EgoTechWorld.com

Java Spring Boot Complete CRUD Tutorial

with HTML & Bootstrap 5

LEVEL	FOCUS	DURATION
Beginner	Step-by-step CRUD	2 to 3 hours

A complete beginner-friendly guide to building your first web application backend with Java Spring Boot.

Visit egotechworld.com for more tutorials

What You Will Learn

By the end of this tutorial, you will have built a fully functional CRUD (Create, Read, Update, Delete) web application using Java Spring Boot for the backend and Bootstrap 5 for the frontend. We will build a simple Student Management System where you can add students, view all students, edit them, and delete them.

Tutorial Topics

- Why Java and why Spring Boot? Real reasons explained simply
- Installing Java JDK, Maven, and an IDE on your computer
- Creating your first Spring Boot project with Spring Initializr
- Understanding the project folder structure
- Building the database connection (using H2 for easy testing)
- Creating the Entity, Repository, Service, and Controller layers
- Designing clean Bootstrap 5 pages for forms and tables
- Wiring HTML templates to Java backend with Thymeleaf
- Testing all four CRUD operations end-to-end
- Common errors and how to fix them

Who This Tutorial Is For

This tutorial is written for absolute beginners. You only need to know basic HTML and have a working computer. No prior Java experience is required, but knowing what variables and functions are will help you move faster. Every concept is explained in plain English before any code appears.

What You Will Build

A Student Management web app: a homepage that lists all students in a Bootstrap table, an Add Student form, an Edit page, and a Delete button. Everything saves to a database and survives restarts. Roughly 200 lines of Java total — small enough to understand fully, big enough to feel real.

Part 1: Why Java and Why Spring Boot?

Before writing a single line of code, it helps to know why you are picking this technology. There are many languages and frameworks out there. So why Java?

Why Java is Worth Learning

Java has been one of the most used programming languages in the world for over two decades, and it is not slowing down. Here are the real-world reasons people and companies pick Java today:

Massive Job Market

Banks, insurance companies, telecoms, government systems, and big tech companies run their critical software on Java. If you want a stable IT career, Java skills open more doors than almost any other language.

Write Once, Run Anywhere

Java code compiles to bytecode that runs on any computer with a Java Runtime. Your program works the same on Windows, Linux, or a server in the cloud.

Strong Type System

Java forces you to declare types. This catches mistakes at compile time, before users see them. For large applications with many developers, this saves hours of debugging.

Huge Ecosystem

Whatever you need to build (web apps, mobile apps, big data, banking systems, Android apps), there is a mature Java library for it.

Great Performance

Java is fast. The JVM (Java Virtual Machine) optimizes your code while it runs. For high-traffic websites and enterprise systems, this matters a lot.

Why Spring Boot Specifically?

Plain Java is powerful but verbose. To build a website in plain Java, you would write hundreds of lines of setup code before even displaying "Hello World" in a browser. This is where Spring Boot comes in.

Spring Boot is a framework that makes building Java web applications fast and simple. It removes most of the boring setup so you can focus on the actual features your app needs.

Spring Boot Advantages for Beginners

Auto-Configuration

Spring Boot looks at what is in your project and configures most things automatically. You do not write XML config files like in old Spring.

Embedded Server

It comes with a built-in web server (Tomcat). You do not need to install or set up a separate server. Just run your Java file and your website is live.

Starter Dependencies

Need a database? Add one line. Need security? Add one line. Spring Boot groups related libraries into starter packs that just work together.

Production Ready

Health checks, monitoring, metrics — all built in. The same code you build as a beginner is the same architecture used by huge companies.

Massive Community

Stuck on a problem? Search it on Google or Stack Overflow. Almost every issue you will hit has been answered hundreds of times.

> **Compared to PHP or Python**
> If you have used PHP (Laravel) or Python (Django), Spring Boot is the Java equivalent — same idea, different language. Java is more strict and verbose but generally faster and used in larger enterprise systems. Learning it adds a serious skill to your CV.

Part 2: Installing Everything You Need

Before we write code, we need three things on your computer: the Java Development Kit (JDK), an IDE (code editor), and we will use Maven (which comes built into our IDE). Follow each step carefully.

1 Install Java JDK 17 (or newer)

The JDK contains the Java compiler (javac) and the Java Runtime (JRE). We will use JDK 17, which is a Long Term Support version — meaning it is stable and supported for many years.

Download steps:

1. Open your browser and go to **adoptium.net** (this is a free, trusted source for OpenJDK)
2. Click the big download button. It will detect your operating system automatically.
3. Install the downloaded file. On Windows, accept all defaults and tick "Set JAVA_HOME variable".
4. Open Command Prompt (Windows) or Terminal (Mac/Linux).
5. Type the verification command shown below.

Verify the installation by typing this command:

```
java -version
```

PREVIEW · Expected Output

If installed correctly, you will see something like:

```
openjdk version "17.0.9" 2023-10-17
OpenJDK Runtime Environment Temurin-17.0.9+9
OpenJDK 64-Bit Server VM Temurin-17.0.9+9
```

If you see "command not found" instead, your PATH variable was not set. Restart your computer and try again.

2 Install IntelliJ IDEA Community Edition

IntelliJ IDEA is the most popular IDE for Java development. The Community Edition is completely free and includes everything we need for Spring Boot. It is much more beginner-friendly than older alternatives like Eclipse.

