCourseName());  
  
        return "index";  
    }  
  
    @GetMapping("/delete/{course-id}")  
    public String getDeleteById(  
        @PathVariable(name = "course-id") Integer courseId  
    ) {  
        System.out.println("----- CourseController getDeleteById() -----");  
        System.out.println("course-id: " + courseId);  
    }  
}
```
- Bottom Status Bar:** Java: Ready, Ln 48, Col 1, Spaces: 4, UTF-8, CRLF, Java

Controller and Service, Course

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT:** .mvn, .vscode (settings.json), data, src (main, java\com\workshop\stud..., controller, CourseController.java, EnrollController.java, FacultyController.java, StudentController.java, TutorialController.java), entity, repository, service, StudentApplication.java, resources (static, templates), and a redacted section.
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN:** Standard project management sections.
- Central Area:** Code editor showing CourseController.java.

```
19  public class CourseController {  
36      public String getById(  
44          System.out.println("Course Name: " + entity.getCourseName());  
45  
46      return "index";  
47  }  
48  
49  @GetMapping("/delete/{course-id}")  
50  public String getDeleteById(  
51      @PathVariable(name = "course-id") Integer courseId  
52  ) {  
53      System.out.println(x:"---- CourseController getDeleteById() ----");  
54      System.out.println("course-id: " + courseId);  
55  
56      System.out.println(x:"---- CourseController getDeleteById() Result ----");  
57      courseService.deleteCourseById(courseId);  
58  
59      return "index";  
60  }  
61  
62  @PostMapping("/")  
63  public String postInsertAndUpdate(  
64      @RequestParam() Map<String, String> param  
65  ) {  
66      System.out.println(x:"---- CourseController postInsertAndUpdate() ----");  
67      System.out.println("course-id: " + param.get(key:"course-id"));  
68      System.out.println("course-name: " + param.get(key:"course-name"));  
69      System.out.println("course-desc: " + param.get(key:"course-desc"));
```
- Bottom Status Bar:** Ln 61, Col 6, Spaces: 4, UTF-8, CRLF, Java

Controller and Service, Course

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Back, Forward, Refresh, Home, Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT**: .mvn, .vscode (settings.json), data, src (main, java\com\workshop\stud..., controller, CourseController.java, EnrollController.java, FacultyController.java, StudentController.java, TutorialController.java), entity, repository, service, StudentApplication.java, resources (static, templates), and a folder named "CourseController".
 - OUTLINE**, **TIMELINE**, **JAVA PROJECTS**, **MAVEN**.
- Central Area (Code Editor):** CourseController.java (selected tab). The code implements a Controller for managing courses. It includes methods for deleting a course by ID, inserting/updating a course, and saving a course entity.
- Bottom Status Bar:** Labeled "Java: Ready". Other status indicators include file counts (0), Java version (0), and encoding (UTF-8).

```
public class CourseController {
    public String getDeleteById(
        @RequestParam("course-id") Integer id
    ) {
        return "index";
    }

    @PostMapping("/")
    public String postInsertAndUpdate(
        @RequestParam() Map<String, String> param
    ) {
        System.out.println("----- CourseController postInsertAndUpdate() -----");
        System.out.println("course-id: " + param.get(key:"course-id"));
        System.out.println("course-name: " + param.get(key:"course-name"));
        System.out.println("course-desc: " + param.get(key:"course-desc"));

        System.out.println("----- CourseController postInsertAndUpdate() Result -----");
        CourseEntity entity = new CourseEntity();
        if(null != param.get(key:"course-id")) {
            entity.setCourseId(Integer.parseInt(param.get(key:"course-id")));
        }
        entity.setCourseName(param.get(key:"course-name"));
        entity.setCourseDescription(param.get(key:"course-desc"));
        CourseEntity result = courseService.saveCourse(entity);
        System.out.println("Course ID: " + result.getId());
        System.out.println("Course Name: " + result.getName());
        return "index";
    }
}
```

TEST
COURSE CONTROLLER

Controller and Service, Enroll

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder. The "src/main/java/com/workshop/student/controller" package contains several controller classes: CourseController.java, FacultyController.java, StudentController.java, TutorialController.java, EnrollController.java (highlighted with a red box and circled '1'), and others.
- Code Editor (Right):** The file "EnrollController.java" is open. The code defines a controller class "EnrollController" with three private fields annotated with "@Autowired": "studentService", "courseService", and "enrollService".

```
22 @Controller
23 @RequestMapping("/enroll")
24 public class EnrollController {
25
26     @Autowired
27     private StudentService studentService;
28
29     @Autowired
30     private CourseService courseService;
31
32     @Autowired
33     private EnrollService enrollService;
34
35     @GetMapping("", "/")
36     public String getAll() {
37         System.out.println("----- EnrollController getAll() -----");
38
39         List<EnrollEntity> enrolls = enrollService.getEnrollAll();
40         System.out.println("----- EnrollController getAll() Result -----");
41         System.out.println("Size: " + enrolls.size());
42
43         return "index";
44     }
45
46     @GetMapping("/{enroll-id}")
47     public String getById(
48         @PathVariable(name = "enroll-id") Integer enrollId
49     ) {
```
- Annotations:**
 - A red box highlights the three autowired fields (lines 26-29).
 - A red box highlights the three lines of code that print the list size (lines 41-43).
 - A red circle with the number '1' is placed over the autowired fields.
 - A red circle with the number '2' is placed over the printed list size lines.
- Status Bar (Bottom):** Shows "Java: Ready" and other status indicators like file count (0), line count (0), and encoding (UTF-8).

Controller and Service, Enroll

Controller and Service, Enroll

The screenshot shows a Java application structure in the Explorer sidebar under the STUDENT project. The src/main/java/com/workshop/student/controller directory contains several controller classes: CourseController.java, EnrollController.java (highlighted with a red box), FacultyController.java, StudentController.java, and TutorialController.java. Below these are entity, repository, service, and StudentApplication.java files.

The EnrollController.java file contains the following code:

```
src > main > java > com > workshop > student > controller > EnrollController.java > Language Support for Java(TM) by Red Hat, Inc
```

```
24  ic class EnrollController {  
47  public String getById() {  
54      System.out.println("----- EnrollController getById() Result -----");  
55      System.out.println("Course Name: " + entity.getCourse().getCourseName());  
56      System.out.println("Student First Name: " + entity.getStudent().getStudentFirstName());  
57      System.out.println("Student Last Name: " + entity.getStudent().getStudentLastName());  
58  
59      return "index";  
60  }  
61  
62  @GetMapping("/delete/{enroll-id}")  
63  public String getDeleteById(@PathVariable(name = "enroll-id") Integer enrollId)  
64  {  
65      System.out.println("----- EnrollController getDeleteById() -----");  
66      System.out.println("enroll-id: " + enrollId);  
67  
68      System.out.println("----- EnrollController getDeleteById() Result -----");  
69      enrollService.deleteEnrollById(enrollId);  
70  
71      return "index";  
72  }  
73  
74  @PostMapping("/")  
75  public String postInsertAndUpdate(@RequestParam() Map<String, String> param)  
76  {  
77      System.out.println("----- EnrollController postInsertAndUpdate() -----");  
78      System.out.println("enroll-id: " + param.get("enroll-id"));  
79  
80  }
```

The code uses System.out.println statements to log the course name, student first name, student last name, and the enrollment ID being deleted. A red box highlights the line where the enrollment is deleted from the database: `enrollService.deleteEnrollById(enrollId);`. The status bar at the bottom indicates "Java: Ready".

Controller and Service, Enroll

The screenshot shows a Java application structure in the Explorer sidebar and the content of `EnrollController.java` in the main editor area.

Explorer Sidebar:

- STUDENT** folder:
 - .mvn
 - .vscode
 - settings.json
 - data
 - src
 - main
 - java\com\workshop...
 - controller
 - CourseController.java
 - EnrollController.java (highlighted with a red border)
 - FacultyController.java
 - StudentController.java
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - test
 - target
- OUTLINE
- TIMELINE
- JAVA PROJECTS
- MAVEN

Editor Area:

File Edit Selection View ... ⏪ ⏩ 🔍 student

src > main > java > com > workshop > student > controller > EnrollController.java > Language Support for Java(TM) by Red Hat

```
24  ic class EnrollController {  
25      public String postInsertAndUpdate(  
26          System.out.println(x:"----- EnrollController postInsertAndUpdate() -----");  
27          System.out.println("enroll-id: " + param.get(key:"enroll-id"));  
28          System.out.println("course-id: " + param.get(key:"course-id"));  
29          System.out.println("student-id: " + param.get(key:"student-id"));  
30  
31          System.out.println(x:"----- EnrollController postInsertAndUpdate() Result -----");  
32          Integer courseId = Integer.parseInt(param.get(key:"course-id"));  
33          CourseEntity courseEntity = courseService.getCourseById(courseId);  
34          System.out.println("Course ID:" + courseEntity.get courseId());  
35  
36          Integer studentId = Integer.parseInt(param.get(key:"student-id"));  
37          StudentEntity studentEntity = studentService.getStudentById(studentId);  
38          System.out.println("Student ID:" + studentEntity.getStudentId());  
39  
40          EnrollEntity entity = new EnrollEntity();  
41          if(null != param.get(key:"enroll-id")) {  
42              entity.setEnrollId(Integer.parseInt(param.get(key:"enroll-id")));  
43          }  
44          entity.setCourse(courseEntity);  
45          entity.setStudent(studentEntity);  
46          EnrollEntity result = enrollService.saveEnroll(entity);  
47          System.out.println("Enroll ID: " + result.getEnrollId());  
48          System.out.println("Course Name: " + result.getCourse().getCourseName());  
49          System.out.println("Student Code: " + result.getStudent().getStudentCode());  
50          return "index";  
51      }  
52  }
```

Ln 74, Col 6 Spaces: 4 UTF-8 CRLF {} Java

Controller and Service, Enroll (Test)

The screenshot shows the Thunder Client interface with the following steps highlighted:

1. URL: localhost:8080/faculty/
2. Body tab selected.
3. Form tab selected.
4. Faculty Name field: Computer
5. Send button.
6. Response terminal output showing the inserted faculty record.

Activity list (left sidebar):

- GET student (3 hours ago)
- GET course (2 hours ago)
- GET student (2 hours ago)
- GET enroll (just now)
- POST faculty (2 hours ago)
- POST course (2 hours ago)
- POST student (2 hours ago)
- POST enroll (just now)

EnrollController.java code (top right):

```
select next value for faculty_seq
Hibernate:
    insert
    into
        faculty
        (faculty_name, faculty_id)
    values
        (?, ?)
```

Response terminal (bottom right):

```
Faculty ID: 1
Faculty Name: Computer
```

Bottom status bar: Java: Ready

Controller and Service, Enroll (Test)

The screenshot shows the Thunder Client interface with a POST request to `localhost:8080/student/`. The request body is in Form format, containing four fields: `student-code` (value: 68001), `student-fname` (value: Student First Name), `student-lname` (value: Student Last Name), and `faculty-id` (value: 1). The response status is 200 OK, size is 293 Bytes, and time is 37 ms. The terminal shows the generated SQL insert statement.

- POST localhost:8080/student/
- Body
- Form
- student-code: 68001
student-fname: Student First Name
student-lname: Student Last Name
faculty-id: 1
- Send
- next value for student_seq
Hibernate:
insert
into
student
(faculty_id, student_code, student_frist_name, student_last_name, student_id)
values
(?, ?, ?, ?, ?)
Student ID: 1
Student First Name: Student First Name
Student Last Name: Student Last Name

Controller and Service, Enroll (Test)

The screenshot shows the Thunder Client interface for testing RESTful APIs. A POST request is being sent to `localhost:8080/course/`. The request body is set to `Form` and contains the following fields:

- `course-name`: Programming (checked)
- `course-id`: 1
- `field name`: value

The response status is `200 OK`, size is `293 Bytes`, and time is `27 ms`. The response body shows the generated Hibernate SQL query and the returned course details.

Numbered callouts point to specific elements:

- POST localhost:8080/course/ (Request URL)
- Body tab (Request type)
- Form tab (Body type)
- course-name: Programming (Form field value)
- Send button (Action button)
- Course ID: 1
Course Name: Programming (Response output)

Controller and Service, Enroll (Test)

The screenshot shows the Thunder Client interface for testing RESTful APIs. A POST request is being sent to `localhost:8080/enroll/`. The request body is a form containing `course-id` and `student-id`. The response shows a successful `200 OK` status with a size of `293 Bytes` and a time of `342 ms`. The response body displays a Hibernate SQL insert statement and the resulting enrollment details.

1. Method: POST
2. Body tab
3. Form tab
4. Form Fields:
 course-id
 student-id
5. Send button
6. Response body:

```
next value for enroll_seq
Hibernate:
    insert
    into
        enroll
        (course_id, student_id, enroll_id)
    values
        (?, ?, ?)
Enroll ID: 1
Course Name: Programming
Student Code: 68001
```

File Edit Selection View ... ← → ⌂ student THUNDER CLIENT EnrollController.java 1 faculty student course enroll New Request Activity Collections Env filter activity GET faculty 3 hours ago GET course 2 hours ago GET student 2 hours ago GET enroll just now POST faculty 2 hours ago POST course just now POST student 2 hours ago POST enroll just now PROBLEMS 1 OUTPUT PORTS DEBUG CONSOLE TERMINAL Maven-student + - × Response next value for enroll_seq Hibernate: insert into enroll (course_id, student_id, enroll_id) values (?, ?, ?) Enroll ID: 1 Course Name: Programming Student Code: 68001

Controller and Service, Enroll (Test)

The screenshot shows the Thunder Client interface for testing a REST API. The main area displays a GET request to `localhost:8080/enroll/`. The response status is `200 OK`, size is `293 Bytes`, and time is `29 ms`. The response body contains a SQL query and its results.

Request:

```
GET /enroll/
```

Response Headers:

```
Status: 200 OK
Size: 293 Bytes
Time: 29 ms
```

Response Body (SQL Query):

```
se1_0.student_frist_name,
se1_0.student_last_name
from
student se1_0
left join
faculty f1_0
on f1_0.faculty_id=se1_0.faculty_id
where
se1_0.student_id=?
```

Response Body (Result):

```
----- EnrollController getAll() Result -----
Size: 1
[]
```

The interface includes a sidebar with activity logs and a terminal tab at the bottom.

Controller and Service, Enroll (Test)

The screenshot shows the Thunder Client interface with the following details:

- Request URL:** GET localhost:8080/enroll/1 (highlighted with red box #1)
- Send Button:** A blue "Send" button with a red box #2.
- Activity Tab:** Active tab, showing a list of recent requests.
- Response Status:** Status: 200 OK, Size: 293 Bytes, Time: 85 ms.
- Response Content:**

```
student s1_0
    on s1_0.student_id=ee1_0.student_id
left join
    faculty f1_0
        on f1_0.faculty_id=s1_0.faculty_id
where
    ee1_0.enroll_id=?
```

----- EnrollController getById() Result -----
Course Name: Programming
Student First Name: Student First Name
Student Last Name: Student Last Name

- Terminal Tab:** Shows the SQL query used to retrieve the data.
- Bottom Status Bar:** Java: Ready.

Controller and Service, Enroll (Test)

The screenshot shows the H2 Console interface running on localhost:8080. The left sidebar displays the database schema with tables: COURSE, ENROLL, STUDENT, and FACULTY. The ENROLL table is selected. The main area contains a SQL query and its execution results.

SQL Query:

```
SELECT * FROM ENROLL
LEFT JOIN STUDENT
ON ENROLL.STUDENT_ID = STUDENT.STUDENT_ID
LEFT JOIN COURSE
ON ENROLL.COURSE_ID = ENROLL.COURSE_ID
```

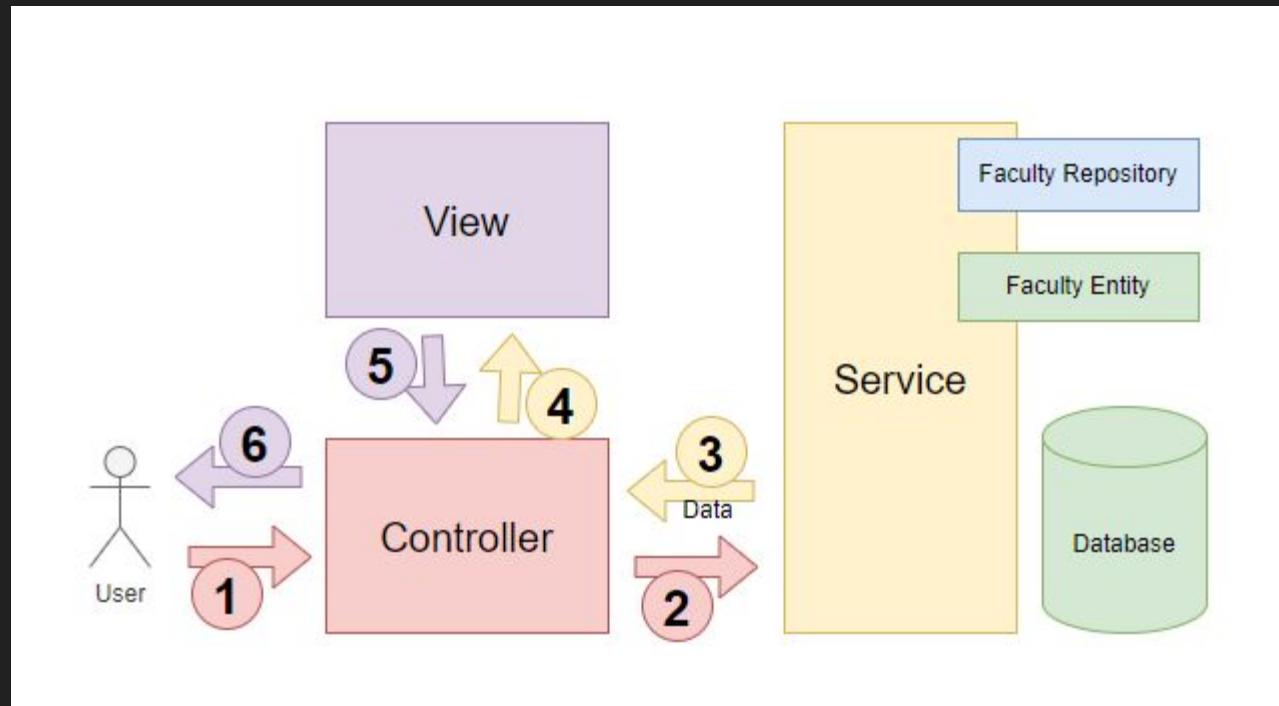
Execution Buttons: Run (highlighted with red box and circled 1), Run Selected, Auto complete, Clear, SQL stats.

Result Table:

	STUDENT_ID	STUDENT_CODE	STUDENT_FIRST_NAME	STUDENT_LAST_NAME	COURSE_ID	COURSE_DESCRIPTION	COURSE_NAME
	1	68001	Student First Name	Student Last Name	1	null	Programming

View (Day 2)

View



SSR VS CSR

<https://www.geeksforgeeks.org/server-side-rendering-vs-client-side-rendering-vs-server-side-generation/>

Server-Side Rendering

Server-side rendering (SSR) is the process of rendering web pages on the server and sending the fully-rendered HTML to the client. In this approach, the server generates the HTML, including any dynamic data, and sends it to the client as a complete page. The client then displays the page without any further processing.

One example of a popular SSR framework is Next.js. With Next.js, you can write React code and have it automatically rendered on the server, providing the benefits of SSR without having to manage the server yourself.

Advantages:

- Faster initial load times
- Improved SEO optimization
- Can provide a better user experience for users with slower internet connections or less powerful devices

Disadvantages:

- Can require more server resources and maintenance
- This can result in slower subsequent page loads if the client needs to make additional server requests

Working of SSR: When a user requests a page, the server generates the HTML for that page, including any dynamic data. The fully-rendered HTML is then sent to the client, which can display the page without any further processing.

Uses: SSR is commonly used for content-heavy websites, such as blogs or news websites, where fast initial load times and good SEO optimization are important.

blognone.com

JETBRAINS IDEs
IntelliJ IDEA

For productive
Spring development

Features Interview Forum Jobs Workplace Company Profile Search

NVIDIA ร่วมมือ Hugging Face เปิดใช้งาน NVIDIA DGX รับนิเทศ ตัวค้นหาภาษาจีน

By: law | 21:40 on 30 July 2024 | Tags: NVIDIA, Hugging Face, Artificial Intelligence

NVIDIA ประกาศความร่วมมือกับ Hugging Face นำบริการ NVIDIA DGX Cloud รุ่น H100 ออก มาใช้ในการฝึกอบรมโมเดลภาษาจีนเพื่อเพิ่มประสิทธิภาพและลดเวลาฝึกอบรมลงครึ่ง

ผู้ใช้ที่ต้องการใช้บริการที่ดังกล่าวสามารถสมัคร Hugging Face แบบ Enterprise (เดือนละ 20,000 ชั่วโมง) และทดลองใช้ฟรี "NVIDIA NIM Enterprise" โดยเขียนภาษา Python ได้โดยตรงในภาษา Python ไม่ต้องติดตั้ง Hugging Face เนื่องจาก H100 ที่มาพร้อมฟีเจอร์นี้จะสามารถทำงานได้โดยอัตโนมัติ

Read more

AMD ออกเทคโนโลยีใหม่ชื่อ Fluid Motion Frames 2 และ Latency ของเฟรมแลนด์ได้เร็ว

By: mks | 21:18 on 30 July 2024 | Tags: AMD, Radeon, Graphic, GPU

AMD ได้เปิดตัวเทคโนโลยีใหม่ชื่อ Fluid Motion Frames (AMF) (เบื้องต้นและอย่างทัน FSR ที่มีผลลัพธ์ที่ดีกว่า FSR 2) และ Latency AMF 2 ที่ลดความล่าช้าของเฟรมแลนด์ได้ถึง 28% (ทดสอบกับ Cyberpunk 2077 ความละเอียด 4K) และสามารถใช้ร่วมกับเทคโนโลยี Anti-Lag 2 ของซีพียู Radeon ได้เร็ว

Read more

บล็อก Ad Blocker บน YouTube ทำให้เราอ่านได้ร้าวๆ กันเป็นปีมีนาคม ก็ต้องขออภัย

By: mks | 21:00 on 30 July 2024 | Tags: YouTube, Ad Blocking, Advertising

คุณสามารถตั้งค่า YouTube ของคุณ ad blocker ที่ไม่ต้องลงแอปฯ การตั้งค่าใหม่อาจลากเส้นสีฟ้าไปที่ตัวเลือก YouTube Origin เมื่อคุณต้องการที่จะติดต่อผู้ให้บริการของคุณ 6-20 วินาที คุณต้องตั้งค่าตัวเลือก YouTube สำหรับ กรณีที่คุณต้องการตัดเสียงได้ดีกว่า ต้องตรวจสอบว่าจะต้อง

โฆษณา ฉุกเฉียบด้วย YouTube ไปหมด

หากคุณต้องการตัดเสียงได้ดีกว่า YouTube ให้ตั้งค่าตัวเลือก YouTube สำหรับ กรณีที่คุณต้องการตัดเสียงได้ดีกว่า ต้องตรวจสอบว่าจะต้อง

Read more

sign in

สมัครสมาชิก ลืมรหัสผ่าน

Username: Password: Log in

Blognone Jobs Premium

Product Management-Cross Border Payments
Bangkok Bank - บางรัก^{กรุงเทพมหานคร}

Senior Blockchain Developer (SCB 10X Lab)
SCB 10X - คลองเตย^{กรุงเทพมหานคร}

เจ้าหน้าที่ ส่วนติดตามระบบดิจิทัล
ศรีสะเกษเมือง^{ไทยเครดิต จำกัด (TCG)}

droidsans.com

SIGN IN

DROIDSAN

REVIEW EDITORIAL TIPS AUTOMOBILE

10 มือถือ Android ที่แรงที่สุด

NEWS 10 อันดับมือถือ Android ที่แรงที่สุด ประจำปี 2024 อันดับ 1 ยังเป็น ROG Phone 8 Pro

บทความน่าอ่าน

เลือกซื้อรถมือสอง ให้ปลอดภัย

ล้ำหน้าปี 2024

LATEST NEWS

NEWS เปิดตัว Snapdragon 4s Gen 2 ชิปรุ่นเล็ก สำหรับมือถือ 5G ราคาสบายกระเป๋า

Thanapoom July 30, 2024

Share Tweet Like

JETBRAINS IDEs | IntelliJ IDEA

Profile With Ease

FXGT.com

Client-Side Rendering

Client-side rendering (CSR) is the process of rendering web pages on the client using JavaScript. In this approach, the server sends the initial HTML file, but the client then uses JavaScript to dynamically update the page as needed. This allows for more interactive and responsive web pages, as the client can update specific parts of the page without needing to reload the entire page.

One example of a popular CSR framework is React. With React, you can write JavaScript code that updates the DOM as needed, providing a more interactive and dynamic web application.

Advantages:

- More dynamic and interactive web applications
- Can provide a smoother and more seamless user experience
- Can reduce the need for additional server requests

Disadvantages:

- Slower initial load times
- Can be less SEO-friendly, as search engines may have difficulty indexing client-rendered content

Working of CSR: When a user requests a page, the server sends the initial HTML file, along with any required JavaScript files. The client then uses JavaScript to update the page as needed, without needing to reload the entire page.

Uses: CSR is commonly used for web applications that require a high degree of interactivity, such as social media platforms or e-commerce websites.

facebook.com

facebook

Facebook helps you connect and share with the people in your life.

[Forgotten password?](#)

[Create a Page](#) for a celebrity, brand or business.

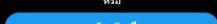
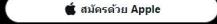
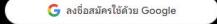
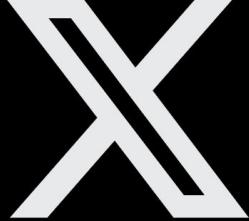
English (UK) ภาษาไทย 日本語 中文(简体) Tiếng Việt Français (France) Deutsch Русский Español Português (Brasil) Italiano +
Sign Up Log in Messenger Facebook Lite Video Places Games Marketplace Meta Pay Meta Store Meta Quest Meta AI Instagram Threads
Fundraisers Services Voting Information Centre Privacy Policy Privacy Centre Groups About Create ad Create Page Developers Careers Cookies
AdChoices ▾ Terms Help Contact uploading and non-users

Meta ©2024

x.com/?lang=th

เกิดขึ้นตอนนี้

เข้าร่วมวันนี้



การยืนยันตัวตนของคุณ ช่วยให้เราสามารถจัดการเรื่องราวและฟีดของคุณได้ดียิ่งขึ้น

มีบัญชีแล้ว? ใช่ ล็อกอิน

เข้าสู่ระบบ

ยังไม่มีบัญชี? สมัครฟรี

สมัครฟรี

Tailwind CSS



Rapidly build modern websites without ever leaving your HTML.

A utility-first CSS framework packed with classes like `flex`, `pt-4`, `text-center` and `rotate-90` that can be composed to build any design, directly in your markup.

[Get started](#) Quick search...

Ctrl K

Use Tailwind CSS

The screenshot shows the homepage of tailwindcss.com. At the top, there's a navigation bar with icons for home, search, and user profile, followed by the URL 'tailwindcss.com'. Below the navigation is a secondary header with the text 'windcss' on the left and 'Docs', 'Components', and 'Blog' on the right. The main title 'tailwindcss.com' is displayed in large red letters. Below it is a large, bold, white text block: 'Rapidly build modern websites without ever leaving your HTML.' A descriptive subtitle follows, mentioning 'A utility-first CSS framework packed with classes like `flex`, `pt-4`, `text-center` and `rotate-90`' that can be composed to build any design, directly in your markup. To the left of the subtitle is a blue 'Get started' button with a red border. To the right is a search bar with placeholder text 'Quick search...' and a keyboard shortcut 'Ctrl K'. In the bottom left corner, there's a dark overlay featuring a portrait of a woman with glasses and a quote: 'Tailwind CSS is the only framework that I've seen scale on large teams. It's easy to customize, adapts to any screen size, and the build size is tiny.' On the right side, a code editor window shows a snippet of Tailwind CSS code:

