**CRUD using Spring Boot & ReactJS**

### ****1. Spring Boot Setup:****

**1.1. Create the Spring Boot Project:**

You can use Spring Initializr to generate a Spring Boot project.

* Go to [Spring Initializr](https://start.spring.io/)
* Choose:
  + Project: Maven
  + Language: Java
  + Spring Boot version: 3.4.3 (or the latest stable version)
  + Group: com.rit
  + Artifact: studentcrud
  + Dependencies:
    - Spring Web
    - Spring Data JPA
    - MySql Database
    - Spring Boot DevTools (optional for development)

Once you generate the project, extract it and import it into your IDE.

**1.2. Define the Student Entity:**

In src/main/java/com/example/studentcrud/model/Student.java:

package com.example.studentcrud.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private int mark;

//generate constructors, getters & setters

}

**1.3. Create the Repository Interface:**

In 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;

public interface StudentRepository extends JpaRepository<Student, Long> {

}

**1.4. Create the Service Layer:**

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

package com.example.studentcrud.service;

import com.example.studentcrud.model.Student;

import com.example.studentcrud.repository.StudentRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class StudentService {

private final StudentRepository studentRepository;

@Autowired

public StudentService(StudentRepository studentRepository) {

this.studentRepository = studentRepository;

}

public List<Student> getAllStudents() {

return studentRepository.findAll();

}

public Optional<Student> getStudentById(Long id) {

return studentRepository.findById(id);

}

public Student createStudent(Student student) {

return studentRepository.save(student);

}

public Student updateStudent(Long id, Student studentDetails) {

Optional<Student> student = studentRepository.findById(id);

if (student.isPresent()) {

student.get().setName(studentDetails.getName());

student.get().setMark(studentDetails.getMark());

return studentRepository.save(student.get());

}

return null;

}

public boolean deleteStudent(Long id) {

if (studentRepository.existsById(id)) {

studentRepository.deleteById(id);

return true;

}

return false;

}

}

**1.5. Create the Controller Layer:**

In 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.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/students")

public class StudentController {

private final StudentService studentService;

@Autowired

public StudentController(StudentService studentService) {

this.studentService = studentService;

}

@GetMapping

public List<Student> getAllStudents() {

return studentService.getAllStudents();

}

@GetMapping("/{id}")

public ResponseEntity<Student> getStudentById(@PathVariable Long id) {

Optional<Student> student = studentService.getStudentById(id);

return student.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

public ResponseEntity<Student> createStudent(@RequestBody Student student) {

Student savedStudent = studentService.createStudent(student);

return new ResponseEntity<>(savedStudent, HttpStatus.CREATED);

}

@PutMapping("/{id}")

public ResponseEntity<Student> updateStudent(@PathVariable Long id, @RequestBody Student studentDetails) {

Student updatedStudent = studentService.updateStudent(id, studentDetails);

if (updatedStudent != null) {

return ResponseEntity.ok(updatedStudent);

}

return ResponseEntity.notFound().build();

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {

boolean isDeleted = studentService.deleteStudent(id);

if (isDeleted) {

return ResponseEntity.noContent().build();

}

return ResponseEntity.notFound().build();

}

}

**1.6. Configure Application Properties (H2 Database):**

In src/main/resources/application.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/sboot\_react\_db

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**1.7. Enabling CORS in Spring Boot:**

**Option 1: Global CORS Configuration**

You can define a global CORS configuration by creating a WebConfig class.

**Step 1: Create WebConfig.java class in your project.**

package com.example.studentcrud.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/api/\*\*") // Allows CORS for all endpoints starting with /api/

.allowedOrigins("http://localhost:5173") // Replace with your frontend URL

.allowedMethods("GET", "POST", "PUT", "DELETE")

.allowedHeaders("\*");

}

}

In this configuration:

* addMapping("/api/\*\*"): Allows CORS requests on all endpoints that start with /api/.
* allowedOrigins("http://localhost:5173"): Specifies the origin allowed to access the resources. Replace http://localhost:5173 with the actual URL of your frontend.
* allowedMethods("GET", "POST", "PUT", "DELETE"): Specifies the HTTP methods allowed.

.

**Option 2: Enable CORS on Specific Controller Methods (Less recommended)**

If you prefer, you can enable CORS only for specific controller methods. In that case, use the @CrossOrigin annotation directly in the controller.

For example, update StudentController.java:

@RestController

@RequestMapping("/api/students")

@CrossOrigin(origins = "http://localhost:5173") // Allow CORS for this controller

public class StudentController {

private final StudentService studentService;

…

}

This will apply CORS to all the endpoints in this controller with @CrossOrigin(origins = "http://localhost:5173").