Download steps:

1. Go to **jetbrains.com/idea/download**
2. Scroll down to the section labeled "Community Edition" — make sure NOT to download the paid Ultimate version.
3. Click Download. Run the installer with default settings.
4. Launch IntelliJ IDEA. You will see a welcome screen — keep it open for the next step.

Why IntelliJ IDEA?
It auto-completes code, highlights errors as you type, runs Spring Boot apps with one click, and includes Maven (so you do not need to install Maven separately). It will save you many hours of confusion as a beginner.

3 What is Maven (and Why You Already Have It)

Maven is a build tool for Java. Think of it as a package manager and task runner combined — like Composer for PHP or pip for Python.

Maven reads a file called **pom.xml** in your project. This file lists all the libraries your project needs. When you build the project, Maven downloads every library automatically. You do not have to hunt down JAR files yourself.

Good News
IntelliJ IDEA Community Edition comes with Maven built in. You do not need to install it separately. When we create our Spring Boot project, Maven will automatically download all required libraries on the first run.

Part 3: Creating Your First Spring Boot Project

We will use **Spring Initializr**, the official tool from the Spring team to generate a starter project. It is a website that asks a few questions and gives you a ready-to-use ZIP file. This is the standard way every Spring developer starts a new project.

1 Generate the Project on start.spring.io

Open your browser and go to **start.spring.io**. You will see a form with many fields. Fill them in exactly like this:

Field	Value	Why
Project	Maven	Standard Java build tool
Language	Java	Our chosen language
Spring Boot	3.2.x (latest stable)	Most recent stable release
Group	com.egotechworld	Reverse domain naming convention
Artifact	studentapp	Project name
Name	studentapp	Same as artifact
Description	Student CRUD App	Short description
Package name	com.egotechworld.studentapp	Auto-filled
Packaging	Jar	Standard for Spring Boot
Java	17	LTS version we installed

2 Add the Required Dependencies

On the right side of Spring Initializr, click **ADD DEPENDENCIES**. A search box opens. Search for and add each of these one by one:

Dependency	What it Does
Spring Web	Lets us build web pages and REST APIs. Includes Tomcat server.
Spring Data JPA	Talks to the database without writing SQL by hand.
H2 Database	A simple in-memory database for testing. No installation needed.
Thymeleaf	Template engine for HTML. Lets Java fill in dynamic data into HTML pages.
Spring Boot DevTools	Auto-restarts the app when you change code. Saves time.

3 Generate, Download, and Open in IntelliJ

1. Click the green **GENERATE** button at the bottom of Spring Initializr.
2. A file named **studentapp.zip** downloads. Extract it to a folder you can find easily, like **C:\projects\studentapp**
3. Open IntelliJ IDEA. Click **Open** and select the extracted folder.
4. IntelliJ will detect it as a Maven project and start downloading dependencies. **Be patient — this takes 2 to 5 minutes the first time.**
5. When the bottom progress bar disappears, the project is ready.

PREVIEW · What You Will See in IntelliJ

On the left side, a folder tree appears showing your project structure. The main areas are **src/main/java** for your Java code and **src/main/resources** for HTML templates and config files. On the bottom, a small Maven progress indicator shows when libraries finish downloading.

Part 4: Understanding the Folder Structure

Before we start coding, let's understand how a Spring Boot project is organized. Knowing where each file goes will save you a lot of confusion.

The Default Structure

```
studentapp/
 |
 +--- src/
 |     |
 |     +--- main/
 |     |     |
 |     |     +--- java/
 |     |     |     |
 |     |     |     +--- com/egotechworld/studentapp/
 |     |     |           |
 |     |     |           +--- StudentappApplication.java  (main class)
 |     |     |
 |     |     +--- resources/
 |     |           |
 |     |           +--- application.properties  (config file)
 |     |           |
 |     |           +--- static/  (CSS, JS, images)
 |     |           |
 |     |           +--- templates/  (HTML files)
 |
 +--- pom.xml  (Maven dependencies file)
```

What Goes Where

`src/main/java/...`

All your Java code lives here. We will create new folders inside our package: **entity**, **repository**, **service**, and **controller**.

`StudentappApplication.java`

The main class. This is where your app starts. Spring Boot generates this for you — you usually do not change it.

`application.properties`

Configuration goes here: database URL, port number, etc.

`src/main/resources/templates/`

All your HTML files go here. Thymeleaf will read them and fill in dynamic data.

`src/main/resources/static/`

Static assets like CSS, JavaScript, and images. We will use this very lightly since Bootstrap loads from a CDN.

`pom.xml`

The Maven file that lists all your dependencies. Spring Initializr already filled this in based on the choices you made.

The Folder Structure We Will Build

By the end of this tutorial, your **com/egotechworld/studentapp** folder will look like this. We will create each file step by step.

```
com/egotechworld/studentapp/
 |
 +--- StudentappApplication.java  (already exists)
 |
 +--- entity/
 |      |
 |      +--- Student.java  (database table model)
 |
 +--- repository/
 |      |
 |      +--- StudentRepository.java  (database operations)
 |
 +--- service/
 |      |
 |      +--- StudentService.java  (business logic)
 |
 +--- controller/
        |
        +--- StudentController.java  (handles web requests)
```

The Layered Architecture

This pattern (Entity → Repository → Service → Controller) is called a **layered architecture**. Each layer has one job. The Controller talks to the Service, the Service talks to the Repository, and the Repository talks to the database. This separation makes apps easier to test and maintain.