```
1 <figure class="bg-slate-100 rounded-xl p-8 dark:bg-slate-800">
2   
3   <div class="pt-6 space-y-4">
4     <blockquote>
5       <p class="text-lg font-medium">
6         Tailwind CSS is the only framework that I've seen scale on
7         large teams. It's easy to customize, adapts to any
8         screen size, and the build size is tiny.
9     </p>
```

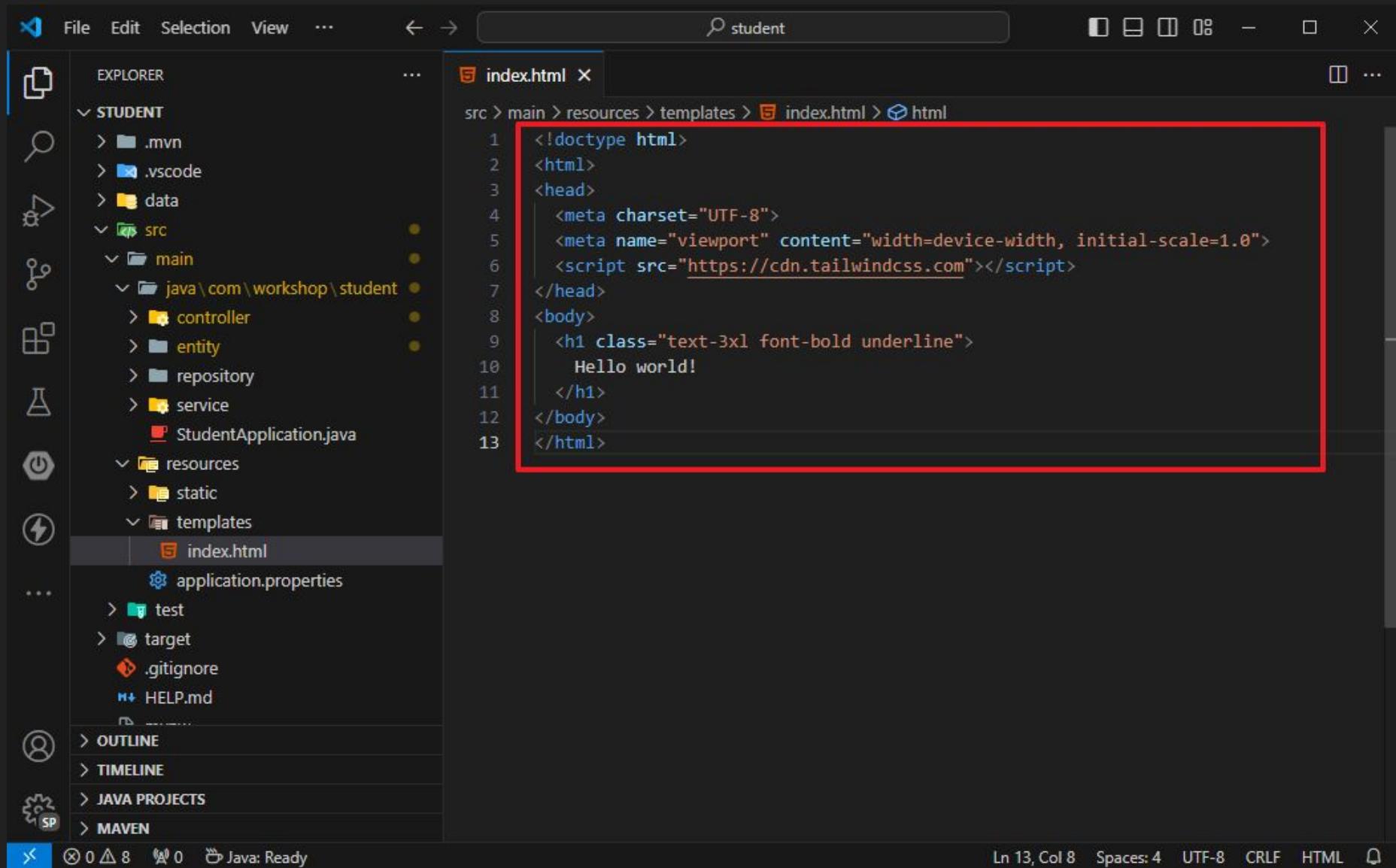
Use Tailwind CSS

The screenshot shows the Tailwind CSS documentation website at tailwindcss.com/docs/installation/play-cdn. The page title is "Get started with Tailwind CSS". The navigation bar includes links for Documentation, Components, Blog, and Showcase. On the left sidebar, under "Getting Started", the "Installation" section is currently selected. The main content area has a heading "Installation" and a paragraph explaining how Tailwind CSS works by scanning files for class names. Below this is a callout box with a blue border and white text, containing the text: "Use the Play CDN to try Tailwind right in the browser without any build step. The Play CDN is designed for development purposes only, and is not the best choice for production." A red box highlights the "Play CDN" button, which is circled with a red number "1". To the right of the callout is a blue exclamation mark icon. In the bottom right corner of the page, there is a code editor window showing an "index.html" file with the following content:

```
<!doctype html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body>
  <h1 class="text-3xl font-bold underline">
    Hello world!
  </h1>
</body>
```

A red box highlights the "index.html" tab, and another red box highlights the code editor interface, which is circled with a red number "2".

Use Tailwind CSS

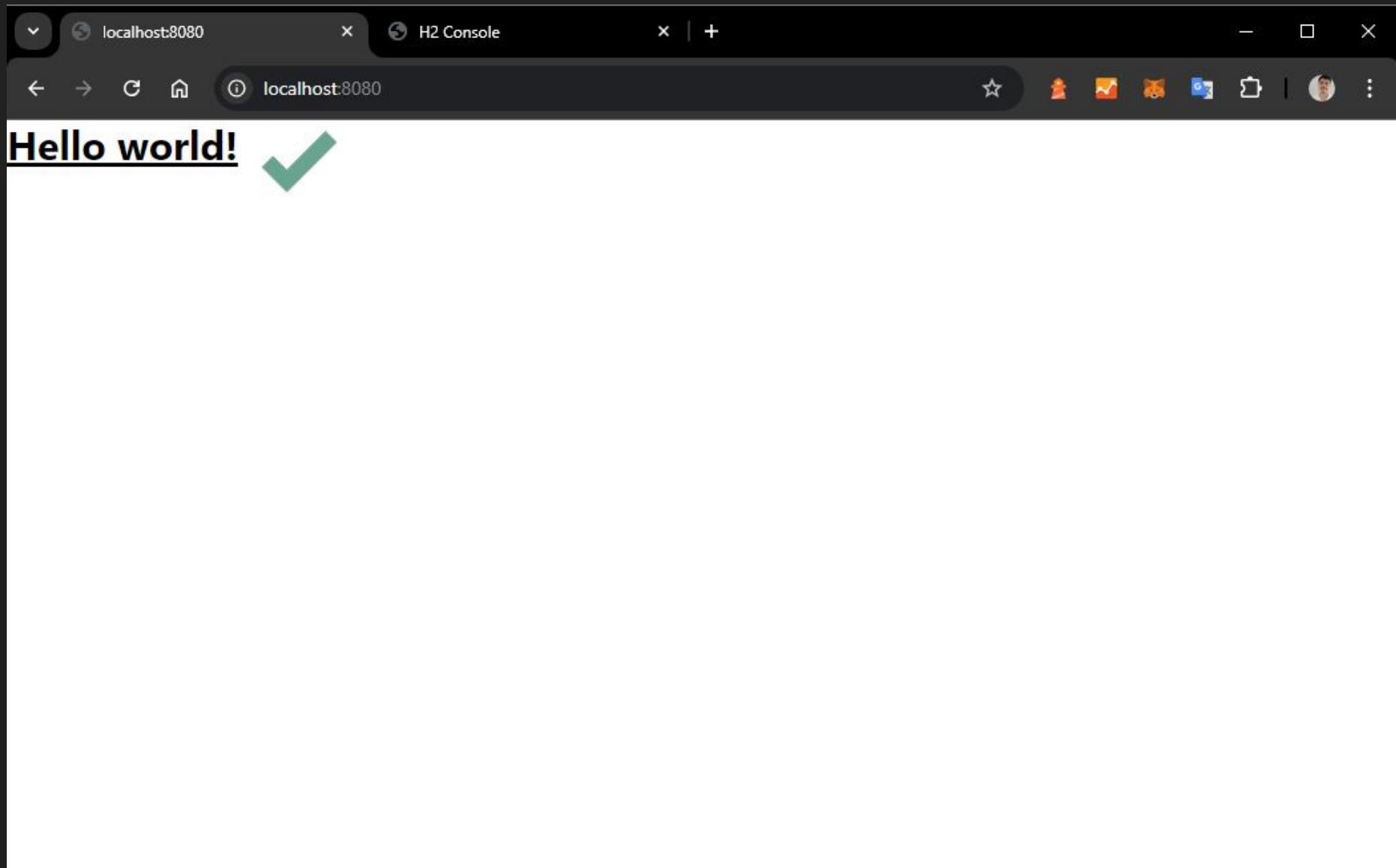


The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left displays a project structure for a Java application named 'STUDENT'. The 'templates' folder contains an 'index.html' file, which is currently open in the editor. The code in 'index.html' is as follows:

```
<!doctype html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body>
  <h1 class="text-3xl font-bold underline">
    Hello world!
  </h1>
</body>
</html>
```

A red box highlights the entire code block in the editor. The status bar at the bottom shows 'Ln 13, Col 8' and other details like 'Spaces: 4', 'UTF-8', 'CRLF', 'HTML', and a file icon.

Use Tailwind CSS



Thymeleaf

Spring Initializr

spring initializr

Web, Security, JPA, Actuator, Devtools...

Press Ctrl for multiple adds

WEB

Jersey
Framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs.

Vaadin
The full-stack web app platform for Spring. Build views fully in Java with Flow, or in React using Hilla.

Netflix DGS
Build GraphQL applications with Netflix DGS and Spring for GraphQL.

TEMPLATE ENGINES

Thymeleaf
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

Apache Freemarker
Java library to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data.

Mustache
Logic-less templates for both web and standalone environments. There are no if statements, else clauses, or for loops. Instead there are only tags.

Groovy Templates
Groovy templating engine.

SECURITY

Spring Security
Highly customizable authentication and access-control framework for Spring applications.

ADD DEPENDENCIES... CTRL + B

Project

Gradle - Groovy Gradle - Kotlin
 Maven

Spring Boot

3.4.0 (SNAPSHOT) 3.4.0 (M1)
 3.2.9 (SNAPSHOT) 3.2.8

Project Metadata

Group: com.example

Artifact: demo

Name: hello-spring

Description: Demo project for Spring Boot

Package name: com.example.demo

Packaging: Jar War

Java: 22 21 17



Thymeleaf

30 July 2023: **Thymeleaf 3.1.2.RELEASE** has been published.

See [what's new in Thymeleaf 3.1](#) and [how to migrate](#).

Thymeleaf is a modern server-side Java template engine for both web and standalone environments.

Thymeleaf's main goal is to bring elegant *natural templates* to your development workflow — HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams.

With modules for Spring Framework, a host of integrations with your favourite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development — although there is much more it can do.

<https://www.thymeleaf.org/>

Thymeleaf - Natural template

Natural templates

HTML templates written in Thymeleaf still look and work like HTML, letting the actual templates that are run in your application keep working as useful design artifacts.

```
1 <table>
2   <thead>
3     <tr>
4       <th th:text="#{msgs.headers.name}">Name</th>
5       <th th:text="#{msgs.headers.price}">Price</th>
6     </tr>
7   </thead>
8   <tbody>
9     <tr th:each="prod: ${allProducts}">
10      <td th:text="${prod.name}">Oranges</td>
11      <td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
12    </tr>
13  </tbody>
14 </table>
```

Thymeleaf - Standard Dialects

1. Standard dialects?

Thymeleaf is very, very extensible, and it allows you to define your own sets of template attributes (or even tags) with the names you want, evaluating the expressions you want in the syntax you want and applying the logic you want. It's more like a *template engine framework*.

Out of the box, nevertheless, it comes with something called *the standard dialects* (named *Standard* and *SpringStandard*) that define a set of features which should be more than enough for most scenarios. You can identify when these standard dialects are being used in a template because it will contain attributes starting with the `th` prefix, like `>`.

Note that the *Standard* and the *SpringStandard* dialects are almost identical, except that *SpringStandard* includes specific features for integrating into Spring MVC applications (like, for example, using *Spring Expression Language* for expression evaluation instead of *OGNL*).

Also note we usually refer to features in the Standard dialects when we talk about Thymeleaf without being more specific.

Thymeleaf - Natural template

Natural templates

HTML templates written in Thymeleaf still look and work like HTML, letting the actual templates that are run in your application keep working as useful design artifacts.

```
1 <table>
2   <thead>
3     <tr>
4       <th th:text="#{msgs.headers.name}">Name</th>
5       <th th:text="#{msgs.headers.price}">Price</th>
6     </tr>
7   </thead>
8   <tbody>
9     <tr th:each="prod: ${allProducts}">
10      <td th:text="${prod.name}">Oranges</td>
11      <td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
12    </tr>
13  </tbody>
14 </table>
```

Thymeleaf - Standard Expression

2. Standard Expression syntax

Most Thymeleaf attributes allow their values to be set as or containing *expressions*, which we will call *Standard Expressions* because of the dialects they are used in. These can be of five types:

- \${...} : Variable expressions.
- *{...} : Selection expressions.
- #{...} : Message (i18n) expressions.
- @{...} : Link (URL) expressions.
- ~{...} : Fragment expressions.

ThymeLeaf - Template / fragment / replace / insert

th:block / th:fragment <

th:replace / th:insert <

th:href + @ { ... } < <https://www.thymeleaf.org/doc/articles/standardurlsyntax.html>

Thymeleaf - Template

Student Management Faculty Student Course Enroll Header

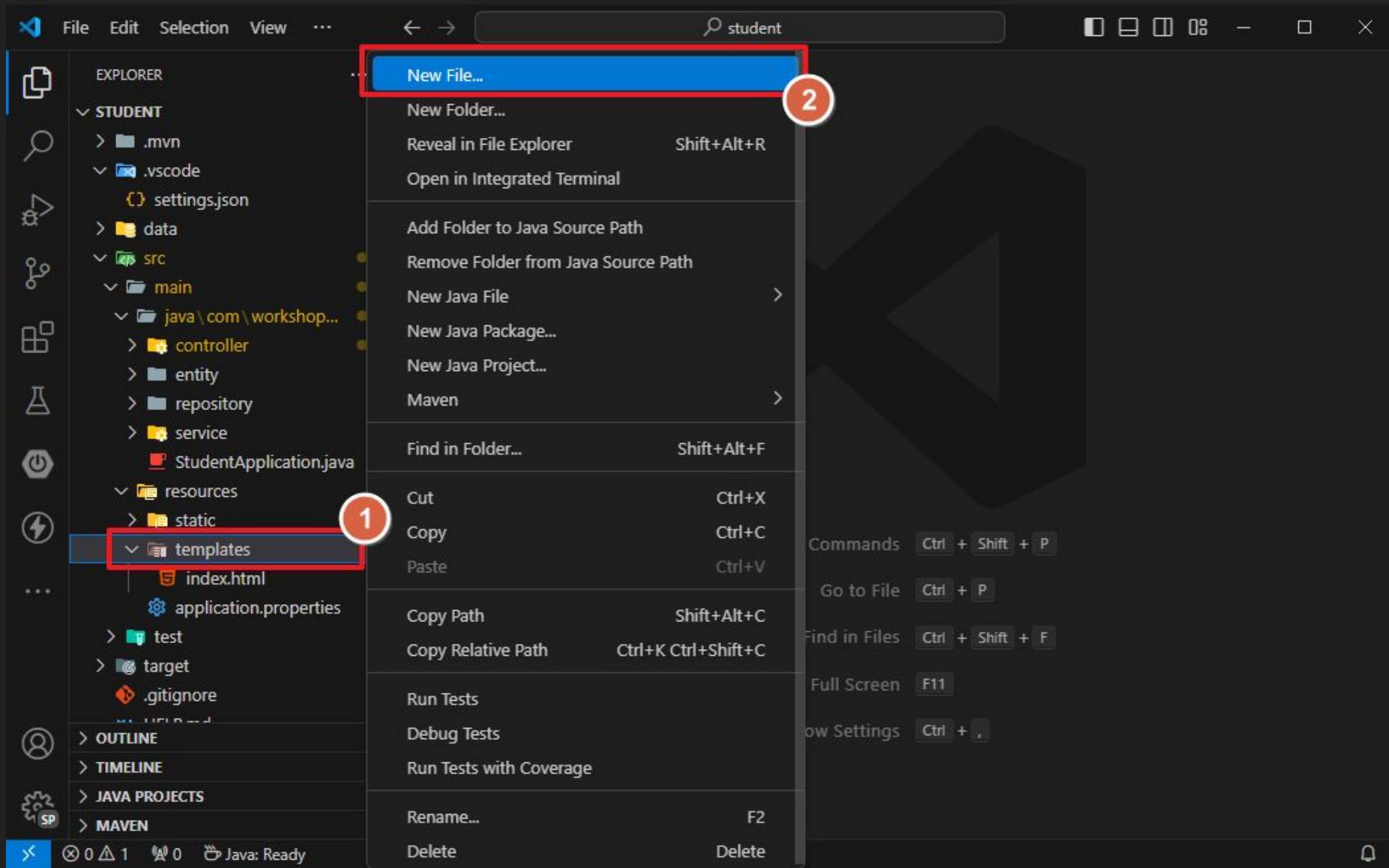
Faculty Id

Faculty Name

Delete Save

ID	Name	Action
1	Computer Science	[View]
2	Multimedia Technology	[View]
3	Software Engineer	[View]
4	Computer Engineer	[View]

Thymeleaf - Template



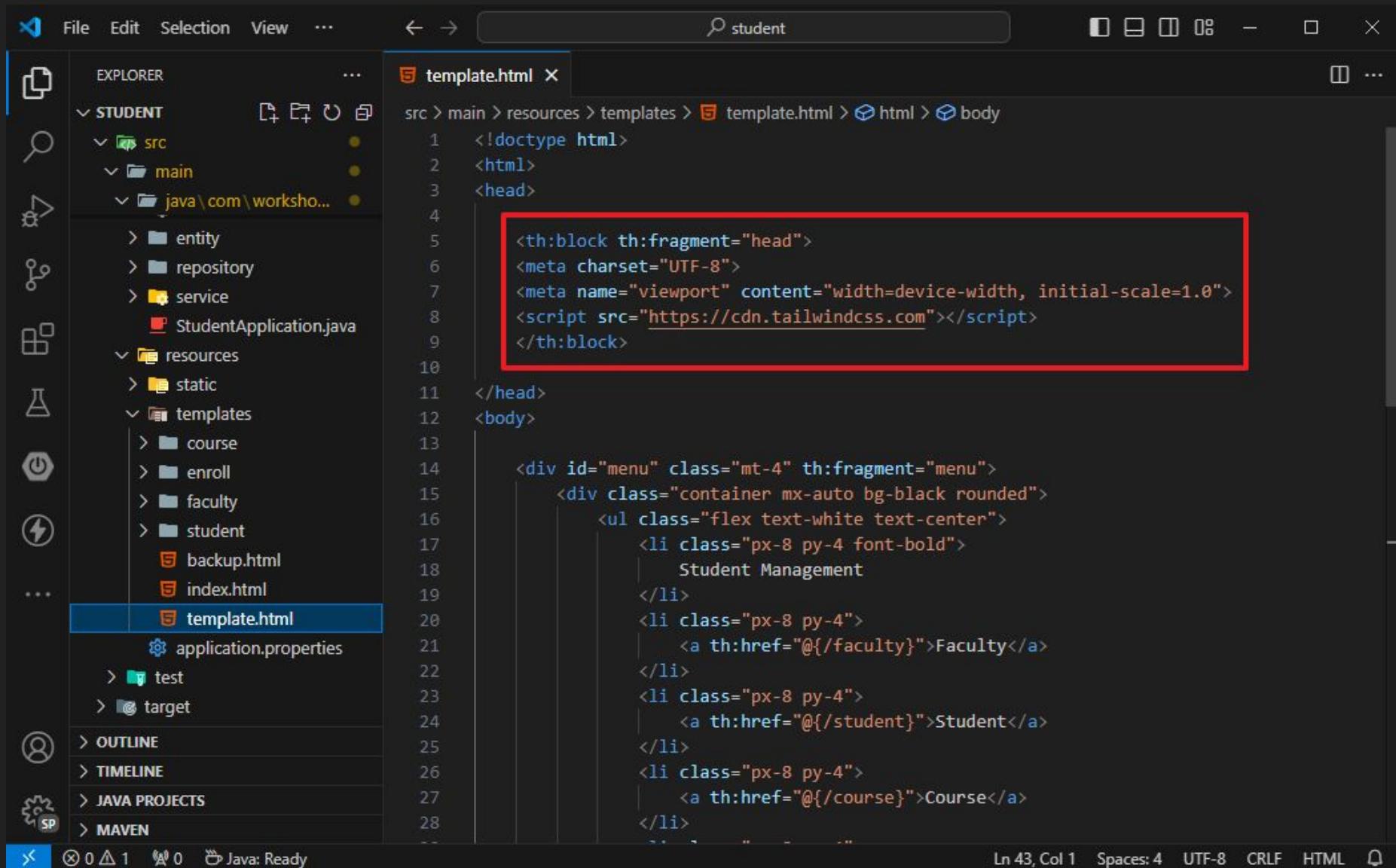
Thymeleaf - Template

The screenshot shows a Java IDE interface with a dark theme. The left sidebar contains icons for file operations, search, and project navigation. The main area displays a file named `template.html` in the `src/main/resources/templates` directory. The code in the editor is:

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body>
</body>
</html>
```

A red box highlights the entire code block in the editor. The file tab for `template.html` is also highlighted with a red box. The status bar at the bottom shows "Java: Ready".

