

Student CRUD Application with Spring Boot + Hibernate (Step-by-step for Eclipse)

This guide walks you through creating a Student CRUD web application using Spring Boot (Spring MVC + Spring Data JPA), Hibernate as the JPA provider, Maven, and Thymeleaf. It is tailored for Eclipse IDE (Eclipse IDE for Enterprise Java Developers). The steps include all files and commands you need.

Prerequisites

1. Eclipse IDE for Enterprise Java Developers (2021-12 or later recommended).
 2. JDK 11 or later installed and configured in Eclipse (we'll use JDK 17 examples; adapt if you use 21).
 3. Maven (Eclipse includes embedded Maven, but having mvn CLI helps).
 4. MySQL (or any RDBMS). Example uses MySQL — create database `student_db`.
 5. Internet access to download Maven dependencies.
-

Project Overview

- Spring Boot (spring-boot-starter-web)
 - Spring Data JPA (uses Hibernate)
 - Thymeleaf for views
 - MySQL Connector/J
-

1) Create the Project in Eclipse

1. File → New → Maven Project → Next.
2. Check “Create a simple project (skip archetype selection)” → Next.
3. Fill:
 - Group Id: `com.example`
 - Artifact Id: `student-crud`
 - Packaging: `jar`
 - Name: `student-crud`
4. Finish.

Eclipse will create a Maven project. Replace the generated `pom.xml` with the content below.

2) pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>student-crud</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.4</version>
    <relativePath/> <!-- Lookup parent from repository -->
  </parent>

  <properties>
    <java.version>17</java.version>
  </properties>

  <dependencies>
    <!-- Web -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Thymeleaf -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- Spring Data JPA (uses Hibernate) -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- MySQL connector -->
    <dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <scope>runtime</scope>
    </dependency>

    <!-- Devtools (optional, for hot reload during development) -->
```

```

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <optional>true</optional>
    </dependency>

    <!-- Test -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>

```

If your project needs to run on Tomcat separately (WAR), we can adapt later; for now Spring Boot's embedded Tomcat is easiest.

3) Application Properties

Create `src/main/resources/application.properties` (or `application.yml`). Example `application.properties`:

```

spring.datasource.url=jdbc:mysql://localhost:3306/student_db?useSSL=false&all
owPublicKeyRetrieval=true&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=your_mysql_password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.thymeleaf.cache=false
server.port=8080

```

- `ddl-auto=update` is convenient for development (creates/updates tables). For production, use `validate` or manage migrations with Flyway/Liquibase.
-

4) Package Structure

Create packages under `src/main/java` like:

```
com.example.studentcrud
├── StudentCrudApplication.java
├── controller
├── model
├── repository
├── service
└── dto      (optional)
```

5) Main Application Class

`src/main/java/com/example/studentcrud/StudentCrudApplication.java`

```
package com.example.studentcrud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

6) Entity: Student

`src/main/java/com/example/studentcrud/model/Student.java`

```
package com.example.studentcrud.model;

import jakarta.persistence.*;

@Entity
@Table(name = "students")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String firstName;
```

```

private String lastName;

@Column(nullable = false, unique = true)
private String email;

// constructors
public Student() {}

public Student(String firstName, String lastName, String email) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
}

// getters and setters
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }
public String getFirstName() { return firstName; }
public void setFirstName(String firstName) { this.firstName = firstName; }
}

public String getLastName() { return lastName; }
public void setLastName(String lastName) { this.lastName = lastName; }
public String getEmail() { return email; }
public void setEmail(String email) { this.email = email; }
}

```

Note: Spring Boot 3 uses Jakarta (jakarta.persistence.*). If you're on older Spring Boot / Hibernate use javax.persistence.* and adjust the dependencies.

7) Repository

src/main/java/com/example/studentcrud/repository/StudentRepository.java

```

package com.example.studentcrud.repository;

import com.example.studentcrud.model.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface StudentRepository extends JpaRepository<Student, Long> {
    Optional<Student> findByEmail(String email);
}

```

Spring Data JPA provides CRUD operations out of the box.