Templates Folder Structure

```
src/main/resources/templates/
 |
 +--- index.html  (homepage - student list)
 |
 +--- add-student.html  (form to add a student)
 |
 +--- edit-student.html  (form to edit a student)
```

Part 5: Configuring the Application

Open the file **src/main/resources/application.properties**. It is empty by default. We will add settings to enable our H2 database and configure how our app runs.

Replace the contents with this:

application.properties

```
# ==========================================<br/># EgoTechWorld Student App
Configuration<br/># ==========================================<br/><br/># Application
name<br/>spring.application.name=studentapp<br/><br/># Server port (default is
8080)<br/>server.port=8080<br/><br/># H2 Database settings<br/>spring.datasource.url=jdb
c:h2:mem:studentdb<br/>spring.datasource.driver-class-name=org.h2.Driver<br/>spring.data
source.username=sa<br/>spring.datasource.password=<br/><br/># Enable H2 console (visit
/h2-console in browser)<br/>spring.h2.console.enabled=true<br/>spring.h2.console.path=/h
2-console<br/><br/># JPA settings - auto create database tables from entities<br/>spring
.jpa.hibernate.ddl-auto=update<br/>spring.jpa.show-sql=true<br/>spring.jpa.properties.hi
bernate.dialect=org.hibernate.dialect.H2Dialect<br/><br/># Thymeleaf settings - disable
cache during development<br/>spring.thymeleaf.cache=false
```

What Each Line Does

`server.port=8080`

Your app will be available at http://localhost:8080. Change this if 8080 is busy.

`spring.datasource.url=jdbc:h2:mem:studentdb`

Use H2 in memory mode. Data lives in RAM and resets when the app stops. Perfect for learning. Later you can swap this for MySQL.

`spring.h2.console.enabled=true`

Enables a web-based database viewer at /h2-console where you can run SQL queries to see your data.

`spring.jpa.hibernate.ddl-auto=update`

JPA automatically creates database tables from your Java entity classes. No SQL needed.

`spring.jpa.show-sql=true`

Print every SQL query to the console. Great for learning what is happening.

`spring.thymeleaf.cache=false`

Tell Thymeleaf not to cache HTML files. When you change a template, you see the change immediately.

Part 6: Building the Entity (Database Model)

An **Entity** is a Java class that maps to a database table. Each field in the class becomes a column in the table. JPA reads our entity and creates the table for us — no SQL needed.

1 Create the entity Folder

In IntelliJ, right-click on the **com.egotechworld.studentapp** package → **New** → **Package**. Name it **entity**. The full package becomes:

```
com.egotechworld.studentapp.entity
```

2 Create Student.java

Right-click on the new **entity** package → **New** → **Java Class**. Name it **Student**. Then paste this code:

entity/Student.java

```
package com.egotechworld.studentapp.entity;<br/><br/>import
jakarta.persistence.Entity;<br/>import jakarta.persistence.GeneratedValue;<br/>import
jakarta.persistence.GenerationType;<br/>import jakarta.persistence.Id;<br/>import
jakarta.persistence.Table;<br/><br/>@Entity<br/>@Table(name =
&amp;quot;students&amp;quot;)<br/>public class Student {<br/><br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;@Id<br
/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;@GeneratedValue(strategy =
GenerationType.IDENTITY)<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;private Long
id;<br/><br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;private String
name;<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;private String
email;<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;private String
course;<br/><br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;// Default constructor (required by
JPA)<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public Student() {}<br/><br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;//
Getters and Setters<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public Long getId() { return id;
}<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public void setId(Long id) { this.id = id;
}<br/><br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public String getName() { return name;
}<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public void setName(String name) { this.name = name;
}<br/><br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public String getEmail() { return email;
}<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public void setEmail(String email) { this.email = email;
}<br/><br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public String getCourse() { return course;
}<br/>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;public void setCourse(String course) { this.course =
course; }<br/>}
```

Understanding the Annotations

Words starting with @ are called **annotations**. They give instructions to Spring and JPA. Here is what each one means:

`@Entity`

Tells JPA: "This class represents a database table." Without this, JPA ignores the class.

`@Table(name = "students")`

Sets the table name to **students**. If you skip this, JPA uses the class name (Student) as the table name.

`@Id`

Marks this field as the primary key — the unique identifier for each row.

`@GeneratedValue(strategy = GenerationType.IDENTITY)`

Tells the database to auto-generate the ID. Each new student gets a fresh number (1, 2, 3...) automatically.

PREVIEW · What This Creates Behind the Scenes

When the app starts, JPA reads this class and runs SQL similar to this for you:

```
CREATE TABLE students (
   id BIGINT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255),
   email VARCHAR(255),
   course VARCHAR(255)
);
```

You did not write SQL. JPA wrote it for you. This is the magic of Object Relational Mapping.

About Getters and Setters

Java does not let you read a private field directly from outside the class. You need **get** and **set** methods. They look repetitive but they are needed. IntelliJ can generate them automatically: right-click in the file → **Generate** → **Getters and Setters**.