Thymeleaf - Template th:block th:fragment



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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT**: src, main, java\com\worksho...
 - entity, repository, service, StudentApplication.java
 - resources**: static, templates
 - course, enroll, faculty, student, backup.html, index.html
 - template.html** (selected)
 - application.properties
 - test, target
- Right Main Area (template.html):**

```
src > main > resources > templates > template.html > html > body
1   <!doctype html>
2   <html>
3   <head>
4
5   <th:block th:fragment="head">
6   <meta charset="UTF-8">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <script src="https://cdn.tailwindcss.com"></script>
9   </th:block>
10
11  </head>
12  <body>
13
14  <div id="menu" class="mt-4" th:fragment="menu">
15      <div class="container mx-auto bg-black rounded">
16          <ul class="flex text-white text-center">
17              <li class="px-8 py-4 font-bold">
18                  Student Management
19              </li>
20              <li class="px-8 py-4">
21                  <a th:href="@{/faculty}">Faculty</a>
22              </li>
23              <li class="px-8 py-4">
24                  <a th:href="@{/student}">Student</a>
25              </li>
26              <li class="px-8 py-4">
27                  <a th:href="@{/course}">Course</a>
28              </li>
```

A red box highlights the `<th:block th:fragment="head">` block.
- Bottom Status Bar:** Ln 43, Col 1, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template th:block th:fragment

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Left Sidebar (EXPLORER):** Shows the project structure under STUDENT:
 - src
 - main
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - student
 - backup.html
 - index.html
 - template.html (selected)
 - application.properties
 - test
 - target- Outline Bar:** OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN.
- Code Editor:** The template.html file is open in the editor. A red box highlights the following code block:

```
<div id="menu" class="mt-4" th:fragment="menu">
    <div class="container mx-auto bg-black rounded">
        <ul class="flex text-white text-center">
            <li class="px-8 py-4 font-bold">
                Student Management
            </li>
            <li class="px-8 py-4">
                <a th:href="@{/faculty}">Faculty</a>
            </li>
            <li class="px-8 py-4">
                <a th:href="@{/student}">Student</a>
            </li>
            <li class="px-8 py-4">
                <a th:href="@{/course}">Course</a>
            </li>
            <li class="px-8 py-4">
                <a th:href="@{/enroll}">Enroll</a>
            </li>
        </ul>
    </div>
</div>
```
- Bottom Status Bar:** Ln 43, Col 1, Spaces: 4, UTF-8, CRLF, HTML.

Thymeleaf - Template th:block th:fragment

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Back, Forward, Home, Stop, Refresh, Minimize, Maximize, Close.
- Left Sidebar (EXPLORER):** STUDENT folder containing src, main, and resources. Under main, there is a java directory with com/workshop... and subfolders entity, repository, service, and StudentApplication.java. Under resources, there are static and templates folders with course, enroll, faculty, student, backup.html, index.html, and template.html. application.properties and test files are also listed.
- Right Panel (Code Editor):** The template.html file is open. The code is as follows:

```
<html>
<body>
    <div id="menu" class="mt-4 th:fragment="menu">
        <div class="container mx-auto bg-black rounded">
            <ul class="flex text-white text-center">
                <a th:href="@{/course}">Course</a>
            </li>
            <li class="px-8 py-4">
                <a th:href="@{/enroll}">Enroll</a>
            </li>
        </ul>
    </div>
</div>

<th:block th:fragment="content-begin">
<div id="content" class="container mx-auto p-4">
</th:block>
<!-- Content --> Content
<th:block th:fragment="content-end">
</div>
</th:block>

</body>
</html>
```

A red box highlights the block from line 36 to line 75, which contains the content placeholder and the two th:block th:fragment blocks.

Bottom Status Bar: Ln 43, Col 1 Spaces: 4 UTF-8 CRLF HTML

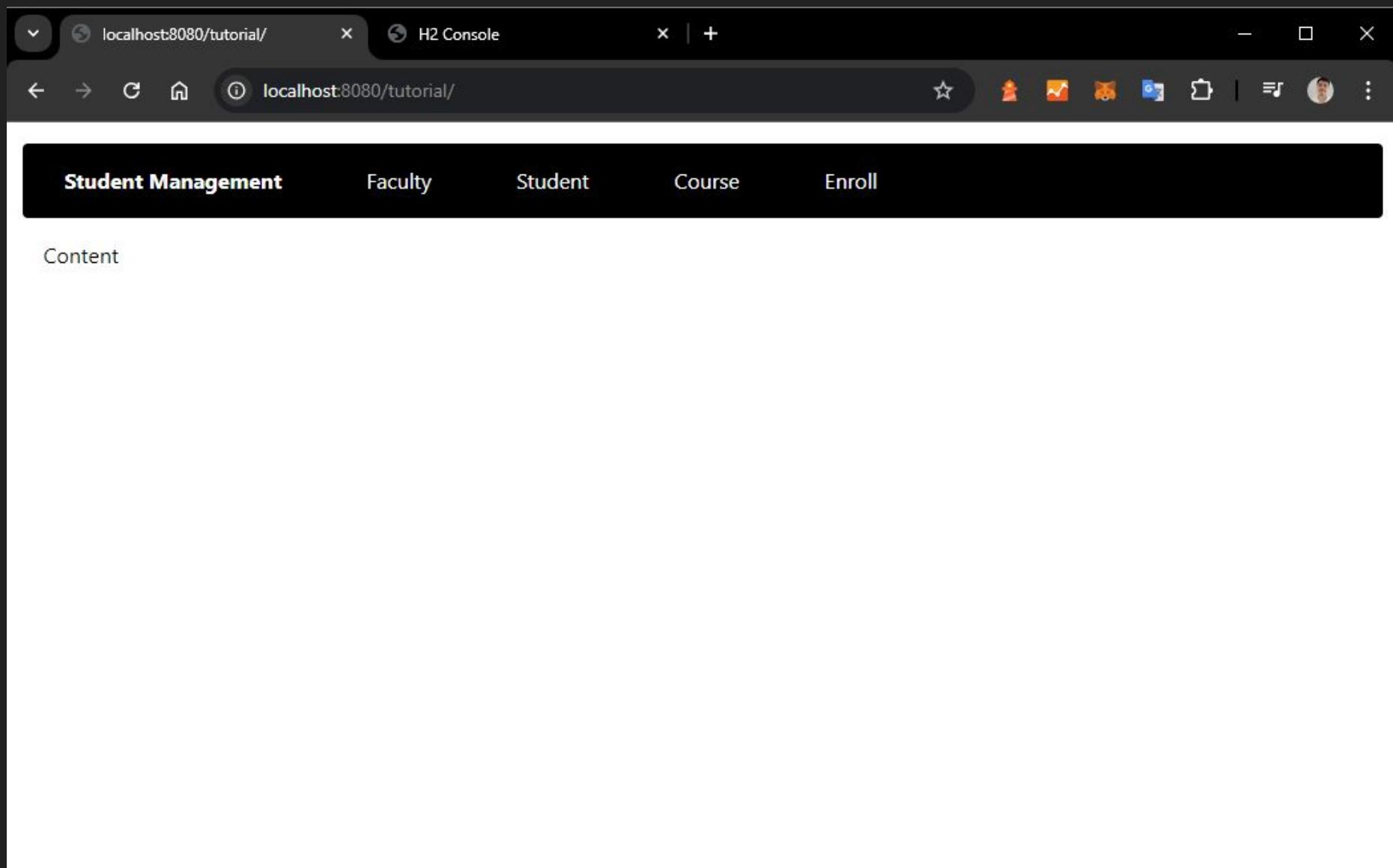
Thymeleaf - Template / Test Template

The screenshot shows a Java application structure in the Explorer sidebar. The `TutorialController.java` file is selected and highlighted with a red border. The code implements a `@Controller` for the `/tutorial` endpoint. It contains two methods: `getTutorial` and `getTutorialPath`. Both methods print the ID of the tutorial and return a template string. The `getTutorial` method also prints a message indicating it's being called.

```
12  @Controller
13  @RequestMapping("/tutorial")
14  public class TutorialController {
15
16      @GetMapping("/")
17      public String getTutorial(
18          @RequestParam(name = "id", required = false, defaultValue = "0") Integer id
19      ) {
20          System.out.println("---- getTutorial ----");
21          System.out.println("ID: " + id);
22          return "template";
23      }
24
25
26      @GetMapping("/{id}")
27      public String getTutorialPath(
28          @PathVariable(name = "id") Integer id
29      ) {
30          System.out.println("---- getTutorialPath ----");
31          System.out.println("ID: " + id);
32          return "index";
33      }
34
35      @PostMapping("/")
36      public String postTutorial(
37          @RequestParam() Map<String, String> param
38      ) {
39          System.out.println("---- postTutorial ----");
40      }
41  }
```

The status bar at the bottom indicates the code is at line 25, column 1, with 4 spaces per tab, in UTF-8 encoding, and using CRLF line endings. The Java language is selected.

Thymeleaf - Template / Test Template



Thymeleaf - Template Faculty index.html

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard window control buttons.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\worksho... (highlighted)
 - service (highlighted)
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - index.html
 - enroll
 - index.html
 - Bottom Left Buttons:** + (New), ⚡ (Run), ...
 - Bottom Left Icons:** OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN.
 - Bottom Status Bar:** Ln 13, Col 8 Spaces: 4 UTF-8 CRLF HTML

The main editor area displays the file `index.html` with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <th:block th:replace="~{template::head}"></th:block>
</head>
<body>
    <th:block th:replace="~{template::menu}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Faculty
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
</html>
```

The code block and the file in the Explorer view are both highlighted with a red border.

Thymeleaf - Template Student index.html

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Left Sidebar (EXPLORER):** Shows the project structure under STUDENT:
 - src
 - main
 - java\com\worksho... (with service and StudentApplication.java)
 - resources
 - static
 - templates
 - course (index.html)
 - enroll (index.html)
 - faculty (index.html)
 - student (selected, highlighted with a red box)
 - index.html (highlighted with a red box)
 - backup.html
 - index.html
 - template.html
- Central Editor:** The content of index.html is displayed:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <th:block th:replace="~{template::head}"></th:block>
</head>
<body>
    <th:block th:replace="~{template::menu}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Student
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
</html>
```
- Bottom Status Bar:** Ln 7, Col 43, Spaces: 4, UTF-8, CRLF, HTML, SP

Thymeleaf - Template Course index.html

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\worksho...
 - service (StudentApplication.java)
 - resources
 - static
 - templates
 - Course Template:** course/index.html (highlighted with a red box).
 - enroll/index.html
 - faculty/index.html
 - student/index.html
 - student/backup.html
 - student/index.html
 - student/template.html
 - application.properties
- Central Area:** The code editor displays the content of `index.html`.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <th:block th:replace="~{template::head}"></th:block>
</head>
<body>
    <th:block th:replace="~{template::menu}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Course
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
</html>
```
- Bottom Status Bar:** Ln 6, Col 7, Spaces: 4, UTF-8, CRLF, HTML, SP

Thymeleaf - Template Enroll index.html

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Back, Forward, Home, Stop, Refresh, Minimize, Maximize, Close.
- Explorer:** Shows the project structure under STUDENT:
 - src
 - main
 - java\com\workshop...
 - service (containing StudentApplication.java)
 - resources
 - static
 - templates
 - course (containing index.html)
 - enroll (highlighted with a red box)
 - index.html (highlighted with a red box)
 - faculty (containing index.html)
 - student (containing index.html, backup.html, index.html, template.html)
- Code Editor:** The file index.html is open, showing Thymeleaf syntax. A red box highlights the entire code content.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <th:block th:replace="~{template::head}"></th:block>
</head>
<body>
    <th:block th:replace="~{template::menu}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Enroll
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
</html>
```
- Bottom Status Bar:** Ln 7, Col 43, Spaces: 4, UTF-8, CRLF, HTML, SP

Thymeleaf - Template Faculty Controller

The screenshot shows an IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Explorer:** Shows the project structure under STUDENT.
 - src
 - main
 - java\com\workshop\student\controller (highlighted with a red box)
 - resources
 - static
 - templates
 - course (highlighted with a red box)
 - index.html
 - enroll
 - index.html
 - faculty
 - index.html
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - target
 - FacultyController.java:** The code defines a controller for managing faculty entities. It includes methods for getting all faculty, getting a faculty by ID, and deleting a faculty by ID. The code uses Thymeleaf to render templates. Two specific template paths are highlighted with red boxes:
 - /faculty/index (highlighted with a red box)
 - /{faculty-id} (highlighted with a red box)
 - Code Editor:** The code editor shows the Java code for the controller. Red arrows point from the highlighted template paths in the Explorer to the corresponding @GetMapping annotations in the code. A large red arrow points downwards from the code editor towards the bottom right corner of the screen.
 - Status Bar:** Ls 34, Col 6 Spaces: 4 UTF-8 CRLF {} Java

Thymeleaf - Template Student Controller

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\workshop\student\controller (highlighted with a red box)
 - resources
 - static
 - templates
 - course (index.html)
 - enroll (index.html)
 - faculty (index.html)
 - student (index.html, backup.html, index.html, template.html)
 - application.properties
 - test
 - target
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN
- Central Area:** Code editor for **StudentController.java**.

```
public class StudentController {  
    @GetMapping("", "/")  
    public String getAll() {  
        System.out.println("----- StudentController getAll() -----");  
  
        List<StudentEntity> students = studentService.getStudentAll();  
        System.out.println("----- StudentController getAll() Result -----");  
        System.out.println("Size: " + students.size());  
  
        return "student/index";  
    }  
  
    @GetMapping("/{student-id}")  
    public String getById(@PathVariable(name = "student-id") Integer studentId) {  
        System.out.println("----- StudentController getById() -----");  
        System.out.println("student-id: " + studentId);  
  
        StudentEntity entity = studentService.getStudentById(studentId);  
        System.out.println("----- StudentController getById() Result -----");  
        System.out.println("Student First Name: " + entity.getStudentFirstName());  
        System.out.println("Student Last Name: " + entity.getStudentLastName());  
  
        return "student/index";  
    }  
  
    @GetMapping("/delete/{student-id}")  
}
```
- Bottom Status Bar:** Line 99, Col 25, Spaces: 4, UTF-8, CRLF, Java

Thymeleaf - Template Course Controller

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

- Title Bar:** File Edit Selection View ... student
- Toolbar:** Standard icons for file operations.
- Search Bar:** student
- Explorer:** Shows the project structure under STUDENT.
 - src
 - main
 - java\com\workshop\student\controller (highlighted with a red box)
 - resources
 - static
 - templates
 - course (highlighted with a red box)
 - index.html (highlighted with a red box)
 - enroll
 - faculty
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - test
 - target
 - Code Editor:** CourseController.java
 - Code content:

```
public class CourseController {  
    @GetMapping({"/", "/"})  
    public String getAll() {  
        System.out.println("----- CourseController getAll() -----");  
  
        List<CourseEntity> courses = courseService.getCourseAll();  
        System.out.println("----- CourseController getAll() Result -----");  
        System.out.println("Size: " + courses.size());  
  
        return "course/index";  
  
    }  
  
    @GetMapping("/{course-id}")  
    public String getById(  
        @PathVariable(name = "course-id") Integer courseId  
    ) {  
        System.out.println("----- CourseController getById() -----");  
        System.out.println("course-id: " + courseId);  
  
        CourseEntity entity = courseService.getCourseById(courseId);  
        System.out.println("----- CourseController getById() Result -----");  
        System.out.println("Course Name: " + entity.getCourseName());  
  
        return "course/index";  
    }  
  
    @GetMapping("/delete/{course-id}")  
}
```
 - Annotations: `@GetMapping`, `@PathVariable`.
 - Line numbers: 19, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49.
 - Status Bar:** Line 46, Col 24, Spaces: 4, UTF-8, CRLF, Java

Thymeleaf - Template Enroll Controller

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\workshop\student\controller (highlighted with a red box)
 - resources
 - static
 - templates
 - course
 - index.html
 - enroll (highlighted with a red box)
 - index.html
 - faculty
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - target
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Central Area (Code Editor):** EnrollController.java (1 tab open). The code defines two methods:

```
public class EnrollController {  
    @GetMapping({ "", "/" })  
    public String getAll() {  
        System.out.println(x: "----- EnrollController getAll() -----");  
  
        List<EnrollEntity> enrolls = enrollService.getEnrollAll();  
        System.out.println(x: "----- EnrollController getAll() Result -----");  
        System.out.println("Size: " + enrolls.size());  
  
        return "enroll/index";  
    }  
  
    @GetMapping("/{enroll-id}")  
    public String getById(  
        @PathVariable(name = "enroll-id") Integer enrollId  
    ) {  
        System.out.println(x: "----- EnrollController getById() -----");  
        System.out.println("enroll-id: " + enrollId);  
  
        EnrollEntity entity = enrollService.getEnrollById(enrollId);  
        System.out.println(x: "----- EnrollController getById() Result -----");  
        System.out.println("Course Name: " + entity.getCourse().getCourseName());  
        System.out.println("Student First Name: " + entity.getStudent().getStudentFirstName());  
        System.out.println("Student Last Name: " + entity.getStudent().getStudentLastName());  
  
        return "enroll/index";  
    }  
}
```
- Bottom Status Bar:** L1 103, Col 31, Spaces: 4, UTF-8, CRLF, {}, Java

Thymeleaf - Template Faculty Test

A screenshot of a web browser window titled "localhost:8080/faculty". The browser has two tabs open: "localhost:8080/faculty" and "H2 Console". The main content area shows a navigation bar with five items: "Student Management", "Faculty", "Student", "Course", and "Enroll". The "Faculty" item is highlighted with a red border. Below the navigation bar, the text "Faculty" is displayed next to a green checkmark icon.

Thymeleaf - Template Student Test

A screenshot of a web browser window titled "localhost:8080/student". The browser has two tabs open: "localhost:8080/student" and "H2 Console". The main content area shows a navigation bar with five items: "Student Management", "Faculty", "Student", "Course", and "Enroll". The "Student" item is highlighted with a red box. Below the navigation bar, the word "Student" is displayed next to a green checkmark icon.

Thymeleaf - Template Course Test

A screenshot of a web browser window titled "localhost8080/course". The browser has two tabs open: "localhost8080/course" and "H2 Console". The main content area shows a dark-themed navigation bar with five items: "Student Management", "Faculty", "Student", "Course", and "Enroll". The "Course" item is highlighted with a red rectangular border. Below the navigation bar, the word "Course" is displayed in green text next to a green checkmark icon.

Thymeleaf - Template Enroll Test

A screenshot of a web browser window titled "localhost:8080/enroll". The browser has two tabs open: "localhost:8080/enroll" and "H2 Console". The main content area shows a dark-themed navigation bar with the following items: "Student Management" (highlighted in red), "Faculty", "Student", "Course", and "Enroll". Below the navigation bar, the word "Enroll" is displayed next to a large green checkmark icon.

ThymeLeaf - Template / fragment / replace / insert

th:block / th:fragment ✓

th:replace / th:insert ✓

th:href + @ { ... } ✓ <https://www.thymeleaf.org/doc/articles/standardurlsyntax.html>

th:classappend <

Thymeleaf - Template th:classappend

The screenshot shows a code editor interface with a dark theme. On the left is a vertical toolbar with various icons. The main area displays an HTML template file named "template.html". The code is as follows:

```
src > main > resources > templates > template.html > html > body
  2   <html>
  12  <body>
  13
  14      <div id="menu" class="mt-4" th:fragment="menu(active)">
  15          <div class="container mx-auto bg-black rounded">
  16              <ul class="flex text-white text-center">
  17                  <li class="px-8 py-4 font-bold">
  18                      Student Management
  19                  </li>
  20                  <li class="px-8 py-4">
  21                      th:classappend="${#strings.equals(active,'faculty')?'bg-slate-800 font-bold':''}">
  22                          <a th:href="@{/faculty}">Faculty</a>
  23                  </li>
  24                  <li class="px-8 py-4">
  25                      th:classappend="${#strings.equals(active,'student')?'bg-slate-800 font-bold':''}">
  26                          <a th:href="@{/student}">Student</a>
  27                  </li>
  28                  <li class="px-8 py-4">
  29                      th:classappend="${#strings.equals(active,'course')?'bg-slate-800 font-bold':''}">
  30                          <a th:href="@{/course}">Course</a>
  31                  </li>
  32                  <li class="px-8 py-4">
  33                      th:classappend="${#strings.equals(active,'enroll')?'bg-slate-800 font-bold':''}">
  34                          <a th:href="@{/enroll}">Enroll</a>
  35                  </li>
  36          </ul>
  37      </div>
  38  </div>
```

Annotations:

- A red box highlights the line `th:fragment="menu(active)"`. A red circle with the number 1 is positioned above the line.
- Four red boxes highlight the `th:classappend` attributes on lines 21, 25, 29, and 33. A red circle with the number 2 is positioned to the right of the third highlighted line.

Bottom status bar: Ln 39, Col 1 Spaces: 4 UTF-8 CRLF HTML

Thymeleaf - Template th:classappend

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard window control icons.
- Left Sidebar:** Explorer view showing the project structure under STUDENT. A red box highlights the "faculty" folder and its "index.html" file.
- Central Area:** Code editor showing "index.html" content. The line containing "<th:block th:replace=""~{template::menu(faculty)}"" is highlighted with a red box.
- Bottom Status Bar:** Ln 13, Col 8, Spaces: 4, UTF-8, CRLF, HTML, Q.

```
src > main > resources > templates > faculty > index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <th:block th:replace="~{template::head}"></th:block>
5  </head>
6  <body>
7  |   <th:block th:replace="~{template::menu(faculty)}"></th:block>
8  |   <th:block th:replace="~{template::content-begin}"></th:block>
9  |   <!-- Content -->
10 |   Faculty
11 |   <th:block th:replace="~{template::content-end}"></th:block>
12 </body>
13 </html>
```

Thymeleaf - Template th:classappend

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Left Sidebar (EXPLORER):** Shows the project structure under STUDENT.
 - src
 - main
 - java\com\workshop...
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - Central Area:** Shows the content of index.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <th:block th:replace="~{template::head}"></th:block>
</head>
<body>
    <th:block th:replace="~{template::menu('student')}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Student
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
</html>
```
 - Bottom Status Bar:** Ln 13, Col 8 Spaces: 4 UTF-8 CRLF HTML

Thymeleaf - Template th:classappend

The screenshot shows a Java application named "STUDENT" in an IDE. The project structure is as follows:

- src**: Contains **main**, **java\com\workshop** (containing **repository**, **service**, and **StudentApplication.java**), **resources** (containing **static** and **templates** folder), and **enroll**, **faculty**, **student** (each containing **index.html**), **backup.html**, **template.html**, and **application.properties**.
- templates**: Contains **course** (containing **index.html**) and **enroll**, **faculty**, **student** (each containing **index.html**), **backup.html**, **template.html**, and **application.properties**.

The **index.html** file under **course** is open in the editor. The code is:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <th:block th:replace="~{template::head}"></th:block>
</head>
<body>
    <th:block th:replace="~{template::menu('course')}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Course
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
</html>
```

The placeholder `menu('course')` is highlighted with a red box.