### ****2. React Frontend Setup:****

**2.1. Create the React Project:**

Run the following commands to create a new React project:

npx create-react-app student-crud

cd student-crud

npm start

**2.2. Install Axios for HTTP Requests:**

npm install axios

**2.3. Create the Student CRUD Components:**

In src/components/StudentList.js:

import React, { useState, useEffect } from 'react';

import axios from 'axios';

const StudentList = () => {

    const [students, setStudents] = useState([]);

    const [newStudent, setNewStudent] = useState({ name: '', mark: '' });

    const [editStudent, setEditStudent] = useState(null);

    useEffect(() => {

        fetchStudents();

    }, []);

    const fetchStudents = async () => {

        const response = await axios.get('http://localhost:8080/api/students');

        setStudents(response.data);

    };

    const handleDelete = async (id) => {

        await axios.delete(`http://localhost:8080/api/students/${id}`);

        fetchStudents();

    };

    const handleCreate = async () => {

        await axios.post('http://localhost:8080/api/students', newStudent);

        setNewStudent({ name: '', mark: '' });

        fetchStudents();

    };

    const handleUpdate = async () => {

        await axios.put(`http://localhost:8080/api/students/${editStudent.id}`, editStudent);

        setEditStudent(null);

        fetchStudents();

    };

    const handleInputChange = (e) => {

        const { name, value } = e.target;

        if (editStudent) {

            setEditStudent({ ...editStudent, [name]: value });

        } else {

            setNewStudent({ ...newStudent, [name]: value });

        }

    };

    return (

        <div>

            <h1>Students List</h1>

            <table>

                <thead>

                    <tr>

                        <th>ID</th><th>Name</th><th>Mark</th><th>Delete</th><th>Edit</th>

                    </tr>

                </thead>

                <tbody>

                    {students.map((student) => (

                        <tr key={student.id}>

                            <td>{student.id}</td>

                            <td>{student.name}</td>

                            <td>{student.mark}</td>

                            <td><button onClick={() => handleDelete(student.id)}>Delete</button></td>

                            <td><button onClick={() => setEditStudent(student)}>Edit</button></td>

                        </tr>

                    ))}

                </tbody>

            </table>

            {!editStudent && (

                <div>

                    <h2>Create New Student</h2>

                    <input

                        type="text"

                        name="name"

                        value={newStudent.name}

                        onChange={handleInputChange}

                        placeholder="Name"

                    />

                    <input

                        type="number"

                        name="mark"

                        value={newStudent.mark}

                        onChange={handleInputChange}

                        placeholder="Mark"

                    />

                    <button onClick={handleCreate}>Create</button>

                </div>

            )}

            {editStudent && (

                <div>

                    <h2>Edit Student</h2>

                    <input

                        type="text"

                        name="name"

                        value={editStudent.name}

                        onChange={handleInputChange}

                    />

                    <input

                        type="number"

                        name="mark"

                        value={editStudent.mark}

                        onChange={handleInputChange}

                    />

                    <button onClick={handleUpdate}>Update</button>

                    <button onClick={() => setEditStudent(null)}>Cancel</button>

                </div>

            )}

        </div>

    );

};

export default StudentList;

**2.4. Update src/App.js to include StudentList:**

import { useState } from 'react'

import './App.css'

import StudentList from './components/StudentList'

function App() {

  const [count, setCount] = useState(0)

  return (

    <>

      <div className="App">

        <h1>Student CRUD App</h1>

        <StudentList />

      </div>

    </>

  )

}

export default App

### ****3. Run the Application:****

* **Spring Boot Backend**: Run mvn spring-boot:run in the Spring Boot project directory.
* **React Frontend**: Run npm start in the React project directory.

Now you have a fully functional Spring Boot backend and React frontend for managing students! You can create, read, update, and delete students using the application.

**React App with Router**

**Folder Structure**

/src

├── /components

│ ├── AddStudent.jsx

│ ├── StudentList.jsx

├── App.jsx

├── App.css

.env

**1. Setting Up the Environment Variables**

In the root folder, create a .env file with the following content:

VITE\_API\_URL=http://localhost:8080

This environment variable will store the base API URL, which is used to make HTTP requests to the backend.

**2. Install Required Dependencies**

You need to install the following dependencies for the app to work:

* react-router-dom for routing
* axios for making HTTP requests

Run the following command to install them:

npm install axios react-router-dom

**Components Overview**

There are two main components:

1. **AddStudent.jsx** - Used for both adding a new student and editing an existing one.
2. **StudentList.jsx** - Displays a list of all students with options to edit or delete them.