Part 7: Building the Repository

The **Repository** is the layer that talks to the database. The amazing thing about Spring Data JPA: you do not write any SQL or even any code — you just write an interface, and Spring creates the implementation automatically.

1 Create the repository Folder

Create a new package called **repository** next to **entity**. The full path:

```
com.egotechworld.studentapp.repository
```

2 Create StudentRepository.java

Inside the **repository** package, create a new file. **This time choose Interface, not Class.** Name it **StudentRepository**:

repository/StudentRepository.java

```
package com.egotechworld.studentapp.repository;<br/><br/>import
com.egotechworld.studentapp.entity.Student;<br/>import
org.springframework.data.jpa.repository.JpaRepository;<br/>import
org.springframework.stereotype.Repository;<br/><br/>@Repository<br/>public interface
StudentRepository extends JpaRepository&lt;Student, Long&gt;
{<br/><br/>    // That is it! All CRUD methods are
inherited.<br/><br/>}
```

That Is Really All There Is To It

By extending **JpaRepository<Student, Long>** we get all of these methods for free, with zero implementation:

Method	What it Does
`save(Student s)`	Insert a new student or update an existing one
`findById(Long id)`	Get a single student by their ID
`findAll()`	Get a list of every student in the database
`deleteById(Long id)`	Delete a student by their ID

`count()`	Count how many students exist
`existsById(Long id)`	Check if a student with that ID exists

Part 8: Building the Service Layer

The **Service** layer holds the business logic. Why have it if the Repository already does CRUD? Two reasons:

1. Business Rules. If you ever need to validate emails, send notifications, or check permissions before saving — that goes here, not in the controller.

2. Reusability. Multiple controllers can use the same service. Multiple services can use the same repository.

1 Create the service Folder

```
com.egotechworld.studentapp.service
```

2 Create StudentService.java

Inside the **service** package, create a new **Class** (not interface) called **StudentService**:

service/StudentService.java

```
package com.egotechworld.studentapp.service;<br/><br/>import
com.egotechworld.studentapp.entity.Student;<br/>import
com.egotechworld.studentapp.repository.StudentRepository;<br/>import
org.springframework.beans.factory.annotation.Autowired;<br/>import
org.springframework.stereotype.Service;<br/><br/>import java.util.List;<br/>import
java.util.Optional;<br/><br/>@Service<br/>public class StudentService
{<br/><br/>    @Autowired<br/>    private
StudentRepository studentRepository;<br/><br/>    // CREATE /
UPDATE<br/>    public Student saveStudent(Student student)
{<br/>        return studentRepository.save(stud
ent);<br/>    }<br/><br/>    // READ
ALL<br/>    public List&lt;Student&gt; getAllStudents()
{<br/>        return studentRepository.findAll()
;<br/>    }<br/><br/>    // READ
ONE<br/>    public Student getStudentById(Long id)
{<br/>        Optional&lt;Student&gt; optional =
studentRepository.findById(id);<br/>        retu
rn
optional.orElse(null);<br/>    }<br/><br/>    //
DELETE<br/>    public void deleteStudent(Long id) {<br/>  
       studentRepository.deleteById(id);<br/>  &n
bsp; }<br/>}
```

Annotation Notes

`@Service`

Tells Spring this class is a service component. Spring will create one instance of it automatically and reuse it everywhere.

`@Autowired`

Tells Spring: "Find an instance of StudentRepository and inject it into this field." This is called **Dependency Injection** — instead of you creating objects, Spring creates them and hands them to your code.

> **Why Optional?**
> **Optional<Student>** is Java's way of saying "this might be null." If a student with the given ID does not exist, **findById** returns an empty Optional instead of crashing. We use **.orElse(null)** to convert it back to a regular Student or null. This prevents NullPointerExceptions, the most common Java error.

Part 9: Building the Controller

The **Controller** is the front-line worker. It receives HTTP requests from the browser, asks the Service for data, and sends back HTML pages. Every URL in your app maps to a method in a controller.

1 Create the controller Folder

```
com.egotechworld.studentapp.controller
```

2 Create StudentController.java

Inside the **controller** package, create a new **Class** called **StudentController**. This file is bigger than the others, but each method does just one thing.