Bottom status bar: Ln 7, Col 52 Spaces: 4 UTF-8 CRLF HTML

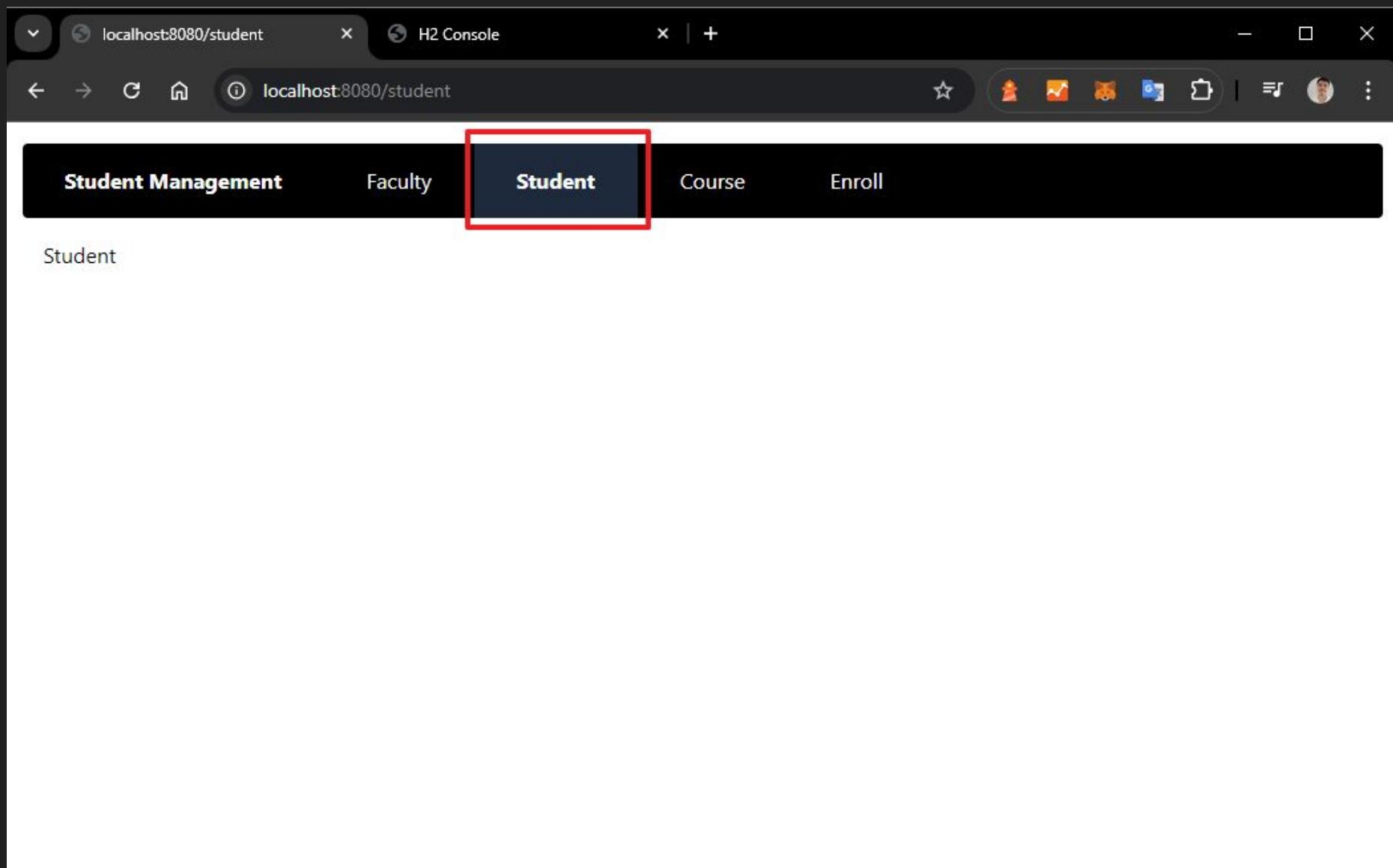
Thymeleaf - Template th:classappend

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\worksho... (highlighted)
 - repository
 - service (highlighted)
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - index.html
 - enroll
 - index.html (highlighted)
 - faculty
 - index.html
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - Right Panel:** Shows the content of the selected file, `index.html`, which is a Thymeleaf template. The code includes:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <th:block th:replace="~{template::head}"></th:block>
</head>
<body>
    <th:block th:replace="~{template::menu('enroll')}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Enroll
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
</html>
```
 - Status Bar:** Ln 13, Col 8 Spaces: 4 UTF-8 CRLF HTML

Thymeleaf - Template th:classappend



ThymeLeaf - Template / fragment / replace / insert

th:block / th:fragment ✓

th:replace / th:insert ✓

th:href + @ { ... } ✓ <https://www.thymeleaf.org/doc/articles/standardurlsyntax.html>

th:classappend ✓

th:text / th:each / th:value <

Thymeleaf - Template Create Form

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard window control icons.
- Left Sidebar (EXPLORER):** Shows the project structure under STUDENT.
 - src
 - main
 - java\com\workshop...
 - service (StudentApplication.java)
 - resources
 - static
 - templates
 - course (index.html)
 - enroll (index.html)
 - faculty
 - index.html
 - student (index.html, backup.html, index.html, template.html)
- Central Area:** The file index.html is open in the editor. The code is a Thymeleaf template for creating a faculty record.

```
<html lang="en">
<body>
    <th:block th:replace="~{template::content-begin}"></th:block>
    <!-- Content -->
    Faculty
    <form th:action="@{/faculty/}" method="post">
        <div class="my-2">
            <label class="w-32 inline-block">Faculty Id:</label>
            <input type="text" name="faculty-id"
                   class="border border-blue-600 p-1 rounded w-80"
                   value="0" disabled
            />
        </div>
        <div class="my-2">
            <label class="w-32 inline-block">Faculty Name:</label>
            <input type="text" name="faculty-name"
                   class="border border-blue-600 p-1 rounded w-80"
            />
        </div>
        <div class="my-2 text-right">
            <button type="submit"
                   class="text-white font-bold bg-blue-600 py-2 px-4 rounded">
                Save Faculty
            </button>
        </div>
    <th:block th:replace="~{template::content-end}"></th:block>
</body>
```
- Bottom Status Bar:** Shows build status (0 errors, 0 warnings), Java: Ready, and file statistics (Ln 9, Col 21, Spaces: 4, UTF-8, CRLF, HTML).

Thymeleaf - Template Web

localhost8080/faculty/ H2 Console

localhost8080/faculty

Student Management Faculty Student Course Enroll

Faculty

Faculty Id: 0

Faculty Name: Computer

Save Faculty

Thymeleaf - Template Console

The screenshot shows an IDE interface with the title "Thymeleaf - Template Console". The left sidebar displays a project structure under "STUDENT" with several "index.html" files located in different folders. The main editor window shows the content of "index.html" from the "faculty" folder. The terminal below shows the output of running this template, which includes a form for entering Faculty ID and Name, and a successful insertion into a database:

```
src > main > resources > templates > faculty > index.html > html > body
2   <html lang="en">
6     <body>
11       <form th:action="@{/faculty/}" method="post">
12         <div class="my-2">
13           <label class="w-32 inline-block">Faculty Id:</label>
14           <input type="text" name="faculty-id"
15             class="border border-blue-600 p-1 rounded w-80"
16             value="0" disabled
17           />
18         </div>
19         <div class="my-2">
20           <label class="w-32 inline-block">Faculty Name:</label>
21           <input type="text" name="faculty-name"
22             class="border border-blue-600 p-1 rounded w-80"
23           />
PROBLEMS 1 OUTPUT PORTS DEBUG CONSOLE TERMINAL
Maven-student + ▾
```

select
next value for faculty_seq
Hibernate:
insert
into
faculty
(faculty_name, faculty_id)
values
(:, :)

Faculty ID: 1
Faculty Name: Computer

A red box highlights the "Faculty ID: 1" and "Faculty Name: Computer" entries in the terminal output, and a green checkmark is placed next to the highlighted text.

Thymeleaf - Template Map data to Model

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Explorer:** Shows the project structure under STUDENT:
 - .vscode
 - data
 - src
 - main
 - java\com\workshop...
 - controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - index.html
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
 - Editor:** FacultyController.java
 - Code:**

```
20  @Controller
21  @RequestMapping("/faculty")
22  public class FacultyController {
23
24      @Autowired
25      private FacultyService facultyService;
26
27      @GetMapping("", "/")
28      public String getAll(ModelMap model) {
29          System.out.println("----- getAll() -----");
30
31          // List<FacultyEntity> faculties = facultyService.getFacultyAll();
32          // System.out.println("----- getAll() Result -----");
33          // System.out.println("Size: " + faculties.size());
34
35          List<FacultyEntity> faculties = facultyService.getFacultyAll();
36          model.addAttribute("faculties", faculties);
37
38          return "faculty/index";
39      }
40
41      @GetMapping("/{faculty-id}")
42      public String getById(
43          @PathVariable(name = "faculty-id") Integer facultyId
44      ) {
45          System.out.println("----- getById() -----");
46          System.out.println("faculty-id: " + facultyId);
47      }
}
```
 - Annotations:** The code block from line 27 to line 37 is highlighted with a red rectangle. A red circle with a plus sign is positioned above the start of this block, and another red circle with a minus sign is positioned below its end.
 - Status Bar:** Ln 40, Col 2 Spaces: 4 UTF-8 CRLF {} Java

Thymeleaf - Template th:text th:each

The screenshot shows an IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\worksho... (contains StudentApplication.java)
 - resources
 - static
 - templates
 - course (contains index.html)
 - enroll (contains index.html)
 - faculty (contains index.html)
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - Center Area:** Shows the content of `index.html`.

```
<html lang="en">
<body>
    <table class="w-full">
        <thead>
            <tr class="text-center">
                <td class="w-[10%]">#</td>
                <td class="w-[10%]">Faculty ID</td>
                <td class="w-[60%]">Faculty Name</td>
                <td class="w-[20%]">Action</td>
            </tr>
        </thead>
        <tbody>
            <tr th:each="faculty, iterStat: ${faculties}">
                <td th:text="${iterStat.count}" class="text-center"></td>
                <td th:text="${faculty.facultyId}" class="text-center"></td>
                <td th:text="${faculty.facultyName}"></td>
                <td class="text-center">
                    <a th:href="@{/faculty/{id}(id=${faculty.facultyId})}">
                        Edit
                    </a>
                    <a th:href="@{/faculty/delete/{id}(id=${faculty.facultyId})}">
                        Delete
                    </a>
                </td>
            </tr>
        </tbody>
    </table>
</body>
```

A red box highlights the `th:each` loop, and a red arrow points from the word "faculty" in the loop to the `faculty` variable in the `th:each` attribute.
 - Bottom Status Bar:** Labeled "Java: Ready".
 - Bottom Right:** Line 32, Col 1, Spaces: 4, UTF-8, CRLF, HTML, and a refresh icon.

6.2 Keeping iteration status

When using `th:each`, Thymeleaf offers a mechanism useful for keeping track of the status of your iteration: the `status variable`.

Status variables are defined within a `th:each` attribute and contain the following data:

- The current *iteration index*, starting with 0. This is the `index` property.
- The current *iteration index*, starting with 1. This is the `count` property.
- The total amount of elements in the iterated variable. This is the `size` property.
- The `iter variable` for each iteration. This is the `current` property.
- Whether the current iteration is even or odd. These are the `even/odd` boolean properties.
- Whether the current iteration is the first one. This is the `first` boolean property.
- Whether the current iteration is the last one. This is the `last` boolean property.

Let's see how we could use it with the previous example:

```
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
  </tr>
  <tr th:each="prod,iterStat : ${prods}" th:class="${iterStat.odd}? 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #{true} : #{false}">yes</td>
  </tr>
</table>
```

The status variable (`iterStat` in this example) is defined in the `th:each` attribute by writing its name after the `iter` variable itself, separated by a comma. Just like the `iter` variable, the status variable is also scoped to the fragment of code defined by the tag holding the `th:each` attribute.

Thymeleaf - Template Web

localhost:8080/faculty

localhost:8080/faculty

Student Management Faculty Student Course Enroll

Faculty

Faculty Id: 0

Faculty Name:

Save Faculty

#	Faculty ID	Faculty Name	Action
1	1	Computer	Edit Delete

Thymeleaf - Template postInsertAndUpdate()

The screenshot shows an IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Explorer:** STUDENT folder structure:
 - .mvn
 - .vscode
 - settings.json
 - data
 - src
 - main
 - java\com\workshop...
 - controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN- Code Editor:** FacultyController.java (line 1)

```
public class FacultyController {  
    }  
  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        ModelMap model,  
        @RequestParam() Map<String, String> param  
    ) {  
        // System.out.println("---- postInsertAndUpdate() ----");  
        // System.out.println("faculty-id: " + param.get("faculty-id"));  
        // System.out.println("faculty-name: " + param.get("faculty-name"));  
  
        // System.out.println("---- postInsertAndUpdate() Result ----");  
        FacultyEntity entity = new FacultyEntity();  
        if(null != param.get(key:"faculty-id")) {  
            entity.setFacultyId(Integer.parseInt(param.get(key:"faculty-id")));  
        }  
        entity.setFacultyName(param.get(key:"faculty-name"));  
        FacultyEntity result = facultyService.saveFaculty(entity);  
        // System.out.println("Faculty ID: " + result.getFacultyId());  
        // System.out.println("Faculty Name: " + result.getFacultyName());  
  
        List<FacultyEntity> faculties = facultyService.getFacultyAll();  
        model.addAttribute("faculties", faculties);  
  
        return "faculty/index";  
    }  
}
```
- Status Bar:** L 96, Col 1 Spaces: 4 UTF-8 CRLF {} Java

Thymeleaf - Template Test

The screenshot shows a web browser window with two tabs: "localhost:8080/faculty/" and "H2 Console". The main content area displays a "Faculty" management interface. At the top, there is a navigation bar with tabs: "Student Management", "Faculty" (which is active), "Student", "Course", and "Enroll". Below the navigation bar, the word "Faculty" is displayed. There are two input fields: "Faculty Id:" containing "0" and "Faculty Name:" which is empty. To the right of these fields is a blue button labeled "Save Faculty". Below this section is a table with three rows of data:

#	Faculty ID	Faculty Name	Action
1	1	Computer	Edit Delete
2	2	Law	Edit Delete
3	3	Science	Edit Delete

At the bottom right of the page, the word "TEST" is written in red.

Thymeleaf - Template deleteById()

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a project structure under "STUDENT". The "controller" folder contains several Java files: CourseController.java, EnrollController.java, FacultyController.java (highlighted in red), StudentController.java, and TutorialController.java.
- Editor (Center):** Displays the content of FacultyController.java. A red box highlights the following code block:

```
@GetMapping("/delete/{faculty-id}")
public String getDeleteById(
    ModelMap model,
    @PathVariable(name = "faculty-id") Integer facultyId
) {
    // System.out.println("---- getDeleteById() ----");
    // System.out.println("faculty-id: " + facultyId);

    // System.out.println("---- getDeleteById() Result ----");
    facultyService.deleteFacultyById(facultyId);

    List<FacultyEntity> faculties = facultyService.getFacultyAll();
    model.addAttribute("faculties", faculties);

    return "faculty/index";
}
```
- Bottom Status Bar:** Shows build status (0 errors, 0 warnings), Java: Ready, and file statistics (Ln 75, Col 6, Spaces: 4, UTF-8, CRLF).

Thymeleaf - Template

The screenshot shows a web application running on localhost:8080. The URL in the address bar is `localhost:8080/faculty/`. The page has a dark theme with a top navigation bar containing links for Student Management, Faculty, Student, Course, and Enroll.

The main content area is titled "Faculty". It contains two input fields: "Faculty Id:" with value "0" and "Faculty Name:" with an empty input field. To the right of these fields is a blue button labeled "Save Faculty".

Below these fields is a table with three columns: "#", "Faculty ID", and "Faculty Name". The table has two rows:

#	Faculty ID	Faculty Name	Action
1	1	Computer	Edit Delete
2	2	Law	Edit Delete

A red box highlights the "Delete" link in the action column for the first row (Faculty ID 1). At the bottom left of the page, there is a red-bordered input field containing the URL `localhost:8080/faculty/delete/1`.

Thymeleaf - Template

localhost:8080/faculty/delete/1 H2 Console

localhost:8080/faculty/delete/1

Student Management Faculty Student Course Enroll

Faculty

Faculty Id: 0

Faculty Name:

Save Faculty

#	Faculty ID	Faculty Name	Action
1	2	Law	Edit Delete

Thymeleaf - Template getById()

The screenshot shows a Java application structure in the Explorer sidebar under the 'STUDENT' project. The 'controller' package contains several files: CourseController.java, EnrollController.java, FacultyController.java (highlighted in yellow), StudentController.java, and TutorialController.java. The FacultyController.java file is open in the editor.

```
public class FacultyController {
    public String getAll(ModelMap model) {
        return "faculty/index";
    }

    @GetMapping("/{faculty-id}")
    public String getById(
        ModelMap model,
        @PathVariable(name = "faculty-id") Integer facultyId
    ) {
        // System.out.println("---- getById() ----");
        // System.out.println("faculty-id: " + facultyId);

        FacultyEntity entity = facultyService.getFacultyById(facultyId);
        // System.out.println("---- getById() Result ----");
        // System.out.println("Faculty Name: " + entity.getFacultyName());
        model.addAttribute("faculty", entity);

        List<FacultyEntity> faculties = facultyService.getFacultyAll();
        model.addAttribute("faculties", faculties);
    }

    return "faculty/index";
}

@GetMapping("/delete/{faculty-id}")
public String getDeleteById(
    ModelMap model,
    @PathVariable(name = "faculty-id") Integer facultyId
)
```

The code implements a controller method for retrieving a faculty by ID. It uses Thymeleaf's `@GetMapping` annotation and `@PathVariable` to handle the URL and extract the ID. It then calls a service method to get the faculty entity and adds it to the model. Finally, it returns the 'faculty/index' template. A red box highlights the section where the faculty entity is added to the model, and another red box highlights the entire block of code for retrieving all faculties.

Thymeleaf - Template index.html

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho... (contains StudentApplication.java)
 - resources
 - static
 - templates
 - course (contains index.html)
 - enroll (contains index.html)
 - faculty (contains index.html, highlighted)
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - Center Area:** index.html (selected tab) and FacultyController.java (1)
 - Code Editor:** index.html content (line numbers 2 to 32). The code uses Thymeleaf syntax to replace sections of the template.

```
<html lang="en">
<body>
    <th:block th:replace="~{template::menu(faculty)}"></th:block>
    <th:block th:replace="~{template::content-begin}"></th:block>
    
    Faculty
    <form th:action="@{/faculty/}" method="post">
        <div class="my-2">
            <label class="w-32 inline-block">Faculty Id:</label>
            <input type="text" name="faculty-id"
                   class="border border-blue-600 p-1 rounded w-80"
                   th:value="${faculty?.facultyId?:'0'}" readonly
            />
        </div>
        <div class="my-2">
            <label class="w-32 inline-block">Faculty Name:</label>
            <input type="text" name="faculty-name"
                   class="border border-blue-600 p-1 rounded w-80"
                   th:value="${faculty?.facultyName?:''}"
            />
        </div>
        <div class="my-2 text-right">
            <button type="submit"
                   class="text-white font-bold bg-blue-600 py-2 px-4 rounded">
                Save Faculty
            </button>
        </div>
    </form>
```
 - Bottom Status Bar:** Ln 32, Col 12, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template Test

The screenshot shows a web application interface for managing faculty members. The top navigation bar includes links for Student Management, Faculty, Student, Course, and Enroll. The Faculty link is currently active, highlighted in blue.

The main content area is titled "Faculty". It contains two input fields: "Faculty Id:" with value "0" and "Faculty Name:" with an empty input field. A large blue button labeled "Save Faculty" is positioned to the right of these fields.

A table below displays faculty data:

#	Faculty ID	Faculty Name	Action
1	1	Computer	Edit Delete
2	2	Law	Edit Delete

The "Edit" and "Delete" links in the last row of the table are highlighted with red boxes. The URL "localhost:8080/faculty/1" is visible in the browser's address bar at the bottom.

Thymeleaf - Template Test

localhost:8080/faculty/1

localhost:8080/faculty/1

Student Management Faculty Student Course Enroll

Faculty

Faculty Id: 1

Faculty Name: Computer Edit

1 Save Faculty

#	Faculty ID	Faculty Name	Action
1	1	Computer	Edit Delete
2	2	Law	Edit Delete

Thymeleaf - Template Faculty Complete

Screenshot of a web application interface for managing faculty data using Thymeleaf templates.

The browser title bar shows "localhost:8080/faculty/" and the tab title is "localhost:8080/faculty/".

The navigation bar includes links for "Student Management", "Faculty" (which is active), "Student", "Course", and "Enroll".

The main content area is titled "Faculty". It contains two input fields: "Faculty Id:" with value "0" and "Faculty Name:" with an empty input field.

A blue button labeled "Save Faculty" is located on the right side of the input fields.

A table displays faculty data:

#	Faculty ID	Faculty Name	Action
1	1	Computer	Edit Delete
2	2	Law	Edit Delete

A green checkmark icon is positioned next to the "Edit" link for the first faculty entry (ID 1).

ThymeLeaf - Template / fragment / replace / insert

th:block / th:fragment ✓

th:replace / th:insert ✓

th:href + @ { ... } ✓ <https://www.thymeleaf.org/doc/articles/standardurlsyntax.html>

th:classappend ✓

th:text / th:each / th:value ✓

th:if / th:unless <

Thymeleaf - Template Student getAll()

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "STUDENT". Key files include `index.html`, `StudentController.java`, `CourseController.java`, `EnrollController.java`, `FacultyController.java`, `StudentController.java` (highlighted with a red box), and `TutorialController.java`.
- Editor (Center):** The `StudentController.java` file is open. The code implements a `getAll` method that retrieves all students and faculties from services and adds them to a `ModelMap`.

```
public class StudentController {  
    @Autowired  
    private FacultyService facultyService;  
  
    @GetMapping("", "/")  
    public String getAll(ModelMap model) {  
        System.out.println("----- StudentController getAll() -----");  
  
        // List<StudentEntity> students = studentService.getStudentAll();  
        // System.out.println("----- StudentController getAll() Result -----");  
        // System.out.println("Size: " + students.size());  
  
        List<FacultyEntity> faculties = facultyService.getFacultyAll();  
        model.addAttribute("faculties", faculties);  
  
        List<StudentEntity> students = studentService.getStudentAll();  
        model.addAttribute("students", students);  
  
        return "student/index";  
    }  
  
    @GetMapping("/{student-id}")  
    public String getById(  
        ModelMap model,  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        System.out.println("----- StudentController getById() -----");  
    }  
}
```
- Search Bar (Top):** Contains the text "student".
- Status Bar (Bottom):** Shows "Java: Ready", "Ln 47, Col 2", "Spaces: 4", "UTF-8", "CRLF", and file navigation icons.

Thymeleaf - Template Student getById()

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder. The "controller" folder contains several Java files: CourseController.java, EnrollController.java, FacultyController.java, and StudentController.java (which is currently selected and highlighted with a red border). Other files like TutorialController.java, entity, repository, service, and resources are also listed.
- Editor (Center):** Displays the content of StudentController.java. The code defines a controller method `getById` that takes a `ModelMap model` and a `@PathVariable(name = "student-id") Integer studentId`. It prints the student ID and then retrieves the student entity from the service. The retrieved entity is added to the model with the key "student". Additionally, it retrieves all faculties and students and adds them to the model with keys "faculties" and "students" respectively. The code ends with a `return "student/index";` statement and a final `@GetMapping("/delete/{student-id}")` annotation.
- Search Bar (Top):** Contains the text "student".
- Status Bar (Bottom):** Shows "Java: Ready" and other status indicators like file counts and build logs.