**Detailed Explanation of Components**

**3. AddStudent.jsx**

The AddStudent component is used to either create a new student or update an existing one based on whether the id is passed in the URL.

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import { useNavigate, useParams } from 'react-router-dom';

const AddStudent = () => {

const API\_URL = import.meta.env.VITE\_API\_URL;

const [student, setStudent] = useState({ name: '', mark: '' });

const { id } = useParams(); // Get student ID if editing an existing student

const navigate = useNavigate();

useEffect(() => {

if (id) {

// Fetch student data if an id exists (editing mode)

axios.get(`${API\_URL}/api/students/${id}`)

.then(response => {

setStudent(response.data);

})

.catch(error => console.error("There was an error fetching student details", error));

}

}, [id]);

const handleChange = (e) => {

const { name, value } = e.target;

setStudent(prevState => ({ ...prevState, [name]: value }));

};

const handleSubmit = (e) => {

e.preventDefault();

if (id) {

// Update existing student

axios.put(`${API\_URL}/api/students/${id}`, student)

.then(() => navigate('/'))

.catch(error => console.error("Error updating student", error));

} else {

// Create new student

axios.post(`${API\_URL}/api/students`, student)

.then(() => navigate('/'))

.catch(error => console.error("Error creating student", error));

}

};

return (

<div>

<h2>{id ? "Edit Student" : "Add New Student"}</h2>

<form onSubmit={handleSubmit}>

<div>

<label>Name:</label>

<input

type="text"

name="name"

value={student.name}

onChange={handleChange}

required

/>

</div>

<div>

<label>Mark:</label>

<input

type="number"

name="mark"

value={student.mark}

onChange={handleChange}

required

/>

</div>

<button type="submit">{id ? "Update" : "Add"}</button>

</form>

</div>

);

};

export default AddStudent;

**Key Points:**

* **State Management:** The student state holds the details of the student (name, mark).
* **Form Handling:** The handleSubmit function determines if the student is being added or edited by checking if an id exists in the URL.
* **API Requests:**
  + GET to fetch student data for editing.
  + POST to create a new student.
  + PUT to update an existing student.
* **Form Input:** The form has inputs for name and mark, both of which are controlled by React state.

**4. StudentList.jsx**

The StudentList component fetches and displays the list of students from the backend. It allows users to edit or delete students.

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import { Link } from 'react-router-dom';

const StudentList = () => {

const API\_URL = import.meta.env.VITE\_API\_URL;

const [students, setStudents] = useState([]);

useEffect(() => {

fetchStudents();

}, []);

const fetchStudents = async () => {

const response = await axios.get(`${API\_URL}/api/students`);

setStudents(response.data);

};

const handleDelete = async (id) => {

await axios.delete(`${API\_URL}/api/students/${id}`);

fetchStudents(); // Refresh the list after deleting

};

return (

<div>

<h1>Students List</h1>

<Link to="/add">

<button>Add New Student</button>

</Link>

<ul>

{students.map((student) => (

<li key={student.id}>

{student.name} - {student.mark}

<Link to={`/edit/${student.id}`}>

<button>Edit</button>

</Link>

<button onClick={() => handleDelete(student.id)}>Delete</button>

</li>

))}

</ul>

</div>

);

};

export default StudentList;

**Key Points:**

* **Fetching Students:** The useEffect hook calls the fetchStudents function to fetch data from the backend when the component is mounted.
* **Deleting Students:** The handleDelete function sends a DELETE request to remove a student, followed by a call to fetchStudents to refresh the list.
* **Edit Link:** A Link is used to navigate to the /edit/:id route for editing a student.

**5. App.jsx**

The App component is the entry point to the application. It sets up the routing using react-router-dom.

import { useState } from 'react';

import './App.css';

import { Route, BrowserRouter as Router, Routes } from 'react-router-dom';

import StudentList from './components/StudentList';

import AddStudent from './components/AddStudent';

function App() {

const [count, setCount] = useState(0);

return (

<Router>

<div className="App">

<Routes>

<Route path="/" element={<StudentList />} />

<Route path="/add" element={<AddStudent />} />

<Route path="/edit/:id" element={<AddStudent />} /> {/\* Route for editing student \*/}

</Routes>

</div>

</Router>

);

}

export default App;

**Key Points:**

* **Routing:** The Routes component is used to define the paths and their associated components:
  + /: Displays the list of students (StudentList).
  + /add: Displays the form to add a new student (AddStudent).
  + /edit/:id: Displays the form to edit an existing student (AddStudent).