controller/StudentController.java

```
package com.egotechworld.studentapp.controller;<br/><br/>import
com.egotechworld.studentapp.entity.Student;<br/>import
com.egotechworld.studentapp.service.StudentService;<br/>import
org.springframework.beans.factory.annotation.Autowired;<br/>import
org.springframework.stereotype.Controller;<br/>import
org.springframework.ui.Model;<br/>import
org.springframework.web.bind.annotation.*;<br/><br/>@Controller<br/>public class
StudentController
{<br/><br/>    @Autowired<br/>    private
StudentService studentService;<br/><br/>    // HOME PAGE - shows all
students<br/>    @GetMapping(&quot;/&quot;)<br/>   &n
bsp;public String home(Model model) {<br/>       &nbs
p;model.addAttribute(&quot;students&quot;, studentService.getAllStudents());<br/> &
nbsp;     return
&quot;index&quot;;<br/>    }<br/><br/>    //
SHOW ADD FORM<br/>    @GetMapping(&quot;/add&quot;)<br/>  
  public String showAddForm(Model model) {<br/>     &
nbsp;  model.addAttribute(&quot;student&quot;, new
Student());<br/>        return &quot;add-student
&quot;;<br/>    }<br/><br/>    // HANDLE ADD
FORM SUBMIT<br/>    @PostMapping(&quot;/save&quot;)<br/>  
  public String saveStudent(@ModelAttribute Student student) {<br/>  
;      studentService.saveStudent(student);<br/> &nbs
p;      return &quot;redirect:/&quot;;<br/>  &nb
sp; }<br/><br/>    // SHOW EDIT FORM<br/>   &nbs
p;@GetMapping(&quot;/edit/{id}&quot;)<br/>    public String
showEditForm(@PathVariable Long id, Model model)
{<br/>        Student student = studentService.g
etStudentById(id);<br/>        model.addAttribut
e(&quot;student&quot;,
student);<br/>        return &quot;edit-student&
quot;;<br/>    }<br/><br/>    // HANDLE EDIT
FORM SUBMIT<br/>    @PostMapping(&quot;/update/{id}&quot;)<br/> 
;   public String updateStudent(@PathVariable Long id,<br/>  &n
bsp;              
            @ModelAttribute
Student student) {<br/>        student.setId(id)
;<br/>        studentService.saveStudent(student
);<br/>        return &quot;redirect:/&quot;;<br
/>    }<br/><br/>    // DELETE STUDENT<br/> 
;   @GetMapping(&quot;/delete/{id}&quot;)<br/>    pub
lic String deleteStudent(@PathVariable Long id) {<br/>     &nbs
p;  studentService.deleteStudent(id);<br/>      
  return &quot;redirect:/&quot;;<br/>    }<br/>}
```

URL to Method Mapping

Here is how each URL in your app connects to a method in the controller:

URL	Method	What Happens
`GET /`	`home()`	Show list of all students
`GET /add`	`showAddForm()`	Show empty form to add new student
`POST /save`	`saveStudent()`	Process form, save to database
`GET /edit/5`	`showEditForm()`	Show form pre-filled with student 5
`POST /update/5`	`updateStudent()`	Save changes to student 5
`GET /delete/5`	`deleteStudent()`	Remove student 5 from database

Controller Annotations Explained

`@Controller`

Marks this class as a controller that returns HTML pages. (For JSON APIs, you would use **@RestController** instead.)

`@GetMapping("/path")`

Maps GET requests (when someone visits a URL) to this method.

`@PostMapping("/path")`

Maps POST requests (form submits) to this method.

`@PathVariable Long id`

Reads the value from the URL path. For **/edit/5**, id becomes 5.

`@ModelAttribute Student student`

Takes form fields and packs them into a Student object automatically.

`Model model`

A box for sending data to the HTML page. Use **model.addAttribute(...)** to add data.

`return "index"`

Tells Spring to render the file **templates/index.html**.

`return "redirect:/"`

Tells the browser to go to the homepage. Used after form submits to prevent double-submission.

Part 10: Building the HTML Templates

Now we build the user interface. We will use **Bootstrap 5** from a CDN — no installation needed. Just include one CSS link and one JS link, and we get a beautiful responsive design.

Create three files inside **src/main/resources/templates/**:

- `index.html` — The homepage that shows the student list table
- `add-student.html` — The form to create a new student
- `edit-student.html` — The form to edit an existing student

Template 1: index.html (Homepage)