```
public class StudentController {  
    @GetMapping("/{student-id}")  
    public String getById(  
        ModelMap model,  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        System.out.println("----- StudentController getById() -----");  
        System.out.println("student-id: " + studentId);  
  
        // StudentEntity entity = studentService.getStudentById(studentId);  
        // System.out.println("----- StudentController getById() Result -----");  
        // System.out.println("Student First Name: " + entity.getStudentFirstName());  
        // System.out.println("Student Last Name: " + entity.getStudentLastName());  
  
        StudentEntity entity = studentService.getStudentById(studentId);  
        model.addAttribute("student", entity);  
  
        List<FacultyEntity> faculties = facultyService.getFacultyAll();  
        model.addAttribute("faculties", faculties);  
  
        List<StudentEntity> students = studentService.getStudentAll();  
        model.addAttribute("students", students);  
    }  
    @GetMapping("/delete/{student-id}")  
}
```

Thymeleaf - Template Student getDeleteById()

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder, including ".mvn", ".vscode", "data", "src" (with "main" and "controller" subfolders), "entity", "repository", "service", "StudentApplication.java", "resources", "static", and "templates". The "StudentController.java" file is selected and highlighted.
- Search Bar (Top):** Contains the text "student".
- Code Editor (Center):** Displays the "StudentController.java" code. A red box highlights the following section of code:

```
@GetMapping("/delete/{student-id}")
public String getDeleteById(
    ModelMap model,
    @PathVariable(name = "student-id") Integer studentId
) {
    System.out.println(x:"---- StudentController getDeleteById() -----");
    System.out.println("student-id: " + studentId);

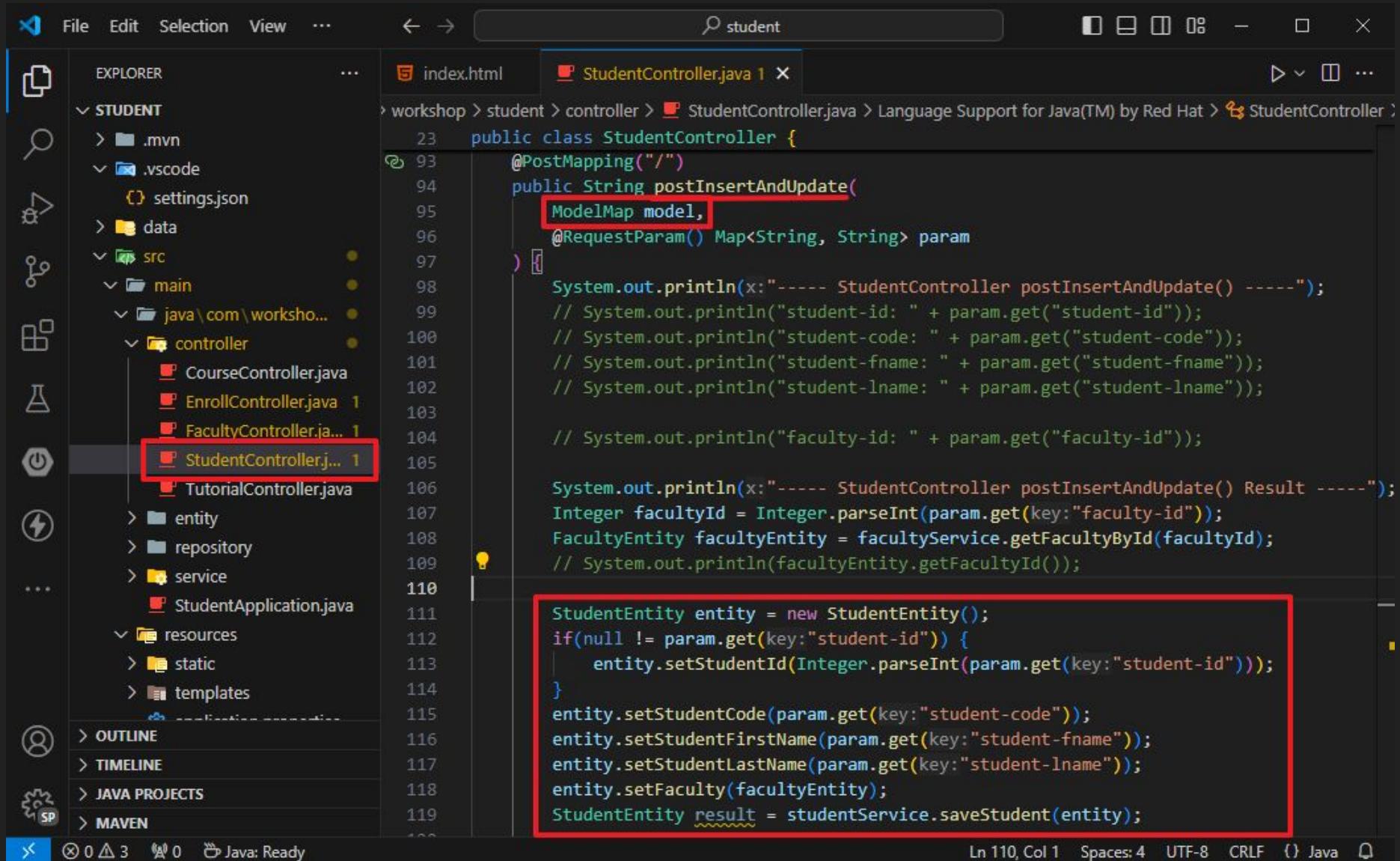
    System.out.println(x:"---- StudentController getDeleteById() Result -----");
    studentService.deleteStudentById(studentId);

    List<FacultyEntity> faculties = facultyService.getFacultyAll();
    model.addAttribute("faculties", faculties);

    List<StudentEntity> students = studentService.getStudentAll();
    model.addAttribute("students", students);

    return "student/index";
}
```
- Bottom Status Bar:** Shows "Ln 92, Col 6" and "Java: Ready".

Thymeleaf - Template Student postInsertAndUpdate()



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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard window control icons.
- Left Sidebar (EXPLORER):**
 - STUDENT**: .mvn, .vscode (with settings.json), data, src (with main, java, controller, entity, repository, service, resources, static, templates).
 - OUTLINE**, **TIMELINE**, **JAVA PROJECTS**, **MAVEN**.
- Central Area:** StudentController.java (line 110)
- Code (highlighted block):**

```
public class StudentController {
    @PostMapping("/")
    public String postInsertAndUpdate(
        ModelMap model,
        @RequestParam() Map<String, String> param
    ) {
        System.out.println("----- StudentController postInsertAndUpdate() -----");
        // System.out.println("student-id: " + param.get("student-id"));
        // System.out.println("student-code: " + param.get("student-code"));
        // System.out.println("student-fname: " + param.get("student-fname"));
        // System.out.println("student-lname: " + param.get("student-lname"));

        // System.out.println("faculty-id: " + param.get("faculty-id"));

        System.out.println("----- StudentController postInsertAndUpdate() Result -----");
        Integer facultyId = Integer.parseInt(param.get("faculty-id"));
        FacultyEntity facultyEntity = facultyService.getFacultyById(facultyId);
        // System.out.println(facultyEntity.getFacultyId());

        StudentEntity entity = new StudentEntity();
        if(null != param.get("student-id")) {
            entity.setStudentId(Integer.parseInt(param.get("student-id")));
        }
        entity.setStudentCode(param.get("student-code"));
        entity.setStudentFirstName(param.get("student-fname"));
        entity.setStudentLastName(param.get("student-lname"));
        entity.setFaculty(facultyEntity);
        StudentEntity result = studentService.saveStudent(entity);
    }
}
```
- Bottom Status Bar:** Line 110, Col 1, Spaces: 4, UTF-8, CRLF, Java

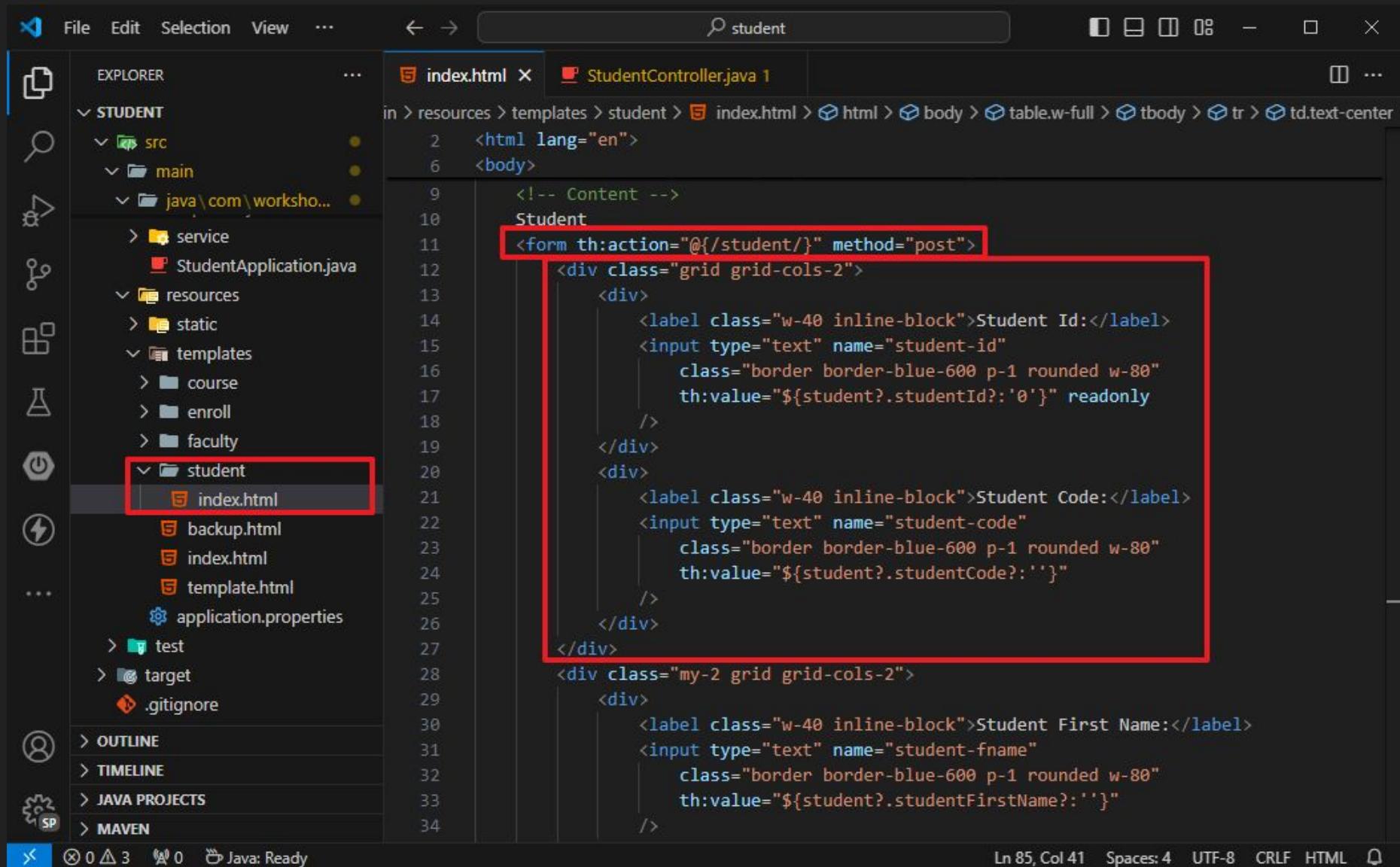
Thymeleaf - Template Student postInsertAndUpdate()

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - .mvn
 - .vscode
 - settings.json
 - data
 - src
 - main
 - java\com\workshop\controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java** (highlighted)
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - Center Area:** Code editor showing **StudentController.java**.

```
public class StudentController {  
    public String postInsertAndUpdate(  
        Integer facultyId = Integer.parseInt(param.get("faculty-id"));  
        FacultyEntity facultyEntity = facultyService.getFacultyById(facultyId);  
        // System.out.println(facultyEntity.getFacultyId());  
  
        StudentEntity entity = new StudentEntity();  
        if(null != param.get("student-id")) {  
            entity.setStudentId(Integer.parseInt(param.get("student-id")));  
        }  
        entity.setStudentCode(param.get("student-code"));  
        entity.setStudentFirstName(param.get("student-fname"));  
        entity.setStudentLastName(param.get("student-lname"));  
        entity.setFaculty(facultyEntity);  
        StudentEntity result = studentService.saveStudent(entity);  
        // System.out.println("Student ID: " + result.getStudentId());  
        // System.out.println("Student First Name: " + result.getStudentFirstName());  
        // System.out.println("Student Last Name: " + result.getStudentLastName());  
  
        List<FacultyEntity> faculties = facultyService.getFacultyAll();  
        model.addAttribute("faculties", faculties);  
  
        List<StudentEntity> students = studentService.getStudentAll();  
        model.addAttribute("students", students);  
    }  
    return "student/index";  
}
```
 - Right Area:** Language Support for Java(TM) by Red Hat > StudentController.java > Continue

Thymeleaf - Template Student index.html Form



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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Explorer:** Shows the project structure under STUDENT:
 - src
 - main
 - java\com\workshop...
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - student
 - index.html (highlighted with a red box)
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - target
 - .gitignore
- Editor:** The index.html file is open, showing Thymeleaf code.

```
<html lang="en">
<body>
    <!-- Content -->
    Student
    <form th:action="@{/student/}" method="post">
        <div class="grid grid-cols-2">
            <div>
                <label class="w-40 inline-block">Student Id:</label>
                <input type="text" name="student-id"
                    class="border border-blue-600 p-1 rounded w-80"
                    th:value="${student?.studentId?:'0'}" readonly
                />
            </div>
            <div>
                <label class="w-40 inline-block">Student Code:</label>
                <input type="text" name="student-code"
                    class="border border-blue-600 p-1 rounded w-80"
                    th:value="${student?.studentCode?:''}"
                />
            </div>
        </div>
        <div class="my-2 grid grid-cols-2">
            <div>
                <label class="w-40 inline-block">Student First Name:</label>
                <input type="text" name="student-fname"
                    class="border border-blue-600 p-1 rounded w-80"
                    th:value="${student?.studentFirstName?:''}"
                />
```
- Status Bar:** Ln 85, Col 41, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template Student index.html Form

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho...
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - student**
 - index.html** (highlighted with a red box)
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - target
 - .gitignore
- Central Area:** The code editor displays the contents of `index.html`. A red box highlights the section of code that generates the student form fields.

```
<html lang="en">
<body>
    <form th:action="@{/student/}" method="post">
        <div class="grid grid-cols-2">
            <div>
                </div>
            </div>
        <div class="my-2 grid grid-cols-2">
            <div>
                <label class="w-40 inline-block">Student First Name:</label>
                <input type="text" name="student-fname"
                    class="border border-blue-600 p-1 rounded w-80"
                    th:value="${student?.studentFirstName?:''}">
            </div>
            <div>
                <label class="w-40 inline-block">Student Last Name:</label>
                <input type="text" name="student-lname"
                    class="border border-blue-600 p-1 rounded w-80"
                    th:value="${student?.studentLastName?:''}">
            </div>
        </div>
        <div class="my-2 grid grid-cols-2">
            <div>
                <label class="w-40 inline-block">Faculty:</label>
                <select name="faculty-id">
```
- Bottom Status Bar:** Ln 85, Col 41, Spaces: 4, UTF-8, CRLF, HTML, etc.

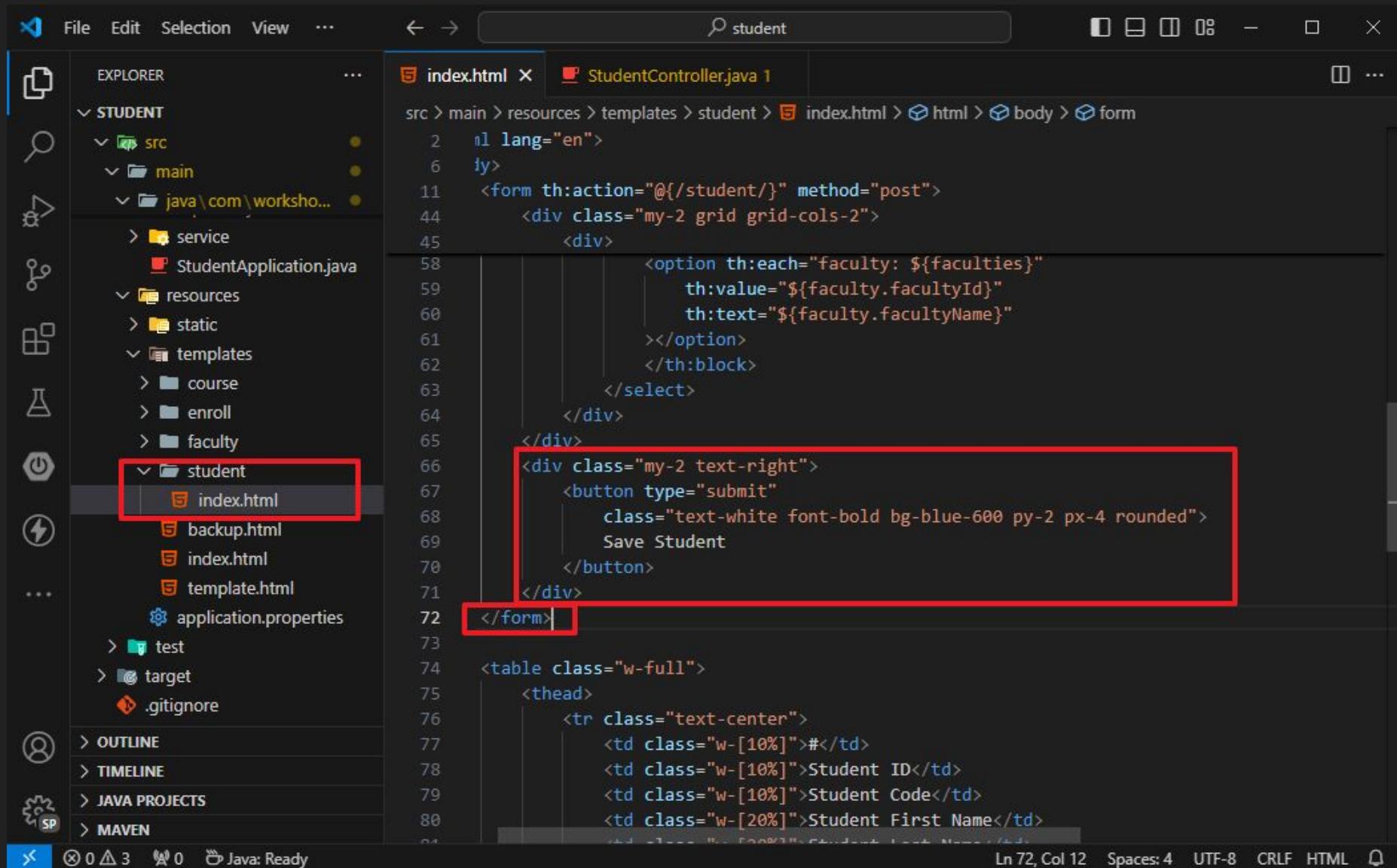
Thymeleaf - Template Student index.html Form

The screenshot shows an IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho...
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - target
 - .gitignore
- Central Area:** Shows the content of index.html.

```
<html lang="en">
    <body>
        <form th:action="@{/student/}" method="post">
            <div class="my-2 grid grid-cols-2">
                <div>
                    <label class="w-40 inline-block">Faculty:</label>
                    <select name="faculty-id" class="border border-blue-600 p-1 rounded w-80">
                        <option value="0">Select Faculty</option>
                        <th:block th:if="${student?.faculty != null}">
                            <option th:each="faculty: ${faculties}" th:value="${faculty.facultyId}" th:text="${faculty.facultyName}" th:selected="${student.faculty.facultyId == faculty.facultyId}"></option>
                        </th:block>
                        <th:block th:unless="${student?.faculty != null}">
                            <option th:each="faculty: ${faculties}" th:value="${faculty.facultyId}" th:text="${faculty.facultyName}"></option>
                        </th:block>
                    </select>
                </div>
            </div>
            <div class="my-2 text-right">
                <button type="submit">
            </div>
        </form>
    </body>
</html>
```
- Bottom Status Bar:** Ln 65, Col 15, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template Student index.html Form



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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Explorer:** Shows the project structure under STUDENT:
 - src
 - main
 - java\com\workshop...
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - student
 - index.html (highlighted with a red box)
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - target
 - .gitignore
- Outline:** Shows the outline of the current file.
- Timeline:** Shows the timeline of the project.
- Java Projects:** Shows the list of Java projects.
- Maven:** Shows the Maven settings.
- Status Bar:** Shows the status "Java: Ready" and other system information like line count (Ln 72), column count (Col 12), and encoding (UTF-8).

The code in the editor is:

```
<html lang="en">
    <body>
        <form th:action="@{/student/}" method="post">
            <div class="my-2 grid grid-cols-2">
                <div>
                    <option th:each="faculty: ${faculties}"
                           th:value="${faculty.facultyId}"
                           th:text="${faculty.facultyName}"></option>
                </th:block>
            </select>
        </div>
        <div class="my-2 text-right">
            <button type="submit"
                   class="text-white font-bold bg-blue-600 py-2 px-4 rounded">
                Save Student
            </button>
        </div>
    </form>
    <table class="w-full">
        <thead>
            <tr class="text-center">
                <td class="w-[10%]">#</td>
                <td class="w-[10%]">Student ID</td>
                <td class="w-[10%]">Student Code</td>
                <td class="w-[20%]">Student First Name</td>
            </tr>
        </thead>
```

Thymeleaf - Template Student index.html Table

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\workshop...
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - target
 - .gitignore
- Center Area:** Shows the content of index.html.

```
<html lang="en">
<body>
    <form th:action="@{/student/}" method="post">
        </form>
        <table class="w-full">
            <thead>
                <tr class="text-center">
                    <td class="w-[10%]">#</td>
                    <td class="w-[10%]">Student ID</td>
                    <td class="w-[10%]">Student Code</td>
                    <td class="w-[20%]">Student First Name</td>
                    <td class="w-[20%]">Student Last Name</td>
                    <td class="w-[20%]">Faculty Name</td>
                    <td class="w-[10%]">Action</td>
                </tr>
            </thead>
            <tbody>
                <tr th:each="student, iterStat: ${students}">
                    <td th:text="${iterStat.count}" class="text-center"></td>
                    <td th:text="${student.studentId}" class="text-center"></td>
                    <td th:text="${student.studentCode}"></td>
                    <td th:text="${student.studentFirstName}"></td>
                    <td th:text="${student.studentLastName}"></td>
                    <td th:text="${student?.faculty?.facultyName?: ''}"></td>
                    <td class="text-center">
                        <a th:href="@{/student/{id}(id=${student.studentId})}">
                            Edit
                        </a>
                    </td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```
- Bottom Status Bar:** Ln 72, Col 12, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template Student index.html Table

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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Standard icons for file operations.
- Explorer:** Shows the project structure under STUDENT:
 - src
 - main
 - java\com\workshop...
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty
 - student
 - index.html
 - backup.html
 - index.html
 - template.html
 - test
 - target
 - .gitignore
 - index.html Content:** The code displays a table of student data using Thymeleaf directives. A red box highlights the entire table structure from the opening <tbody> tag to the closing </table> tag.
 - Code Preview:** Shows the generated HTML output below the Thymeleaf code.
 - Status Bar:** Ln 72, Col 12, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template Student Test

The screenshot shows a web browser window with two tabs: "localhost8080/student/" and "H2 Console". The main content area displays a student management interface.

Header: The top navigation bar includes links for "Student Management", "Faculty", "Student" (which is the active tab), "Course", and "Enroll".

Form Fields: The "Student" tab contains fields for "Student Id" (value: 0), "Student Code" (empty), "Student First Name" (empty), "Student Last Name" (empty), and "Faculty" (dropdown menu with "Select Faculty" option).

Text: A large red "TEST" text is centered above the form fields.

Action Button: A blue "Save Student" button is located on the right side of the form.