8) Service Layer

src/main/java/com/example/studentcrud/service/StudentService.java

```
package com.example.studentcrud.service;

import com.example.studentcrud.model.Student;
import java.util.List;
import java.util.Optional;

public interface StudentService {
    List<Student> findAll();
    Optional<Student> findById(Long id);
    Student save(Student student);
    void deleteById(Long id);
}
```

src/main/java/com/example/studentcrud/service/impl/StudentServiceImpl.java

```
package com.example.studentcrud.service.impl;

import com.example.studentcrud.model.Student;
import com.example.studentcrud.repository.StudentRepository;
import com.example.studentcrud.service.StudentService;
import org.springframework.stereotype.Service;

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

@Service
public class StudentServiceImpl implements StudentService {

    private final StudentRepository repository;

    public StudentServiceImpl(StudentRepository repository) {
        this.repository = repository;
    }

    @Override
    public List<Student> findAll() {
        return repository.findAll();
    }

    @Override
    public Optional<Student> findById(Long id) {
        return repository.findById(id);
    }

    @Override
    public Student save(Student student) {
```

```

        return repository.save(student);
    }

    @Override
    public void deleteById(Long id) {
        repository.deleteById(id);
    }
}

```

9) Controller (MVC)

src/main/java/com/example/studentcrud/controller/StudentController.java

```

package com.example.studentcrud.controller;

import com.example.studentcrud.model.Student;
import com.example.studentcrud.service.StudentService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/students")
public class StudentController {

    private final StudentService service;

    public StudentController(StudentService service) {
        this.service = service;
    }

    @GetMapping
    public String listStudents(Model model) {
        model.addAttribute("students", service.findAll());
        return "students"; // src/main/resources/templates/students/list.html
    }

    @GetMapping("/new")
    public String showCreateForm(Model model) {
        model.addAttribute("student", new Student());
        return "students/form";
    }

    @PostMapping
    public String createStudent(@ModelAttribute Student student) {
        service.save(student);
        return "redirect:/students";
    }
}

```

```

@GetMapping("/edit/{id}")
public String showEditForm(@PathVariable Long id, Model model) {
    var studentOpt = service.findById(id);
    if (studentOpt.isEmpty()) {
        return "redirect:/students";
    }
    model.addAttribute("student", studentOpt.get());
    return "students/form";
}

@GetMapping("/delete/{id}")
public String deleteStudent(@PathVariable Long id) {
    service.deleteById(id);
    return "redirect:/students";
}
}

```

10) Thymeleaf Views

Create templates under src/main/resources/templates/students/.

list.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Students</title>
</head>
<body>
<h1>Students</h1>
<a th:href="@{/students/new}">Add Student</a>
<table border="1">
    <thead>
        <tr><th>ID</th><th>First</th><th>Last</th><th>Email</th><th>Actions</th></tr>
    </thead>
    <tbody>
        <tr th:each="s : ${students}">
            <td th:text="${s.id}"></td>
            <td th:text="${s.firstName}"></td>
            <td th:text="${s.lastName}"></td>
            <td th:text="${s.email}"></td>
            <td>
                <a th:href="@{'/students/edit/' + ${s.id}}">Edit</a>
                <form th:action="@{'/students/delete/' + ${s.id}}" method="post"
style="display:inline">
                    <button type="submit">Delete</button>

```

```

        </form>
    </td>
</tr>
</tbody>
</table>
</body>
</html>

```

form.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Student Form</title>
</head>
<body>
<h1 th:text="${student.id} != null ? 'Edit Student' : 'Add Student'"></h1>
<form th:action="@{/students}" th:object="${student}" method="post">
    <input type="hidden" th:field="*{id}" />
    <div>
        <label>First Name</label>
        <input th:field="*{firstName}" />
    </div>
    <div>
        <label>Last Name</label>
        <input th:field="*{lastName}" />
    </div>
    <div>
        <label>Email</label>
        <input th:field="*{email}" />
    </div>
    <div>
        <button type="submit">Save</button>
        <a th:href="@{/students}">Cancel</a>
    </div>
</form>
</body>
</html>

```

Note: We use the same form for create and edit. When editing, the id hidden field is set and save will perform an update because save() with an id will do merge.

11) Database Setup (MySQL)

Run in MySQL:

```

CREATE DATABASE student_db;
-- optionally create a dedicated user and grant privileges

```

With `spring.jpa.hibernate.ddl-auto=update`, tables will be created automatically when the app starts.

12) Run the Application

In Eclipse: - Run As → Spring Boot App (or Run As → Java Application selecting the main class).

Or using Maven CLI in project folder:

```
mvn spring-boot:run
```

Open <http://localhost:8080/students>

-----Note-----

Follow these steps in order to FIX Maven dependency download issue

✓ 1. First: Update Maven Project

In Eclipse:

Right-click project → Maven → Update Project (Alt + F5)

Check these options:

✓ *Force Update of Snapshots/Releases*

✓ *Also update projects*

Click **OK**.