templates/index.html

```
&lt;!DOCTYPE html&gt;<br/>&lt;html xmlns:th=&quot;http://www.thymeleaf.org&quot;&gt;<br/
>&lt;head&gt;<br/>    &lt;meta
charset=&quot;UTF-8&quot;&gt;<br/>    &lt;title&gt;Students -
EgoTechWorld&lt;/title&gt;<br/>    &lt;link href=&quot;https://cdn.j
sdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css&quot;<br/>   &
nbsp;      rel=&quot;stylesheet&quot;&gt;<br/>&lt;/head&gt
;<br/>&lt;body class=&quot;bg-light&quot;&gt;<br/><br/>&lt;nav class=&quot;navbar
navbar-dark bg-primary&quot;&gt;<br/>  &lt;div
class=&quot;container&quot;&gt;<br/>    &lt;span
class=&quot;navbar-brand mb-0 h1&quot;&gt;EgoTechWorld Student
App&lt;/span&gt;<br/>  &lt;/div&gt;<br/>&lt;/nav&gt;<br/><br/>&lt;div
class=&quot;container mt-4&quot;&gt;<br/>  &lt;div class=&quot;d-flex
justify-content-between mb-3&quot;&gt;<br/>    &lt;h2&gt;All
Students&lt;/h2&gt;<br/>    &lt;a href=&quot;/add&quot;
class=&quot;btn btn-success&quot;&gt;+ Add
Student&lt;/a&gt;<br/>  &lt;/div&gt;<br/><br/>  &lt;div
class=&quot;card shadow-sm&quot;&gt;<br/>    &lt;table
class=&quot;table table-hover
mb-0&quot;&gt;<br/>      &lt;thead class=&quot;table-light
&quot;&gt;<br/>        &lt;tr&gt;<br/> &nbs
p;       &lt;th&gt;ID&lt;/th&gt;&lt;th&gt;Name&
lt;/th&gt;&lt;th&gt;Email&lt;/th&gt;<br/>        
;  &lt;th&gt;Course&lt;/th&gt;&lt;th&gt;Actions&lt;/th&gt;<br/>  &nb
sp;     &lt;/tr&gt;<br/>      &lt
;/thead&gt;<br/>      &lt;tbody&gt;<br/>   
     &lt;tr th:each=&quot;s : ${students}&quot;&gt;<br/> &
nbsp;       &lt;td th:text=&quot;${s.id}&quot;&
gt;&lt;/td&gt;<br/>         &lt;td th
:text=&quot;${s.name}&quot;&gt;&lt;/td&gt;<br/>       
;   &lt;td th:text=&quot;${s.email}&quot;&gt;&lt;/td&gt;<br/>  
       &lt;td th:text=&quot;${s.course}&quot;&g
t;&lt;/td&gt;<br/>         &lt;td&gt;
<br/>           &lt;a th:hr
ef=&quot;@{/edit/{id}(id=${s.id})}&quot;<br/>       &
nbsp;       class=&quot;btn btn-sm btn-warning&quot;&
gt;Edit&lt;/a&gt;<br/>           
 &lt;a th:href=&quot;@{/delete/{id}(id=${s.id})}&quot;<br/>    
            class=&quot;btn btn-sm
btn-danger&quot;<br/>           &
nbsp;   onclick=&quot;return confirm(&apos;Delete this student?&apos;)&qu
ot;&gt;Delete&lt;/a&gt;<br/>          
&lt;/td&gt;<br/>        &lt;/tr&gt;<br/> &n
bsp;    &lt;/tbody&gt;<br/>    &lt;/table&gt;<br
/>  &lt;/div&gt;<br/>&lt;/div&gt;<br/><br/>&lt;/body&gt;<br/>&lt;/html&gt;
```

PREVIEW · What index.html Looks Like in the Browser

A blue navbar at the top with the title **EgoTechWorld Student App**. Below that, a heading **All Students** on the left and a green **+ Add Student** button on the right. Then a clean white table inside a subtle shadowed card showing five columns: ID, Name, Email, Course, and Actions. Each row has a yellow **Edit** button and a red **Delete** button (which asks for confirmation before deleting).

Thymeleaf Special Attributes

Attribute	Purpose
`th:each="s : ${students}"`	Loop through each student in the list
`th:text="${s.name}"`	Set the text content from the variable
`th:href="@{/edit/{id}(id=${s.id})}"`	Build a URL with dynamic parts
`th:value="${s.name}"`	Pre-fill an input field
`th:action="@{/save}"`	Set the form action URL

Template 2: add-student.html

templates/add-student.html

```
&lt;!DOCTYPE html&gt;<br/>&lt;html xmlns:th=&quot;http://www.thymeleaf.org&quot;&gt;<br/
>&lt;head&gt;<br/>    &lt;meta
charset=&quot;UTF-8&quot;&gt;<br/>    &lt;title&gt;Add Student -
EgoTechWorld&lt;/title&gt;<br/>    &lt;link href=&quot;https://cdn.j
sdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css&quot;<br/>   &
nbsp;      rel=&quot;stylesheet&quot;&gt;<br/>&lt;/head&gt
;<br/>&lt;body class=&quot;bg-light&quot;&gt;<br/><br/>&lt;nav class=&quot;navbar
navbar-dark bg-primary&quot;&gt;<br/>  &lt;div
class=&quot;container&quot;&gt;<br/>    &lt;span
class=&quot;navbar-brand mb-0 h1&quot;&gt;EgoTechWorld Student
App&lt;/span&gt;<br/>  &lt;/div&gt;<br/>&lt;/nav&gt;<br/><br/>&lt;div
class=&quot;container mt-4&quot;&gt;<br/>  &lt;div class=&quot;card
shadow-sm&quot;&gt;<br/>    &lt;div class=&quot;card-header
bg-success text-white&quot;&gt;<br/>      &lt;h4
class=&quot;mb-0&quot;&gt;Add New Student&lt;/h4&gt;<br/>    &lt;/di
v&gt;<br/>    &lt;div
class=&quot;card-body&quot;&gt;<br/>      &lt;form
th:action=&quot;@{/save}&quot; th:object=&quot;${student}&quot; method=&quot;post&quot;&
gt;<br/><br/>        &lt;div class=&quot;mb-3&qu
ot;&gt;<br/>          &lt;label class=
&quot;form-label&quot;&gt;Name&lt;/label&gt;<br/>      &nb
sp;   &lt;input type=&quot;text&quot; th:field=&quot;*{name}&quot;<br/>&n
bsp;              
  class=&quot;form-control&quot; required&gt;<br/>    &nbs
p;   &lt;/div&gt;<br/><br/>       &nbs
p;&lt;div class=&quot;mb-3&quot;&gt;<br/>        
;  &lt;label class=&quot;form-label&quot;&gt;Email&lt;/label&gt;<br/> &nb
sp;        &lt;input type=&quot;email&quot; th:f
ield=&quot;*{email}&quot;<br/>         &nbs
p;       class=&quot;form-control&quot; required&gt;<
br/>        &lt;/div&gt;<br/><br/>  &n
bsp;     &lt;div class=&quot;mb-3&quot;&gt;<br/>  &nb
sp;       &lt;label class=&quot;form-label&quot;&gt;C
ourse&lt;/label&gt;<br/>          &lt;
input type=&quot;text&quot; th:field=&quot;*{course}&quot;<br/>    &
nbsp;            class=&quot
;form-control&quot; required&gt;<br/>        &lt
;/div&gt;<br/><br/>        &lt;button
type=&quot;submit&quot; class=&quot;btn btn-success&quot;&gt;Save&lt;/button&gt;<br/>&nb
sp;       &lt;a href=&quot;/&quot; class=&quot;btn bt
n-secondary&quot;&gt;Cancel&lt;/a&gt;<br/>      &lt;/form&
gt;<br/>    &lt;/div&gt;<br/>  &lt;/div&gt;<br/>&lt;/div&g
t;<br/><br/>&lt;/body&gt;<br/>&lt;/html&gt;
```