Data Table: Below the form, a table lists student data:

#	Student ID	Student Code	Student First Name	Student Last Name	Faculty Name	Action
1	70	001	First	Last	Computer	Edit Delete
2	71	002	First 2 Edit	Last 2	Law	Edit Delete

ThymeLeaf - Template / fragment / replace / insert

th:block / th:fragment ✓

th:replace / th:insert ✓

th:href + @ { ... } ✓ <https://www.thymeleaf.org/doc/articles/standardurlsyntax.html>

th:classappend ✓

th:text / th:each / th:value ✓

th:if / th:unless ✓

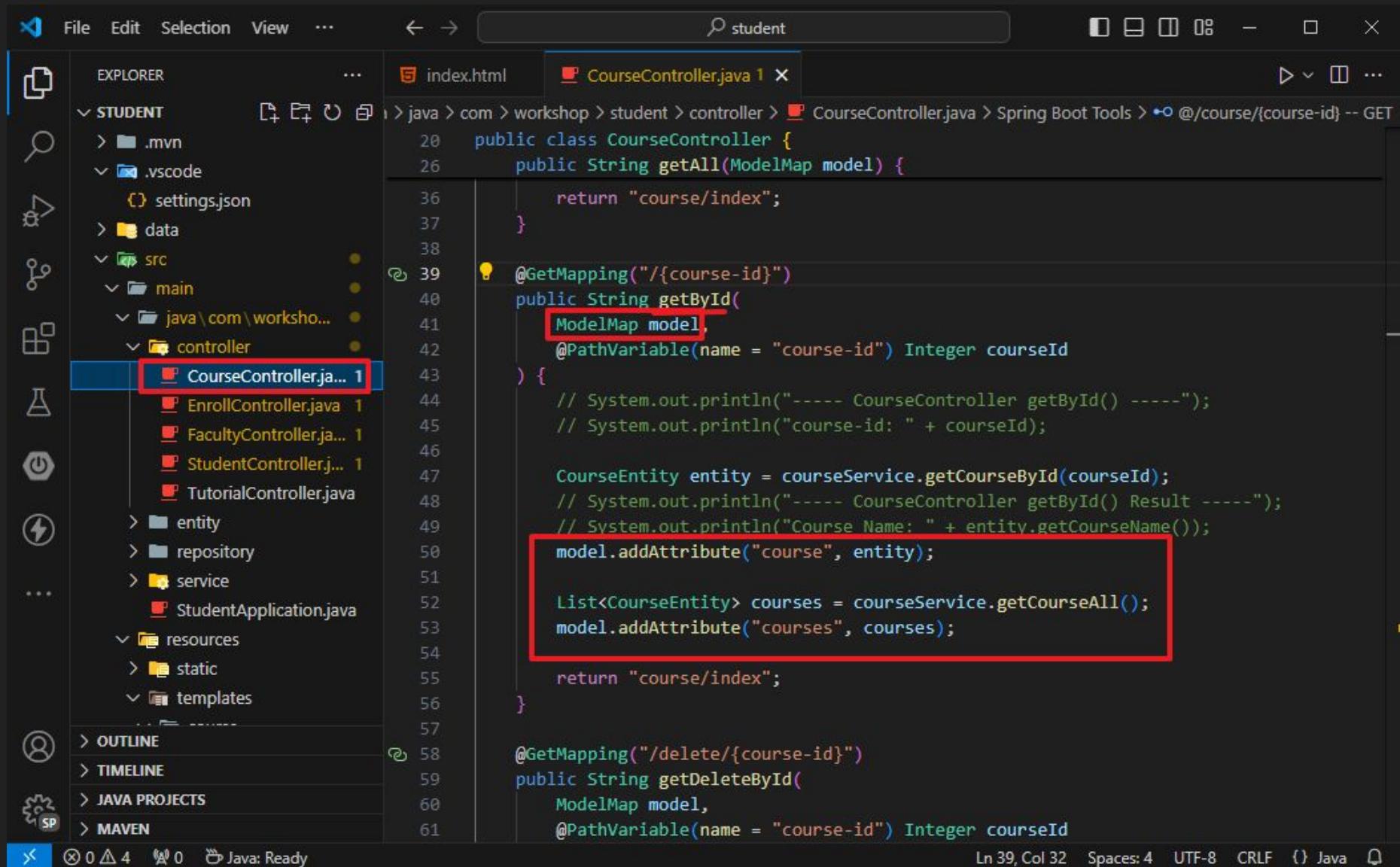
Thymeleaf - Template Course getAll()

The screenshot shows the VS Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (Explorer):**
 - STUDENT** folder:
 - .mvn
 - .vscode (with settings.json)
 - data
 - src
 - main
 - java\com\workshop
 - controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java
 - TutorialController.java
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN
- Central Area:** CourseController.java (18-45 lines).

```
18  @Controller
19  @RequestMapping("/course")
20  public class CourseController {
21
22      @Autowired
23      private CourseService courseService;
24
25      @GetMapping("", "/")
26      public String getAll(ModelMap model) {
27          System.out.println("---- CourseController getAll() ----");
28
29          // List<CourseEntity> courses = courseService.getCourseAll();
30          // System.out.println("---- CourseController getAll() Result ----");
31          // System.out.println("Size: " + courses.size());
32
33          List<CourseEntity> courses = courseService.getCourseAll();
34          model.addAttribute("courses", courses);
35
36          return "course/index";
37      }
38
39      @GetMapping("/{course-id}")
40      public String getById(
41          ModelMap model,
42          @PathVariable(name = "course-id") Integer courseId
43      ) {
44          // System.out.println("---- CourseController getById() ----");
45          // System.out.println("course-id: " + courseId);
46      }
47  }
```
- Bottom Status Bar:** Line 39, Column 32, Spaces: 4, UTF-8, CRLF, Java

Thymeleaf - Template Course getById()



The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder. The "CourseController.java" file is selected and highlighted with a red border.
- Editor (Center):** Displays the "CourseController.java" code. A red box highlights the following section of code:

```
        @GetMapping("/{course-id}")
        public String getById(
            ModelMap model,
            @PathVariable(name = "course-id") Integer courseId
        ) {
            // System.out.println("---- CourseController getById() -----");
            // System.out.println("course-id: " + courseId);

            CourseEntity entity = courseService.getCourseById(courseId);
            // System.out.println("---- CourseController getById() Result -----");
            // System.out.println("Course Name: " + entity.getCourseName());
            model.addAttribute("course", entity);

            List<CourseEntity> courses = courseService.getCourseAll();
            model.addAttribute("courses", courses);
        }

        return "course/index";
    }
```
- Bottom Status Bar:** Shows "Java: Ready" and other standard status bar information.

Thymeleaf - Template Course getDeleteById()

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT:** .mvn, .vscode (with settings.json), data, src (main, java\com\workshop, controller), CourseController.java (highlighted with a red border), EnrollController.java, FacultyController.java, StudentController.java, TutorialController.java, entity, repository, service, StudentApplication.java, resources, static, templates.
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN:** Standard project management sections.
- Central Area:** Code editor for CourseController.java. The code is as follows:

```
public class CourseController {  
    public String getById(  
        ModelMap model,  
        @PathVariable(name = "course-id") Integer courseId  
    ) {  
        // System.out.println("---- CourseController getById() ----");  
        // System.out.println("course-id: " + courseId);  
  
        // System.out.println("---- CourseController getById() Result ----");  
        courseService.deleteCourseById(courseId);  
  
        List<CourseEntity> courses = courseService.getCourseAll();  
        model.addAttribute("courses", courses);  
  
        return "course/index";  
    }  
  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        ModelMap model,  
        @RequestParam() Map<String, String> param  
    ) {  
        // System.out.println("---- CourseController postInsertAndUpdate() ----");  
        courseService.insertOrUpdateCourse(param);  
        return "course/index";  
    }  
}
```
- Bottom Status Bar:** Line 39, Col 32, Spaces: 4, UTF-8, CRLF, Java.

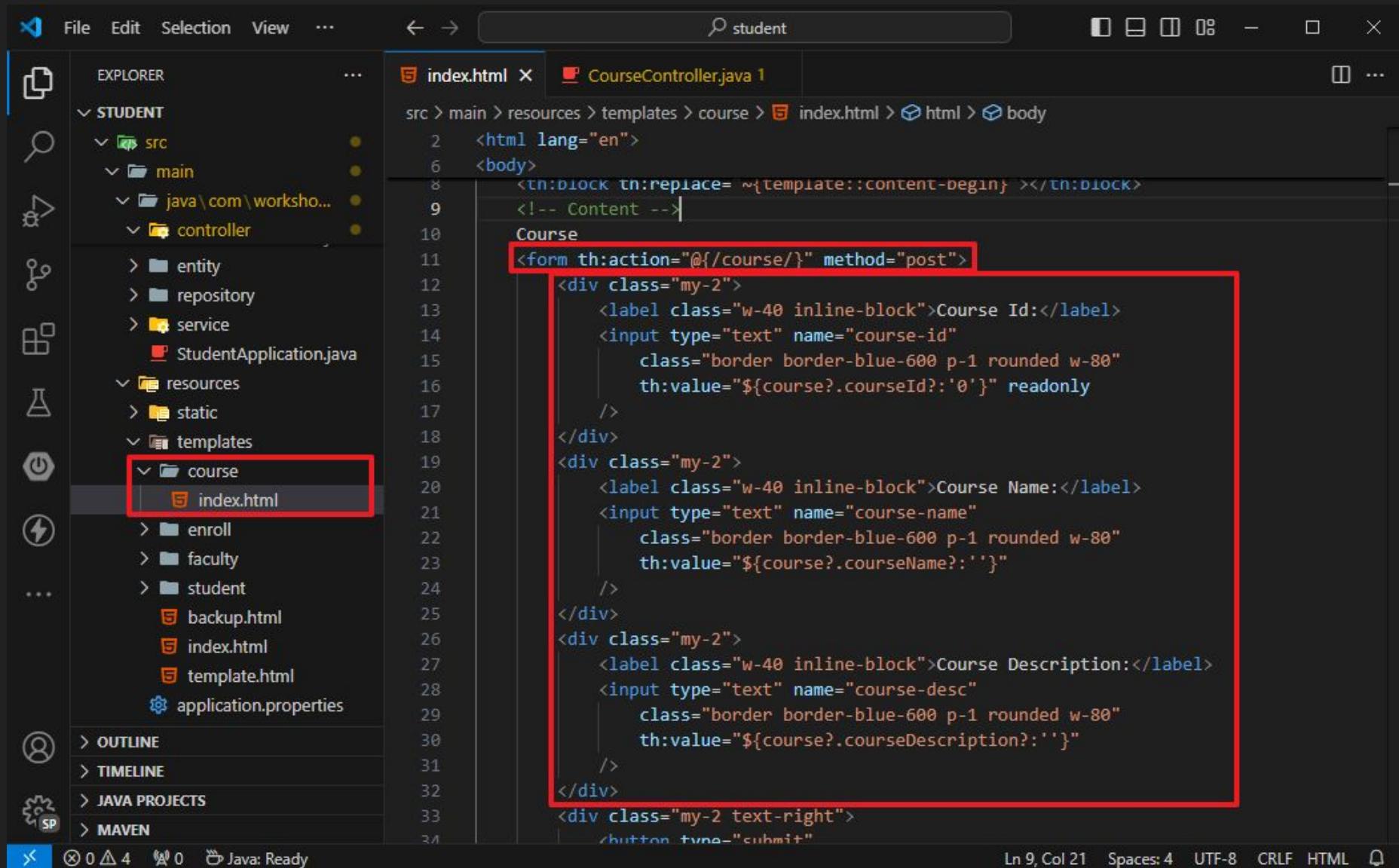
Thymeleaf - Template Course postInsertAndUpdate()

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT:** .mvn, .vscode (with settings.json), data, src (main, java\com\workshop\controller, controller). The **CourseController.java** file is selected and highlighted with a red border.
 - entity, repository, service, StudentApplication.java
 - resources (static, templates)
- Central Area:** Code editor showing **CourseController.java**.

```
public class CourseController {  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        ModelMap model,  
        @RequestParam() Map<String, String> param  
    ) {  
        // System.out.println("---- CourseController postInsertAndUpdate() ----");  
        // System.out.println("course-id: " + param.get("course-id"));  
        // System.out.println("course-name: " + param.get("course-name"));  
        // System.out.println("course-desc: " + param.get("course-desc"));  
  
        // System.out.println("---- CourseController postInsertAndUpdate() Result ----");  
        CourseEntity entity = new CourseEntity();  
        if(null != param.get(key:"course-id")) {  
            entity.setCourseId(Integer.parseInt(param.get(key:"course-id")));  
        }  
        entity.setCourseName(param.get(key:"course-name"));  
        entity.setCourseDescription(param.get(key:"course-desc"));  
        CourseEntity result = courseService.saveCourse(entity);  
        // System.out.println("Course ID: " + result.getId());  
        // System.out.println("Course Name: " + result.getName());  
  
        List<CourseEntity> courses = courseService.getCourseAll();  
        model.addAttribute("courses", courses);  
    }  
    return "course/index";  
}
```
- Bottom Status Bar:** Line 100, Column 32, Spaces: 4, UTF-8, CRLF, Java, etc.

Thymeleaf - Template Course index.html Form



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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Explorer:** Shows the project structure under STUDENT:
 - src
 - main
 - java\com\workshop...
 - controller
 - entity
 - repository
 - service
 - resources
 - static
 - templates
 - course
 - index.html
 - enroll
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
 - Code Editor:** The current file is index.html, located at src/main/resources/templates/course/index.html. The code defines a Thymeleaf form for adding a course:

```
<html lang="en">
<body>
    <tn:block tn:replace= ~{template::content-begin} ></tn:BLOCK>
    <!-- Content -->
    Course
    <form th:action="@{/course/}" method="post">
        <div class="my-2">
            <label class="w-40 inline-block">Course Id:</label>
            <input type="text" name="course-id"
                   class="border border-blue-600 p-1 rounded w-80"
                   th:value="${course?.courseId?:'0'}" readonly
            />
        </div>
        <div class="my-2">
            <label class="w-40 inline-block">Course Name:</label>
            <input type="text" name="course-name"
                   class="border border-blue-600 p-1 rounded w-80"
                   th:value="${course?.courseName?:''}"
            />
        </div>
        <div class="my-2">
            <label class="w-40 inline-block">Course Description:</label>
            <input type="text" name="course-desc"
                   class="border border-blue-600 p-1 rounded w-80"
                   th:value="${course?.courseDescription?:''}"
            />
        </div>
        <div class="my-2 text-right">
            <button type="submit">
        </div>
    </form>
</body>
</html>
```
 - Status Bar:** Ln 9, Col 21 Spaces: 4 UTF-8 CRLF HTML

Thymeleaf - Template Course index.html Form

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho...
 - controller
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course**:
 - index.html
 - enroll
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
 - application.properties
- Central Area:** The code editor displays the content of `index.html`. The file path is `src > main > resources > templates > course > index.html`. The code uses Thymeleaf syntax to create a form for adding a course. A red box highlights the entire form structure from the opening `<form` tag to the closing `</form>` tag.
- Bottom Status Bar:** Ln 9, Col 21, Spaces: 4, UTF-8, CRLF, HTML, SP

Thymeleaf - Template Course index.html Table

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

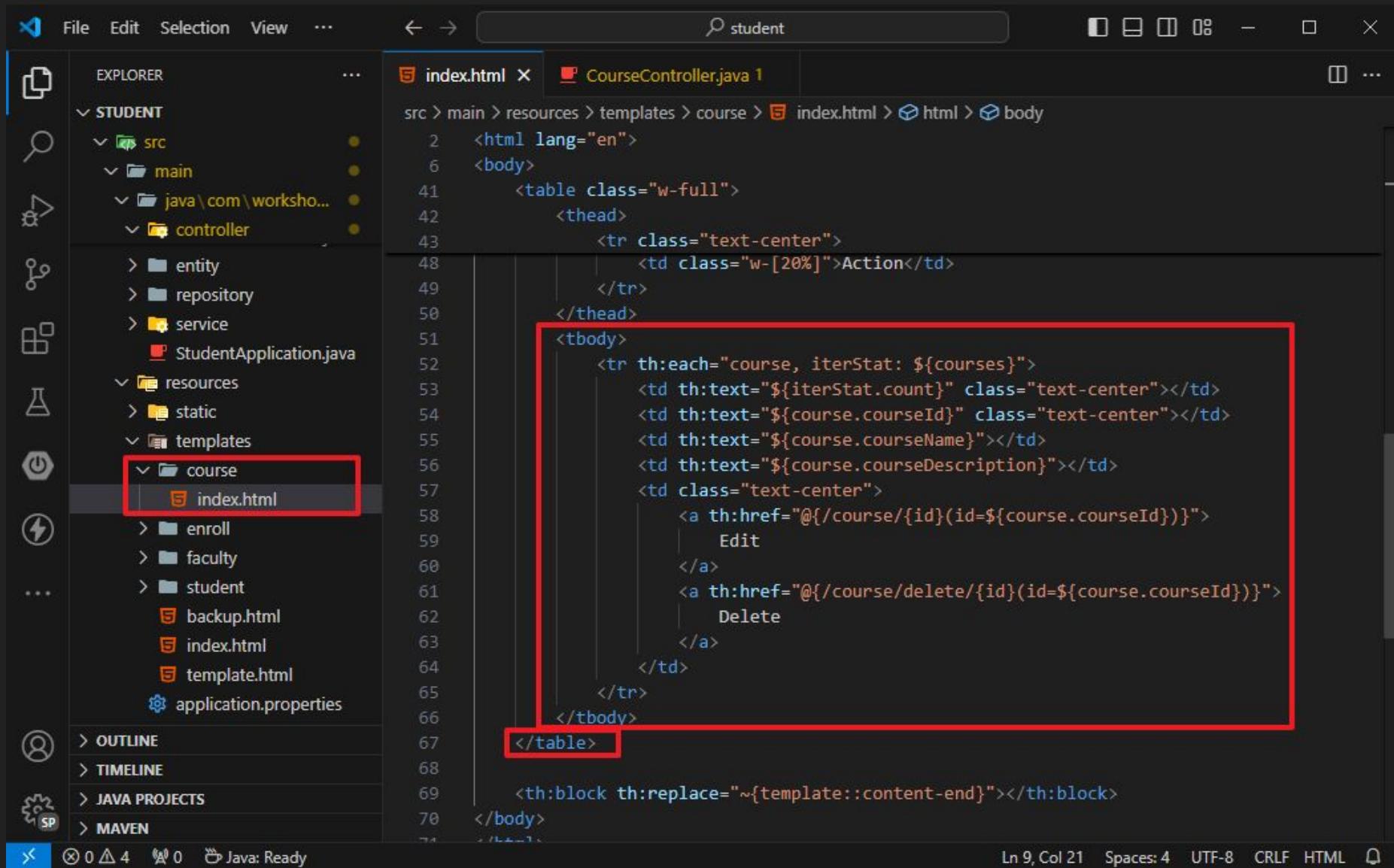
- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho...
 - controller
 - entity
 - repository
 - service
 - resources
 - static
 - templates
 - course
 - index.html
 - enroll
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
 - Bottom Status Bar:** Ln 9, Col 21 Spaces: 4 UTF-8 CRLF HTML

The code editor displays the `index.html` template file under the `course` directory. A red box highlights the table structure:

```
<table class="w-full">
    <thead>
        <tr class="text-center">
            <td class="w-[10%]">#</td>
            <td class="w-[10%]">Course ID</td>
            <td class="w-[30%]">Course Name</td>
            <td class="w-[30%]">Course Description</td>
            <td class="w-[20%]">Action</td>
        </tr>
    </thead>
    <tbody>
        <tr th:each="course, iterStat: ${courses}">
            <td th:text="${iterStat.count}" class="text-center"></td>
            <td th:text="${course.courseId}" class="text-center"></td>
            <td th:text="${course.courseName}"></td>
            <td th:text="${course.courseDescription}"></td>
            <td class="text-center">
                <a th:href="@{/course/{id}(id=${course.courseId})}">
                    Edit
                </a>
                <a th:href="@{/course/delete/{id}(id=${course.courseId})}">
                    Delete
                </a>
            </td>
        </tr>
    </tbody>

```

Thymeleaf - Template Course index.html Table



```
File Edit Selection View ... ⟲ ⟳ 🔎 student EXPLORE STUDENT index.html CourseController.java 1 src > main > resources > templates > course > index.html > html > body
  2   <html lang="en">
  6     <body>
41       <table class="w-full">
42         <thead>
43           <tr class="text-center">
44             <td class="w-[20%]">Action</td>
45           </tr>
46         </thead>
47         <tbody>
48           <tr th:each="course, iterStat: ${courses}">
49             <td th:text="${iterStat.count}" class="text-center"></td>
50             <td th:text="${course.courseId}" class="text-center"></td>
51             <td th:text="${course.courseName}"></td>
52             <td th:text="${course.courseDescription}"></td>
53             <td class="text-center">
54               <a th:href="@{/course/{id}(id=${course.courseId})}">
55                 Edit
56               </a>
57               <a th:href="@{/course/delete/{id}(id=${course.courseId})}">
58                 Delete
59               </a>
60             </td>
61           </tr>
62         </tbody>
63       </table>
64
65       <th:block th:replace="~{template::content-end}"></th:block>
66
67     </body>
68
69   <th:block th:replace="~{template::content-end}"></th:block>
70
71
```

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

- File Explorer (Left):** Shows the project structure under the "STUDENT" package. The "course" folder in "templates" contains "index.html", which is highlighted with a red box.
- Editor (Center):** Displays the content of "index.html". The code is written in Thymeleaf templating language. It includes a table structure with a header row and a body row that iterates over a list of courses. Each course row contains five columns: a "w-[20%]" width column for "Action", followed by four standard text columns for course ID, name, description, and a "text-center" column containing edit and delete links generated by Thymeleaf's href attribute.
- Search Bar (Top):** Contains the text "student".
- Toolbar (Top):** Includes standard file operations like File, Edit, View, and a search icon.
- Status Bar (Bottom):** Shows build statistics (0 errors, 0 warnings), Java status ("Java: Ready"), and file information (Ln 9, Col 21, Spaces: 4, UTF-8, CRLF, HTML).

Thymeleaf - Template Course Test

The screenshot shows a web browser window with two tabs: "localhost8080/course" and "H2 Console". The main content area displays a navigation bar with links: "Student Management", "Faculty", "Student", "Course" (which is highlighted in blue), and "Enroll". Below the navigation bar, there is a section titled "Course" with the word "Test" in red. This section contains three input fields labeled "Course Id:", "Course Name:", and "Course Description:", each with a placeholder value ("0", "", and "") respectively. To the right of these fields is a blue button labeled "Save Course". Below this form, a table is displayed with the following data:

#	Course ID	Course Name	Course Description	Action
1	54	Programming	OOP	Edit Delete

ThymeLeaf - Template / fragment / replace / insert

th:block / th:fragment ✓

th:replace / th:insert ✓

th:href + @ { ... } ✓ <https://www.thymeleaf.org/doc/articles/standardurlsyntax.html>

th:classappend ✓

th:text / th:each / th:value ✓

th:if / th:unless ✓

th:disabled <

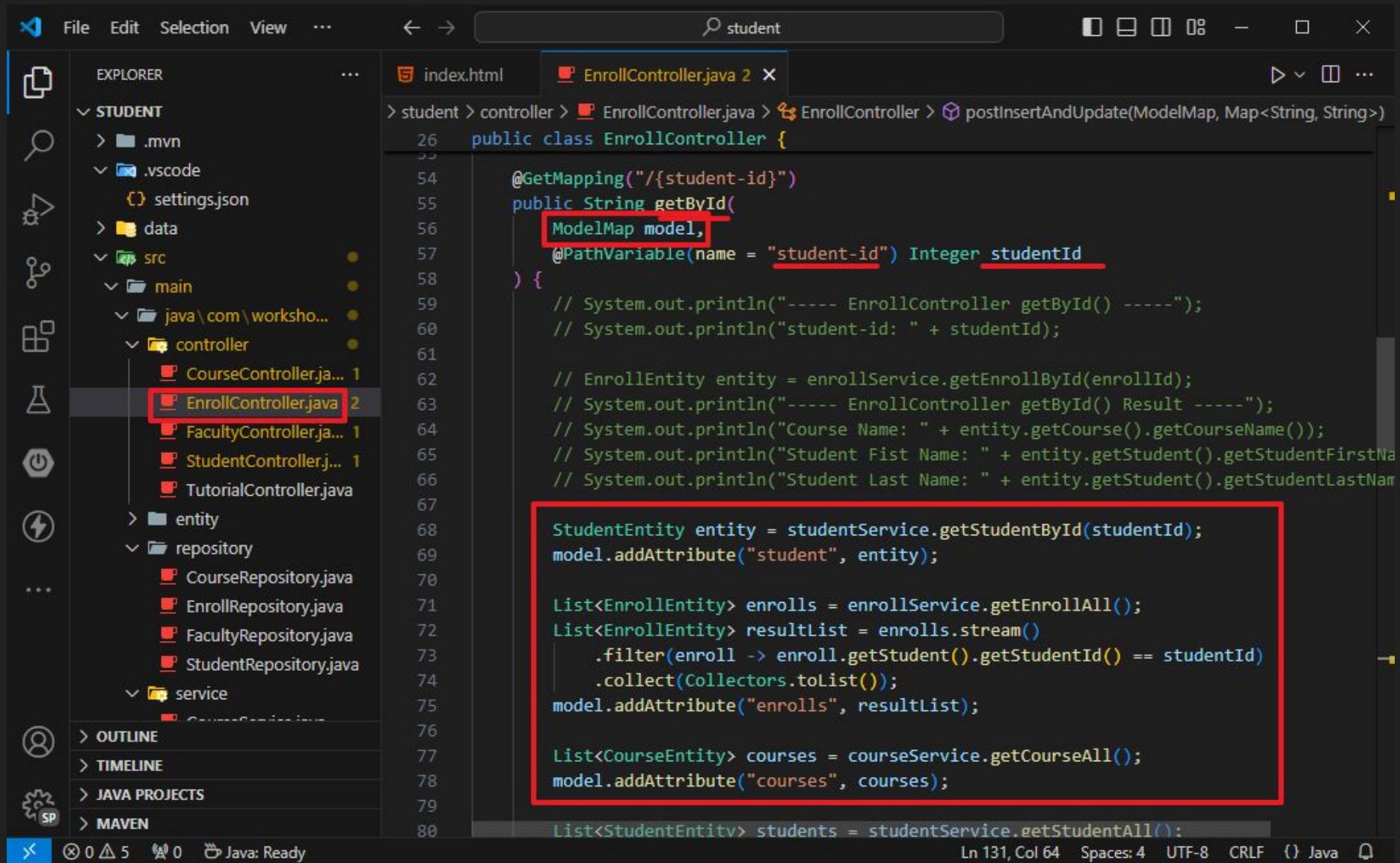
Thymeleaf - Template Enroll getAll()

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Explorer:** Shows the project structure under STUDENT:
 - .mvn
 - .vscode (with settings.json)
 - data
 - src
 - main
 - java\com\workshop...
 - controller
 - CourseController.java 1
 - EnrollController.java 2 (highlighted)
 - FacultyController.java 1
 - StudentController.java 1
 - TutorialController.java
 - entity
 - repository
 - CourseRepository.java
 - EnrollRepository.java
 - FacultyRepository.java
 - StudentRepository.java
 - service
 - EnrollService.java
 - Editor:** EnrollController.java 2 (highlighted)

```
public class EnrollController {  
    @Autowired  
    private EnrollService enrollService;  
  
    @GetMapping({ "", "/" })  
    public String getAll(ModelMap model) {  
        System.out.println("----- EnrollController getAll() -----");  
  
        // List<EnrollEntity> enrolls = enrollService.getEnrollAll();  
        // System.out.println("----- EnrollController getAll() Result -----");  
        // System.out.println("Size: " + enrolls.size());  
  
        List<CourseEntity> courses = courseService.getCourseAll();  
        model.addAttribute("courses", courses);  
  
        List<StudentEntity> students = studentService.getStudentAll();  
        model.addAttribute("students", students);  
  
        return "enroll/index";  
    }  
  
    @GetMapping("/{student-id}")  
    public String getById(  
        ModelMap model,  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        // System.out.println("----- EnrollController getById() -----");  
        // System.out.println("student-id: " + studentId);  
    }  
}
```
 - Bottom Status Bar:** 0 0 5 0 Java: Ready, Ln 131, Col 64, Spaces: 4, UTF-8, CRLF, Java

Thymeleaf - Template Enroll getById()



```
public class EnrollController {  
    @GetMapping("/{student-id}")  
    public String getById(  
        ModelMap model,  
        @PathVariable(name = "student-id") Integer studentId  
    ) {  
        // System.out.println("----- EnrollController getById() -----");  
        // System.out.println("student-id: " + studentId);  
  
        // EnrollEntity entity = enrollService.getEnrollById(enrollId);  
        // System.out.println("----- EnrollController getById() Result -----");  
        // System.out.println("Course Name: " + entity.getCourse().getCourseName());  
        // System.out.println("Student First Name: " + entity.getStudent().getStudentFirstName());  
        // System.out.println("Student Last Name: " + entity.getStudent().getStudentLastName());  
  
        StudentEntity entity = studentService.getStudentById(studentId);  
        model.addAttribute("student", entity);  
  
        List<EnrollEntity> enrolls = enrollService.getEnrollAll();  
        List<EnrollEntity> resultList = enrolls.stream()  
            .filter(enroll -> enroll.getStudent().getStudentId() == studentId)  
            .collect(Collectors.toList());  
        model.addAttribute("enrolls", resultList);  
  
        List<CourseEntity> courses = courseService.getCourseAll();  
        model.addAttribute("courses", courses);  
  
        List<StudentEntity> students = studentService.getStudentAll();  
    }  
}
```

Thymeleaf - Template Enroll getById()

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "STUDENT" folder. The "controller" folder contains several controller classes: CourseController.java, EnrollController.java (highlighted with a red border), FacultyController.java, StudentController.java, and TutorialController.java.
- Search Bar (Top):** Contains the text "student".
- Code Editor (Right):** Displays the content of EnrollController.java. The code implements a method to get a student by ID and returns a Thymeleaf template named "enroll/index". A specific section of the code, which retrieves all students and adds them to the model, is highlighted with a red box.
- Bottom Status Bar:** Shows the following information: Line 131, Column 64; Spaces: 4; UTF-8; CRLF; Java; and a file icon.

```
public class EnrollController {  
    public String getById(  
        // EnrollmentEntity entity = enrollService.getEnrollById(enrollId);  
        // System.out.println("---- EnrollController getById() Result ----");  
        // System.out.println("Course Name: " + entity.getCourse().getCourseName());  
        // System.out.println("Student Fist Name: " + entity.getStudent().getStudentFirstName());  
        // System.out.println("Student Last Name: " + entity.getStudent().getStudentLastName());  
  
        StudentEntity entity = studentService.getStudentById(studentId);  
        model.addAttribute("student", entity);  
  
        List<EnrollEntity> enrolls = enrollService.getEnrollAll();  
        List<EnrollEntity> resultList = enrolls.stream()  
            .filter(enroll -> enroll.getStudent().getStudentId() == studentId)  
            .collect(Collectors.toList());  
        model.addAttribute("enrolls", resultList);  
  
        List<CourseEntity> courses = courseService.getCourseAll();  
        model.addAttribute("courses", courses);  
  
        List<StudentEntity> students = studentService.getStudentAll();  
        model.addAttribute("students", students);  
  
        return "enroll/index";  
    }  
  
    @GetMapping("/delete/{enroll-id}")  
    public String getDeleteById(  
        ModelMap model  
    ) {  
        return "enroll/index";  
    }  
}
```

Thymeleaf - Template Enroll getDeleteById()

The screenshot shows a Java code editor in VS Code with the file `EnrollController.java` open. The code implements a controller for managing enrollments. A red box highlights the method `getDeleteById`, which takes an `enrollId` as a path variable and calls the `enrollService.deleteEnrollById` service method. Another red box highlights the logic for populating the Thymeleaf model with course and student lists.

```
public class EnrollController {
    public String getById() {
        return "enroll/index";
    }

    @GetMapping("/delete/{enroll-id}")
    public String getDeleteById(
        ModelMap model,
        @PathVariable(name = "enroll-id") Integer enrollId
    ) {
        // System.out.println("---- EnrollController getDeleteById() -----");
        // System.out.println("enroll-id: " + enrollId);

        // System.out.println("---- EnrollController getDeleteById() Result -----");
        enrollService.deleteEnrollById(enrollId);

        List<CourseEntity> courses = courseService.getCourseAll();
        model.addAttribute("courses", courses);

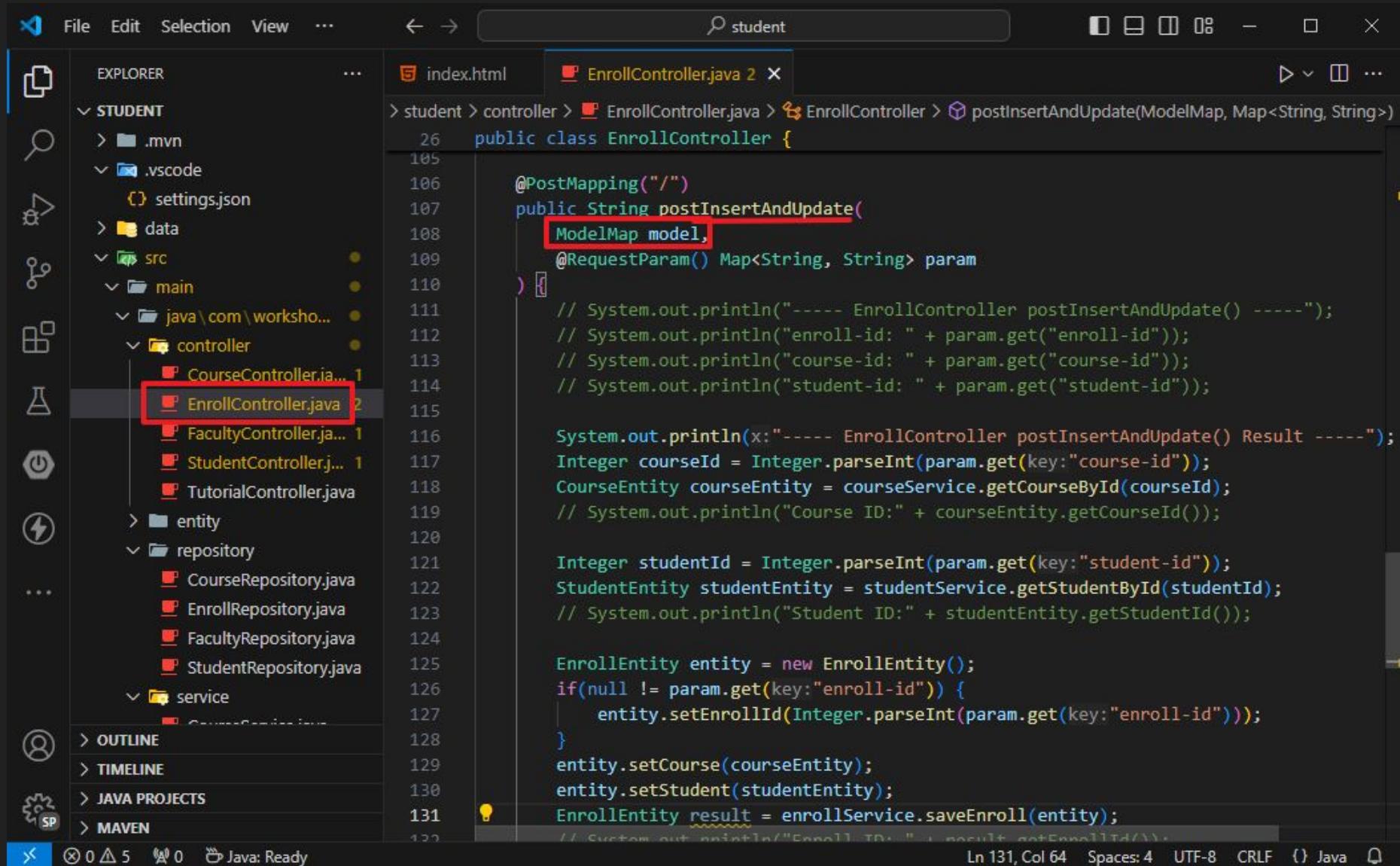
        List<StudentEntity> students = studentService.getStudentAll();
        model.addAttribute("students", students);

        return "enroll/index";
    }

    @PostMapping("/")
    public String postInsertAndUpdate(
        ModelMap model,
        @RequestParam() Map<String, String> param
    ) {
        String name = param.get("name");
        String id = param.get("id");
        String course = param.get("course");
        String student = param.get("student");

        if (name != null && !name.isEmpty()) {
            courseService.insertCourse(name);
        }
        if (id != null && !id.isEmpty()) {
            studentService.insertStudent(id);
        }
        if (course != null && !course.isEmpty()) {
            studentService.insertStudent(course);
        }
        if (student != null && !student.isEmpty()) {
            courseService.insertCourse(student);
        }
    }
}
```

Thymeleaf - Template Enroll postInsertAndUpdate()



The screenshot shows a Java code editor in VS Code with the following details:

- File Explorer (Left):** Shows the project structure under "STUDENT". The "src/main/java/com/workshop/controller" package contains several controller classes: CourseController.java, FacultyController.java, StudentController.java, TutorialController.java, and EnrollController.java. The EnrollController.java file is currently selected and highlighted with a red border.
- Editor Area (Center):** Displays the code for the EnrollController.java file. The method `postInsertAndUpdate` is highlighted with a red box. The code uses Thymeleaf-style syntax for parameter extraction and service layer interaction.
- Bottom Status Bar:** Shows build status (0 errors, 0 warnings), Java environment ("Java: Ready"), and other system information like line and column numbers (Ln 131, Col 64).

```
public class EnrollController {  
    @PostMapping("/")  
    public String postInsertAndUpdate(  
        ModelMap model,  
        @RequestParam() Map<String, String> param  
    ) {  
        // System.out.println("---- EnrollController postInsertAndUpdate() ----");  
        // System.out.println("enroll-id: " + param.get("enroll-id"));  
        // System.out.println("course-id: " + param.get("course-id"));  
        // System.out.println("student-id: " + param.get("student-id"));  
  
        System.out.println(x:"---- EnrollController postInsertAndUpdate() Result ----");  
        Integer courseId = Integer.parseInt(param.get(key:"course-id"));  
        CourseEntity courseEntity = courseService.getCourseById(courseId);  
        // System.out.println("Course ID: " + courseEntity.get courseId());  
  
        Integer studentId = Integer.parseInt(param.get(key:"student-id"));  
        StudentEntity studentEntity = studentService.getStudentById(studentId);  
        // System.out.println("Student ID: " + studentEntity.getStudentId());  
  
        EnrollEntity entity = new EnrollEntity();  
        if(null != param.get(key:"enroll-id")) {  
            entity.setEnrollId(Integer.parseInt(param.get(key:"enroll-id")));  
        }  
        entity.setCourse(courseEntity);  
        entity.setStudent(studentEntity);  
        EnrollEntity result = enrollService.saveEnroll(entity);  
        // Custom out.println("Enroll ID: " + result.getEnrollId());  
    }  
}
```

Thymeleaf - Template Enroll postInsertAndUpdate()

File Edit Selection View ... ⟲ ⟳ 🔎 student

EXPLORER

STUDENT

- .mvn
- .vscode
 - settings.json
- data
- src
 - main
 - java\com\workshop...
 - controller
 - CourseController.java 1
 - EnrollController.java 2
 - FacultyController.java 1
 - StudentController.java 1
 - TutorialController.java
 - entity
 - repository
 - CourseRepository.java
 - EnrollRepository.java
 - FacultyRepository.java
 - StudentRepository.java
 - service
 - CourseService.java

OUTLINE

TIMELINE

JAVA PROJECTS

MAVEN

index.html

EnrollController.java 2

```
src > main > java > com > workshop > student > controller > EnrollController.java > EnrollController.java
```

```
26 public class EnrollController {  
107     public String postInsertAndUpdate(  
119         // System.out.println("Course ID: " + courseEntity.getcourseId());  
120  
121         Integer studentId = Integer.parseInt(param.get(key:"student-id"));  
122         StudentEntity studentEntity = studentService.getStudentById(studentId);  
123         // System.out.println("Student ID: " + studentEntity.getStudentId());  
124  
125         EnrollEntity entity = new EnrollEntity();  
126         if(null != param.get(key:"enroll-id")) {  
127             entity.setEnrollId(Integer.parseInt(param.get(key:"enroll-id")));  
128         }  
129         entity.setCourse(courseEntity);  
130         entity.setStudent(studentEntity);  
131         EnrollEntity result = enrollService.saveEnroll(entity);  
132         // System.out.println("Enroll ID: " + result.getEnrollId());  
133         // System.out.println("Course Name: " + result.getCourse().getCourseName());  
134         // System.out.println("Student Code: " + result.getStudent().getStudentCode());  
135  
136         List<CourseEntity> courses = courseService.getCourseAll();  
137         model.addAttribute("courses", courses);  
138  
139         List<StudentEntity> students = studentService.getStudentAll();  
140         model.addAttribute("students", students);  
141  
142         return "enroll/index";  
143     }
```

Ln 144, Col 1 Spaces: 4 UTF-8 CRLF {} Java

Thymeleaf - Template Enroll index.html Form

The screenshot shows an IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho...
 - service
 - FacultyService.java
 - StudentService.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - index.html** (highlighted with a red box)
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Central Area:** Shows the content of **index.html**. The code is:

```
<html lang="en">
<body>
    <!-- Content -->
    Enroll
    <form th:action="@{/enroll/}" method="post">
        <div class="grid grid-cols-2">
            <div>
                <label class="w-40 inline-block">Enroll Id:</label>
                <input type="text" name="enroll-id"
                    class="border border-blue-600 p-1 rounded w-80"
                    th:value="${enroll?.enrollId?:'0'}" readonly
                />
            </div>
        </div>
        <div class="my-2 grid grid-cols-2">
            <div>
                <label class="w-40 inline-block">Student:</label>
                <input type="hidden" name="student-id"
                    th:value="${student?.studentId?:''}"
                />
                <input type="text" name="student-name" readonly
                    class="border border-blue-600 p-1 rounded w-80"
                    th:value="${student?.studentCode?:''}"
                />
            </div>
            <div>
                <label class="w-40 inline-block">Course:</label>
                <select name="course-id">
```
- Bottom Status Bar:** L0, Col 21, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template Enroll index.html Form

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\worksho...
 - service
 - FacultyService.java
 - StudentService.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - index.html
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Central Area:** Shows the content of `index.html`. The code is highlighted with syntax coloring for Thymeleaf and HTML. A red box highlights the section of the code that handles student and course selection.
- Bottom Status Bar:** Shows build errors (0), Java status (Ready), and system information (Ln 9, Col 21, Spaces: 4, UTF-8, CRLF, HTML).

```
<html lang="en">
<body>
    <form th:action="@{/enroll/}" method="post">
        <div class="grid grid-cols-2">
            </div>
            <div class="my-2 grid grid-cols-2">
                <div>
                    <label class="w-40 inline-block">Student:</label>
                    <input type="hidden" name="student-id"
                           th:value="${student?.studentId?:''}">
                </div>
                <div>
                    <label class="w-40 inline-block">Course:</label>
                    <select name="course-id"
                           class="border border-blue-600 p-1 rounded w-80">
                        <option th:each="course: ${courses}"
                               th:value="${course.courseId}"
                               th:text="${course.courseName}"></option>
                    </select>
                </div>
            </div>
            <div class="my-2 text-right">
                <button type="submit" th:disabled="${student?.null}">Submit</button>
            </div>
        </div>
    </form>
</body>
</html>
```

Thymeleaf - Template Enroll index.html Form

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

- File Explorer (Left):** Shows the project structure under the 'STUDENT' package:
 - src**: Contains main, java, and service packages.
 - resources**: Contains static, templates, course, and enroll subfolders. The 'enroll' folder is highlighted with a red box.
 - FacultyService.java, StudentService.java, and StudentApplication.java are listed under the main package.
 - application.properties is also listed.
- Code Editor (Right):** Displays the content of 'index.html'. The code uses Thymeleaf syntax to create a form for enrolling a student. It includes fields for 'studentCode' and 'course', and a 'submit' button.

```
<html lang="en">
<body>
    <form th:action="@{/enroll/}" method="post">
        <div class="my-2 grid grid-cols-2">
            <div>
                th:value="${student?.studentCode?:''}"
            </div>
            <div>
                <label class="w-40 inline-block">Course:</label>
                <select name="course-id" class="border border-blue-600 p-1 rounded w-80">
                    <option th:each="course: ${courses}" th:value="${course.courseId}" th:text="${course.courseName}"></option>
                </select>
            </div>
        </div>
        <div class="my-2 text-right">
            <button type="submit" th:disabled="${student == null}" class="text-white font-bold bg-blue-600 py-2 px-4 rounded">
                Enroll
            </button>
        </div>
    </form>
```
- Status Bar (Bottom):** Shows build errors (0), Java status (Ready), and other system information like line and column numbers.

Thymeleaf - Template Enroll index.html Table

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

- File Explorer (Left):** Shows a project structure under "STUDENT". The "enroll" folder contains "index.html", which is highlighted with a red box.
- Editor Area (Center):** Displays the content of "index.html".

```
<html lang="en">
<body>
    Student List
    <table class="w-full">
        <thead>
            <tr class="text-center">
                <td class="w-[10%]">#</td>
                <td class="w-[10%]">Student ID</td>
                <td class="w-[10%]">Student Code</td>
                <td class="w-[20%]">Student First Name</td>
                <td class="w-[20%]">Student Last Name</td>
                <td class="w-[20%]">Faculty Name</td>
                <td class="w-[10%]">Action</td>
            </tr>
        </thead>
        <tbody>
            <tr th:each="student, iterStat: ${students}">
                <td th:text="${iterStat.count}" class="text-center"></td>
                <td th:text="${student.studentId}" class="text-center"></td>
                <td th:text="${student.studentCode}"></td>
                <td th:text="${student.studentFirstName}"></td>
                <td th:text="${student.studentLastName}"></td>
                <td th:text="${student?.faculty?.facultyName?: ''}"></td>
                <td class="text-center">
                    <a th:href="@{/enroll/{id}(id=${student.studentId})}">
                        View Course
                    </a>
                </td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```
- Search Bar (Top):** Contains the text "student".
- Bottom Status Bar:** Shows file statistics: Ln 9, Col 21, Spaces: 4, UTF-8, CRLF, HTML.

Thymeleaf - Template Enroll index.html Table

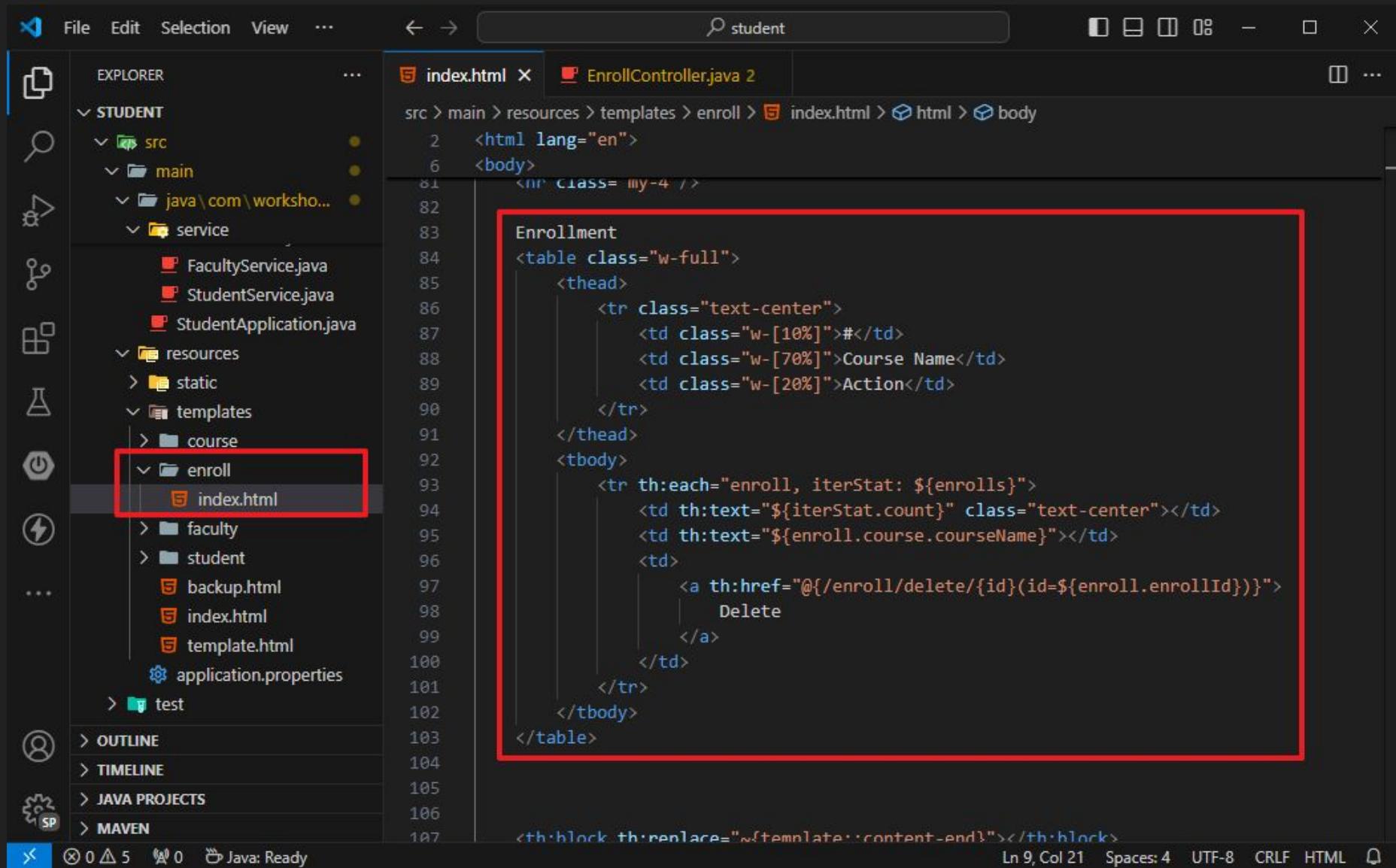
The screenshot shows a code editor interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho...
 - service
 - FacultyService.java
 - StudentService.java
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - index.html
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Central Area:** The file `index.html` is open, showing Thymeleaf template code for generating an enrollment table. A red box highlights the section of code that iterates over students and generates table rows.