PREVIEW · What add-student.html Looks Like

The same blue navbar at the top. Below that, a card with a green header saying **Add New Student**. Inside the card, three input fields labeled **Name**, **Email**, and **Course** — all clean Bootstrap inputs. At the bottom, a green **Save** button next to a gray **Cancel** button. Email field validates the email format automatically thanks to `type="email"`.

Template 3: edit-student.html

This is almost identical to the add form, but with two key differences: the form action goes to **/update/{id}**, and the fields are pre-filled because Thymeleaf reads the student object passed from the controller.

templates/edit-student.html

```
&lt;!DOCTYPE html&gt;<br/>&lt;html xmlns:th=&quot;http://www.thymeleaf.org&quot;&gt;<br/
>&lt;head&gt;<br/>    &lt;meta
charset=&quot;UTF-8&quot;&gt;<br/>    &lt;title&gt;Edit Student -
EgoTechWorld&lt;/title&gt;<br/>    &lt;link href=&quot;https://cdn.j
sdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css&quot;<br/>   &
nbsp;      rel=&quot;stylesheet&quot;&gt;<br/>&lt;/head&gt
;<br/>&lt;body class=&quot;bg-light&quot;&gt;<br/><br/>&lt;nav class=&quot;navbar
navbar-dark bg-primary&quot;&gt;<br/>  &lt;div
class=&quot;container&quot;&gt;<br/>    &lt;span
class=&quot;navbar-brand mb-0 h1&quot;&gt;EgoTechWorld Student
App&lt;/span&gt;<br/>  &lt;/div&gt;<br/>&lt;/nav&gt;<br/><br/>&lt;div
class=&quot;container mt-4&quot;&gt;<br/>  &lt;div class=&quot;card
shadow-sm&quot;&gt;<br/>    &lt;div class=&quot;card-header
bg-warning&quot;&gt;<br/>      &lt;h4
class=&quot;mb-0&quot;&gt;Edit Student&lt;/h4&gt;<br/>    &lt;/div&g
t;<br/>    &lt;div
class=&quot;card-body&quot;&gt;<br/>      &lt;form th:acti
on=&quot;@{/update/{id}(id=${student.id})}&quot;<br/>      
;       th:object=&quot;${student}&quot; method
=&quot;post&quot;&gt;<br/><br/>        &lt;div c
lass=&quot;mb-3&quot;&gt;<br/>         &nbs
p;&lt;label class=&quot;form-label&quot;&gt;Name&lt;/label&gt;<br/>   &nb
sp;      &lt;input type=&quot;text&quot; th:field=&quot;*{
name}&quot;<br/>            
     class=&quot;form-control&quot; required&gt;<br/> &nbs
p;      &lt;/div&gt;<br/><br/>    &nbs
p;   &lt;div class=&quot;mb-3&quot;&gt;<br/>     
;    &lt;label class=&quot;form-label&quot;&gt;Email&lt;/label
&gt;<br/>         &lt;input
type=&quot;email&quot; th:field=&quot;*{email}&quot;<br/>     &
nbsp;           class=&quot;form-
control&quot; required&gt;<br/>        &lt;/div&
gt;<br/><br/>        &lt;div class=&quot;mb-3&qu
ot;&gt;<br/>          &lt;label class=
&quot;form-label&quot;&gt;Course&lt;/label&gt;<br/>      &
nbsp;   &lt;input type=&quot;text&quot; th:field=&quot;*{course}&quot;<br
/>              &n
bsp;  class=&quot;form-control&quot; required&gt;<br/>    
    &lt;/div&gt;<br/><br/>       
 &lt;button type=&quot;submit&quot; class=&quot;btn btn-warning&quot;&gt;Update&lt;
/button&gt;<br/>        &lt;a href=&quot;/&quot;
class=&quot;btn btn-secondary&quot;&gt;Cancel&lt;/a&gt;<br/>    &nbs
p; &lt;/form&gt;<br/>    &lt;/div&gt;<br/>  &lt;/div&
gt;<br/>&lt;/div&gt;<br/><br/>&lt;/body&gt;<br/>&lt;/html&gt;
```

PREVIEW · What edit-student.html Looks Like

Same layout as the add form, but with a yellow header reading **Edit Student**. All three input fields come pre-filled with the existing student's information thanks to Thymeleaf's `th:field`. The submit button is yellow and says **Update**. Clicking it sends the form to `/update/{id}` where the controller saves the changes.