```
<html lang="en">
<body>
    <table class="w-full">
        <thead>
            <tr class="text-center">
                <td class="w-[10%]">Action</td>
            </tr>
        </thead>
        <tbody>
            <tr th:each="student, iterStat: ${students}">
                <td th:text="${iterStat.count}" class="text-center"></td>
                <td th:text="${student.studentId}" class="text-center"></td>
                <td th:text="${student.studentCode}"></td>
                <td th:text="${student.studentFirstName}"></td>
                <td th:text="${student.studentLastName}"></td>
                <td th:text="${student?.faculty?.facultyName?: ''}"></td>
                <td class="text-center">
                    <a th:href="@{/enroll/{id}(id=${student.studentId})}">
                        View Course
                    </a>
                </td>
            </tr>
        </tbody>
    </table>
    <hr class="my-4"/>

```
- Bottom Status Bar:** Ln 9, Col 21, Spaces: 4, UTF-8, CRLF, HTML, Q

Thymeleaf - Template Enroll index.html Table



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

- File Menu:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbar:** Minimize, Maximize, Close
- Left Sidebar (EXPLORER):**
 - STUDENT** folder
 - src
 - main
 - java\com\worksho... (FacultyService.java, StudentService.java, StudentApplication.java)
 - resources
 - static
 - templates
 - course
 - enroll
 - index.html (highlighted by a red box)
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
 - application.properties
 - test
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Editor Area:** Shows the file **index.html** with the following content:

```
2 <html lang="en">
6 <body>
81     <div class="my-4" />
82
83     Enrollment
84     <table class="w-full">
85         <thead>
86             <tr class="text-center">
87                 <td class="w-[10%]">#</td>
88                 <td class="w-[70%]">Course Name</td>
89                 <td class="w-[20%]">Action</td>
90             </tr>
91         </thead>
92         <tbody>
93             <tr th:each="enroll, iterStat: ${enrolls}">
94                 <td th:text="${iterStat.count}" class="text-center"></td>
95                 <td th:text="${enroll.course.courseName}"></td>
96                 <td>
97                     <a th:href="@{/enroll/delete/{id}(id=${enroll.enrollId})}">
98                         Delete
99                     </a>
100                </td>
101            </tr>
102        </tbody>
103    </table>
104
105
106
107 <th:block th:replace="~{template::content-end}"></th:block>
```
- Status Bar:** Ln 9, Col 21 Spaces: 4 UTF-8 CRLF HTML

Thymeleaf - Template Enroll Test

The screenshot shows a web application interface for student enrollment. At the top, there is a navigation bar with tabs: Student Management, Faculty, Student, Course, and Enroll. The Enroll tab is currently active, indicated by a dark blue background.

The main content area has a title "Test" centered above form fields. The form includes:

- An "Enroll" button on the left.
- A text input field for "Enroll Id" containing the value "0".
- A dropdown menu for "Student" with the value "002".
- A dropdown menu for "Course" with the value "Programming".
- A large blue "Enroll" button on the right.

Below the form is a section titled "Student List" which displays a table of student information:

#	Student ID	Student Code	Student First Name	Student Last Name	Faculty Name	Action
1	70	001	First	Last	Computer	View Course
2	71	002	First 2 Edit	Last 2	Law	View Course

At the bottom is a section titled "Enrollment" which displays a table of course enrollments:

#	Course Name	Action
1	Programming	Delete
2	Thai Language	Delete

ThymeLeaf - Template / fragment / replace / insert

th:block / th:fragment ✓

th:replace / th:insert ✓

th:href + @ { ... } ✓ <https://www.thymeleaf.org/doc/articles/standardurlsyntax.html>

th:classappend ✓

th:text / th:each / th:value ✓

th:if / th:unless ✓

th:disabled ✓

th:scr / th:href (static file) <

Thymeleaf - Template Static File

The screenshot shows a Java application structure in the Explorer view and its corresponding code in the main editor.

File Structure (EXPLORER):

- STUDENT
- > data
- < src
 - main
 - java\com\workshop\controller
 - CourseController.java
 - EnrollController.java
 - FacultyController.java
 - StudentController.java
 - TutorialController.java (highlighted with a red box)
 - entity
 - repository
 - service
 - StudentApplication.java
 - resources
 - static
 - templates
 - course
 - enroll
 - faculty

Code Editor:

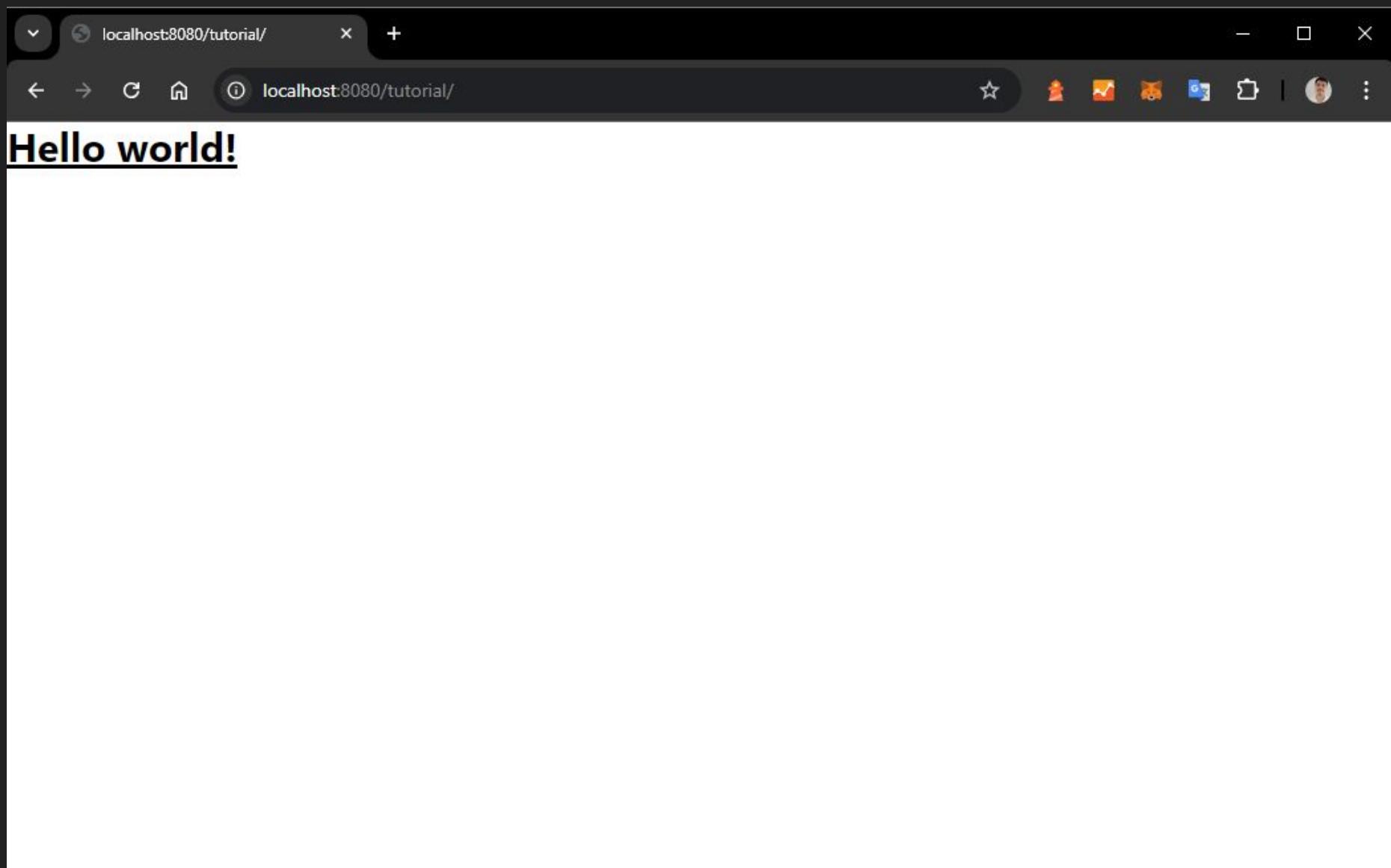
```
student
TutorialController.java index.html
src > main > java > com > workshop > student > controller > TutorialController.java > TutorialController
```

```
12
13     @Controller
14     @RequestMapping("/tutorial")
15     public class TutorialController {
16
17         @GetMapping("/")
18         public String getTutorial(
19             @RequestParam(name = "id", required = false, defaultValue = "0") Integer id
20         ) {
21             System.out.println("---- getTutorial ----");
22             System.out.println("ID: " + id);
23             return "index";
24         }
25
26         @GetMapping("/{id}")
27         public String getTutorialPath(
28             @PathVariable(name = "id") Integer id
29         ) {
30             System.out.println("---- getTutorialPath ----");
31             System.out.println("ID: " + id);
32             return "index";
33         }
34
35         @PostMapping("/")
36         public String postTutorial(
37             @RequestParam() Map<String, String> param
38         ) {
39             System.out.println("---- postTutorial ----");

```

Ln 25, Col 1 Spaces: 4 UTF-8 CRLF {} Java

Thymeleaf - Template Static File



Thymeleaf - Template Static File

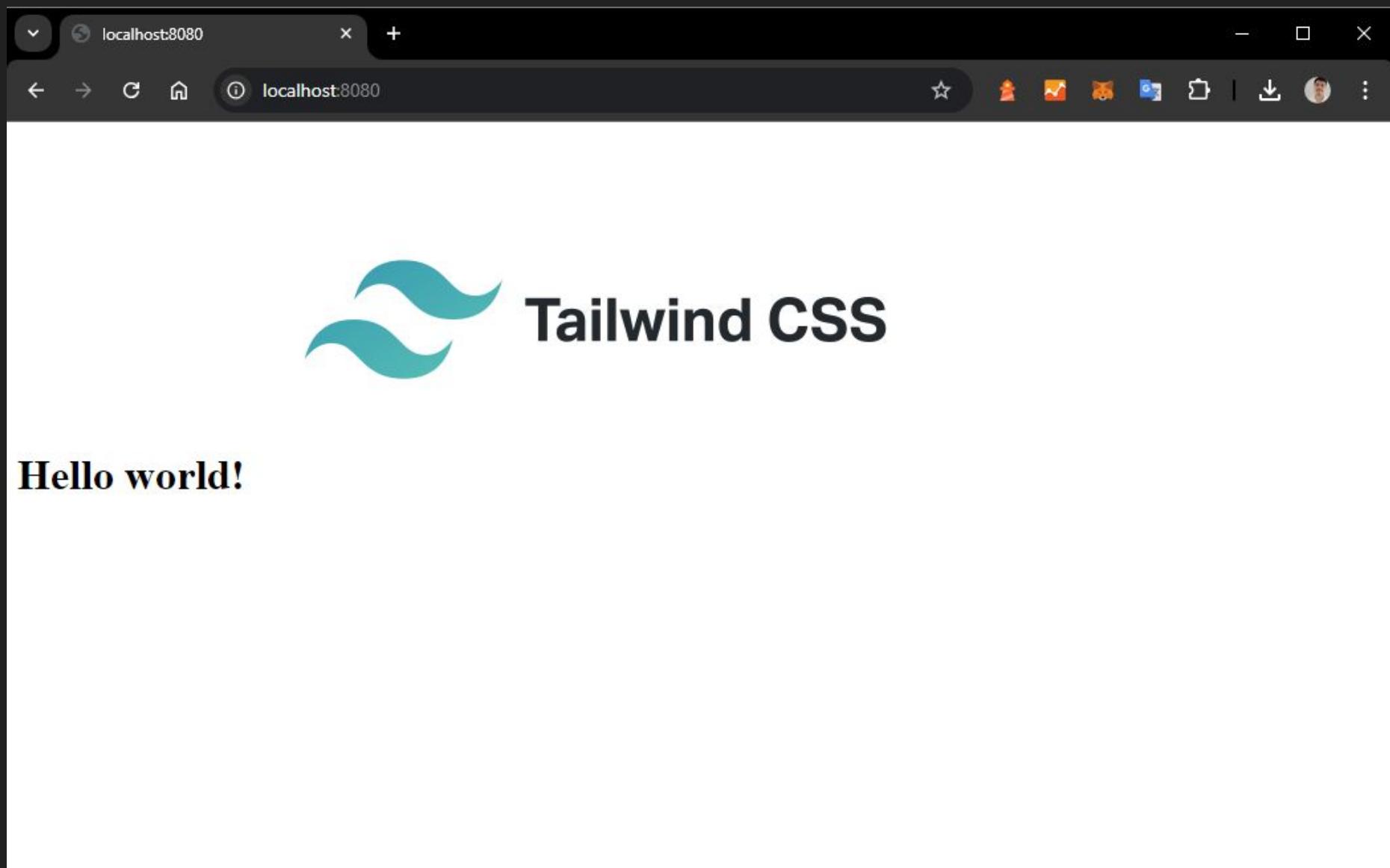
The screenshot shows a Java IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT** folder:
 - src
 - main
 - java\com\worksho... (containing StudentApplication.java)
 - resources folder:
 - static folder (containing images, scripts, styles)
 - tailwind-logo.png (highlighted with a red box)
 - templates folder (containing course, enroll, faculty, student, backup.html, index.html, template.html, application.properties)
 - test folder
 - OUTLINE, TIMELINE, JAVA PROJECTS, MAVEN sections.
- Central Area:**
 - TutorialController.java** tab (redacted)
 - index.html** tab (highlighted with a red box)
 - Code Editor:**

```
src > main > resources > templates > index.html > html
1   <!doctype html>
2   <html>
3     <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     </head>
7     <body>
8       <h1 class="text-3xl font-bold underline">
9         Hello world!
10        
11      </h1>
12    </body>
13  </html>
```

A red box highlights the `tailwind-logo.png` file in the resources/static/images directory. A red arrow points from this box to the corresponding Thymeleaf `th:src` attribute in the `` tag in the code editor.
- Bottom Status Bar:** Line 13, Col 8, Spaces: 2, UTF-8, CRLF, HTML, etc.

Thymeleaf - Template Static File



Thymeleaf - Template Static File <script>

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar:** Explorer, Search, Find, Open, Project, Help, ...
- Project Explorer (STUDENT folder):**
 - src
 - main
 - java\com\worksho... (StudentApplication.java)
 - resources
 - static
 - images (tailwind-logo.png)
 - scripts (main.js)
 - styles
 - templates
 - course
 - enroll
 - faculty
 - student
 - backup.html
 - index.html
 - template.html
- Code Editor:** The file main.js is open, showing the code:

```
1 alert("script alert")
```

The file path is shown in the status bar: src > main > resources > static > scripts > main.js
- Status Bar:** Ln 1, Col 22 Spaces: 4 UTF-8 CRLF {} JavaScript

Thymeleaf - Template Static File <script>

The screenshot shows a Java-based application structure and its corresponding Thymeleaf template.

Project Explorer: The project is named "STUDENT". It contains a "src" folder with "main" and "java" packages. The "main" package contains "StudentApplication.java". Inside "src/main/resources/templates", there are files: "index.html", "template.html", and "backup.html". The "static" folder under "resources" contains "images" (with "tailwind-logo.png") and "scripts" (with "main.js").

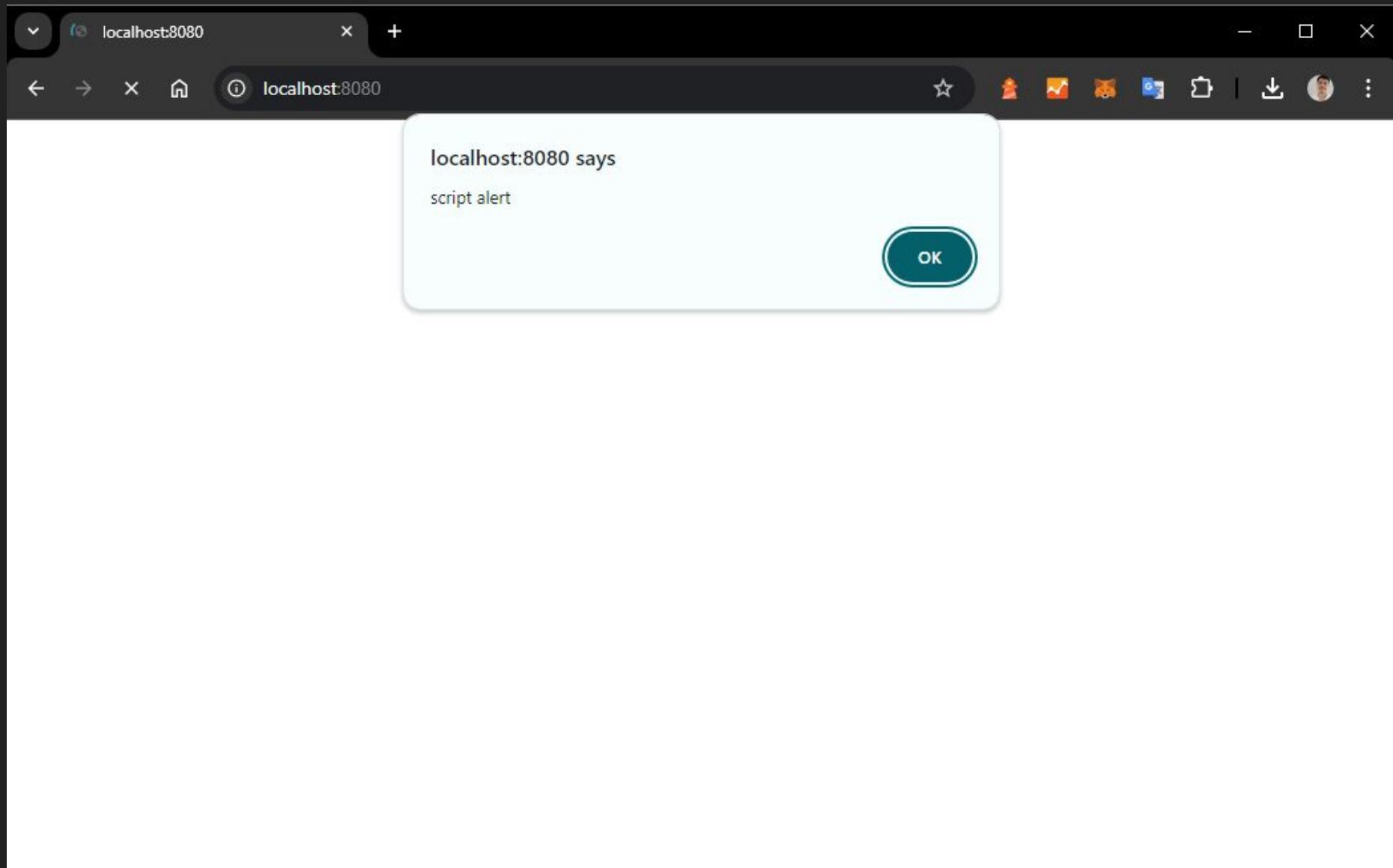
Code Editor: The "index.html" file is open. It includes a meta tag for viewport and a script tag that loads "main.js" from the "scripts" folder in the static resources.

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script th:src="@{/scripts/main.js}"></script>
</head>
<body>
    <h1 class="text-3xl font-bold underline">
        Hello world!
    
    </h1>
</body>
</html>
```

A red box highlights the "scripts" folder in the static resources, and a red arrow points from this box to the "th:src" attribute in the script tag of the template, indicating the source of the static file.

Status Bar: The status bar at the bottom shows "Java: Ready" and other standard IDE status indicators.

Thymeleaf - Template Static File <script>



Thymeleaf - Template Static File <style>

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

- File Explorer (Left):** Shows a project structure under "STUDENT".
 - src:** Contains "main" and "java/com/workshop...".
 - resources:** Contains "static" which further contains "images" (with "tailwind-logo.png"), "scripts" (with "main.js"), and "styles" (with "main.css").
 - templates:** Contains "course", "enroll", "faculty", "student", "backup.html", "index.html", and "template.html".
- Editor (Right):** Displays the content of "main.css".

```
h1 {  
    color: darkgreen;  
}
```

The code is highlighted with a red box. The file path is shown as "src > main > resources > static > styles > main.css".
- Bottom Status Bar:** Shows "Ln 3, Col 2" and other system status like "Java: Ready".

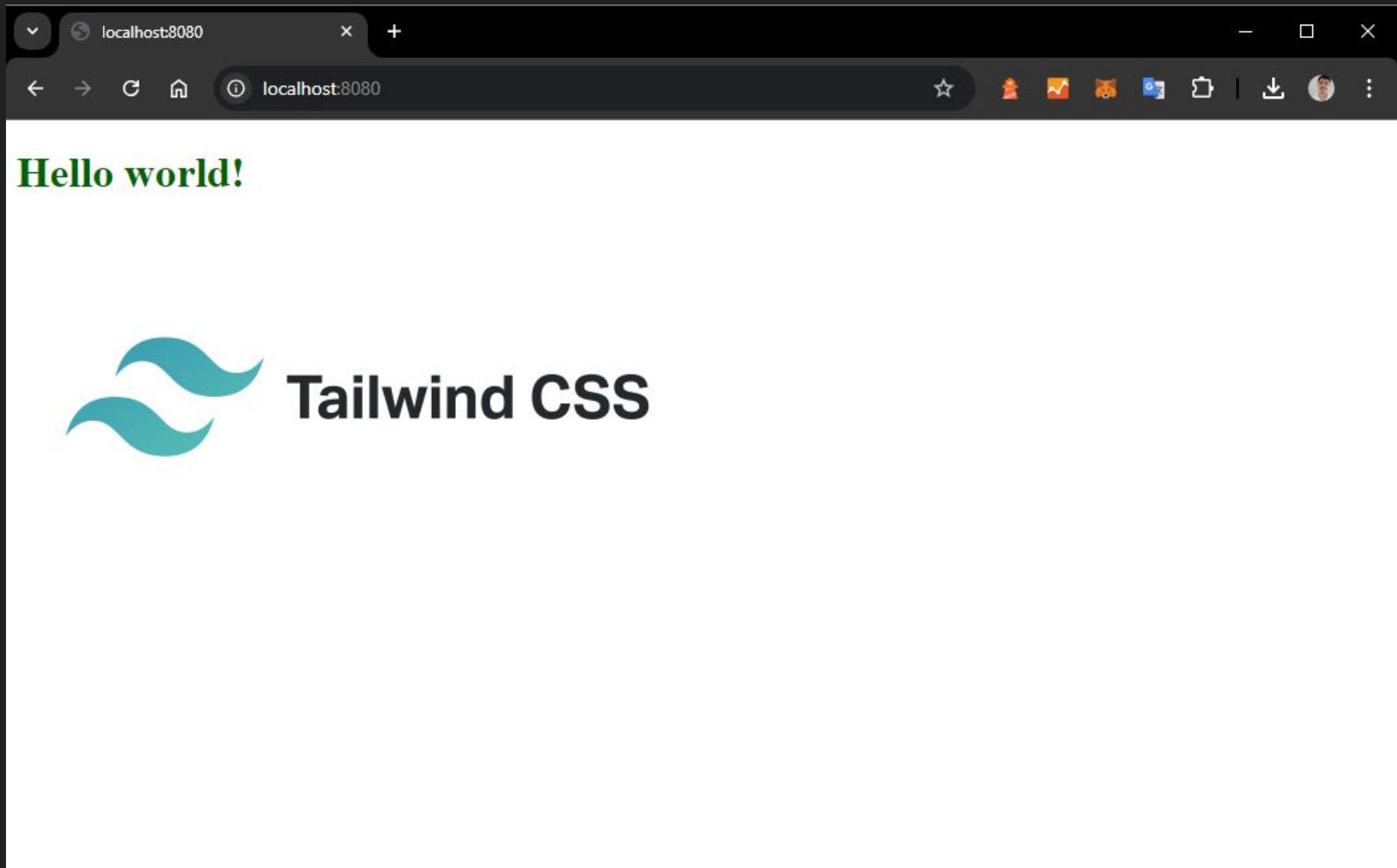
Thymeleaf - Template Static File <style>

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

- File Bar:** File, Edit, Selection, View, ...
- Search Bar:** student
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - STUDENT**:
 - src
 - main
 - java\com\worksho... (StudentApplication.java)
 - resources
 - static
 - images (tailwind-logo.png)
 - scripts (main.js)
 - styles (main.css)
 - templates
 - course
 - enroll
 - faculty
 - student
 - backup.html
 - index.html (selected)
 - template.html
- Central Area:** Code editor showing index.html and main.css files.
- Code Editor:**

```
src > main > resources > templates > index.html > html
1  <!doctype html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <script th:src="@{/scripts/main.js}"></script>
7      <link rel="stylesheet" th:href="@{/styles/main.css}">
8  </head>
9  <body>
10     <h1 class="text-3xl font-bold underline">
11         Hello world!
12     </h1>
13     
14 </body>
15 </html>
```
- Annotations:** A red box highlights the `link` element in the head section of index.html, and a red arrow points from the `main.css` file in the Explorer to the `link` element.
- Bottom Status Bar:** Ln 15, Col 8, Spaces: 2, UTF-8, CRLF, HTML, SP

Thymeleaf - Template Static File <style>



Workshop To-Do List

Requirement

- User can Management Topic (add/edit/remove)
- User can Management List in Topic (add/edit/remove)

UI Example