Part 11: Running and Testing the App

All the code is written. Time to see it in action!

1 Run the Application

1. In IntelliJ, open **StudentappApplication.java** in the main package.
2. Look at the line **public static void main**. To the left, you will see a small green Play arrow icon.
3. Click the green Play icon → **Run 'StudentappApplication'**.
4. The bottom of IntelliJ opens a console showing Spring Boot's startup messages.
5. Wait until you see: **Started StudentappApplication in X seconds**.

2 Open Your App in the Browser

Open your browser and visit:

```
http://localhost:8080
```

PREVIEW · First Visit

You will see your homepage: blue navbar at top, the title **All Students**, and a green **+ Add Student** button. The table will be empty since you have not added anyone yet. Click **+ Add Student** to begin testing.

3 Test the Full CRUD Flow

Walk through each operation to confirm everything works:

CREATE: Click **+ Add Student**. Fill in Name, Email, Course. Click **Save**. You return to the homepage and see your new student in the table.

READ: The homepage already shows all students. Add 2 or 3 more to see the list grow.

UPDATE: Click the yellow **Edit** button next to any student. Fields are pre-filled. Change something and click **Update**. The change appears in the table.

DELETE: Click the red **Delete** button. Confirm the popup. The student vanishes from the list.

4 Inspect the Database (Optional)

Want to see the actual database? Visit the H2 console in your browser:

```
http://localhost:8080/h2-console
```

On the login page, set:

Field	Value
Driver Class	org.h2.Driver
JDBC URL	jdbc:h2:mem:studentdb
User Name	sa
Password	(leave empty)

Click **Connect**. You will see the **STUDENTS** table on the left. Double-click it, then click **Run** to see all rows.

Part 12: Common Errors and How to Fix Them

Almost every beginner hits these problems. Here is what they look like and how to fix them quickly.

Error: Port 8080 was already in use

Something else is using port 8080 (maybe a previous run that did not stop). Fix: change the port in **application.properties** to **server.port=8081** and restart. Or close the other app using port 8080.

Error: Could not autowire. No beans of type 'StudentRepository' found

Spring cannot find your Repository class. Make sure: (1) the class has **@Repository** on it, (2) it is in a sub-package of **com.egotechworld.studentapp**, NOT outside it.

Error: Whitelabel Error Page

The URL has no controller mapping, or your controller method has a bug. Check that the URL matches exactly what is in **@GetMapping** or **@PostMapping**. Check spelling carefully — Java is case sensitive.

Error: Template might not exist or might not be accessible

Thymeleaf cannot find the HTML file. Check: (1) the file is in **src/main/resources/templates/**, (2) the filename in your controller matches exactly without the .html extension. **return "index"** looks for **index.html**.

My changes are not showing up

DevTools should auto-restart, but sometimes it does not. Stop the app (red square in IntelliJ) and click Run again. For HTML changes, just refresh the browser — Thymeleaf cache is disabled in our config.

NullPointerException when editing

The student ID does not exist in the database. This usually happens if you bookmark an edit URL and the data was wiped (H2 in-memory mode loses data on restart). Just go to / and pick an existing student.

Part 13: What to Build Next

Congratulations — you have built a complete CRUD web app with Java Spring Boot! Here are some directions to grow from here:

Easy Next Steps

- Add a search box to filter students by name (use a custom repository method).
- Add validation messages — show "Email is required" when fields are empty.
- Switch from H2 to MySQL so data survives app restarts.
- Add timestamps to track when each student was created.
- Make the table sortable by column.
- Add pagination if you have many records.

Bigger Next Steps

- Add user login with Spring Security so only logged-in users can manage students.
- Build a REST API version (use **@RestController**) and connect a React frontend.
- Add file upload to attach student photos.
- Deploy your app to a free cloud service like Render, Railway, or Heroku.
- Add unit tests with JUnit and Mockito.
- Use Docker to package your app.

> **Keep Learning at EgoTechWorld.com**
> Visit **egotechworld.com** for more beginner-friendly tutorials on PHP, Python, Django, React, Git, and more. New tutorials are added regularly. Every concept on the site is explained in plain English with full working code examples — just like this one.

Final Folder Structure Reference

```
studentapp/
 |
 +--- src/main/java/com/egotechworld/studentapp/
 |     |
 |     +--- StudentappApplication.java
 |     +--- entity/Student.java
 |     +--- repository/StudentRepository.java
 |     +--- service/StudentService.java
 |     +--- controller/StudentController.java
 |
 +--- src/main/resources/
 |     |
 |     +--- application.properties
 |     +--- templates/index.html
 |     +--- templates/add-student.html
 |     +--- templates/edit-student.html
 |
 +--- pom.xml
```

Thank You!

You have completed the
Spring Boot CRUD Tutorial

What You Now Know

- How Java and Spring Boot work together
- Layered architecture (Entity, Repository, Service, Controller)
- JPA and database mapping with annotations
- Thymeleaf for dynamic HTML templates
- Bootstrap 5 for clean responsive UI
- Full CRUD operations end-to-end
- How to debug common Spring Boot errors

EgoTechWorld.com

Learn. Build. Grow.
Visit us for more beginner-friendly tutorials