Building a Robust REST API with Spring Boot, H2 Database, and Exception Handling

Github link: [hacker123shiva/Working-with-h2-database: Building a Robust REST API with Spring Boot, H2 Database, and Exception Handling (github.com)]
LinkedId: https://www.linkedin.com/in/shivasrivastava1/

Introduction

In this blog, we'll build a REST API using Spring Boot, the H2 in-memory database, and Spring Data JPA. We'll also cover how to handle exceptions gracefully and test our API with Postman. Let's dive in!

1. Project Structure

We will follow this project structure:

```
src
├── main
│   ├── java
│   │   └── com
│   │       └── telusko
│   │           ├── controller
│   │           │   └── StudentController.java
│   │           ├── dao
│   │           │   └── StudentRepository.java
│   │           ├── entity
│   │           │   └── Student.java
│   │           ├── exception
│   │           │   ├── GlobalExceptionHandler.java
│   │           │   ├── StudentNotFoundException.java
│   │           │   └── ErrorResponse.java
│   │           └── service
│   │               └── StudentService.java
│   └── resources
│       ├── application.properties
│       └── static
│       └── templates
└── test
```

```
    └── java
```

2. Dependencies and Pom.xml file

Add the following dependencies to your `pom.xml`:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.3.4</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.telusko</groupId>
	<artifactId>shiva</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>h2-database-1</name>
	<description>This is feature of restclient</description>
	<url/>
	<licenses>
		<license/>
	</licenses>
	<developers>
		<developer/>
	</developers>
	<scm>
		<connection/>
		<developerConnection/>
		<tag/>
		<url/>
	</scm>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
```

```
                <artifactId>spring-boot-starter-data-jpa</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>com.h2database</groupId>
                <artifactId>h2</artifactId>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
            </dependency>
            <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>
                    <configuration>
                        <excludes>
                            <exclude>

<groupId>org.projectlombok</groupId>
                                <artifactId>lombok</artifactId>
                            </exclude>
                        </excludes>
                    </configuration>
                </plugin>
```

```
            </plugins>
        </build>
</project>
```

3. H2 Database Configuration

File: `application.properties`

```
spring.application.name=h2-database-1

# H2 Database Configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.sql.init.platform=h2

# Enable the H2 console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# JPA / Hibernate Configuration
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

server.port=9999
```

4. Entity Class: `Student`

File: `Student.java`

```
package com.telusko.entity;

import jakarta.persistence.*;
```

```java
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "StudentTable")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"student_seq")
    @SequenceGenerator(name = "student_seq", sequenceName =
"student_sequence", allocationSize = 1, initialValue = 2115800000)
    private int id;

    private String name;

    private String email;

    @Transient
    private String dob; // Not persisted in the database
}
```

Explanation of Annotations:

- `@Entity`: Marks this class as a JPA entity.
- `@Table`: Specifies the table name in the database.
- `@Id`: Indicates the primary key.
- `@GeneratedValue`: Automatically generates values for the primary key.
- `@SequenceGenerator`: Configures the primary key generation strategy.
- `@Transient`: Indicates that this field should not be persisted.

5. DAO Interface: `StudentRepository`

File: `StudentRepository.java`

```java
package com.telusko.dao;
```

```java
import org.springframework.data.jpa.repository.JpaRepository;
import com.telusko.entity.Student;

// DAO interface for Student
public interface StudentRepository extends JpaRepository<Student, Integer>
{
}
```

Explanation:

- Extends `JpaRepository` to provide CRUD operations for the `Student` entity.

6. Service Layer: `StudentService`

File: `StudentService.java`

```java
package com.telusko.service;

import com.telusko.dao.StudentRepository;
import com.telusko.entity.Student;
import com.telusko.exception.StudentNotFoundException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class StudentService {

    @Autowired
    private StudentRepository studentRepository;

    public List<Student> getAllStudents() {
        return studentRepository.findAll();
    }

    public Student getStudentById(int id) {
        return studentRepository.findById(id)
                .orElseThrow(() -> new StudentNotFoundException("Student
with ID " + id + " not found."));
```

```
    }

    public Student saveStudent(Student student) {
        return studentRepository.save(student);
    }

    public void deleteStudent(int id) {
        if (!studentRepository.existsById(id)) {
            throw new StudentNotFoundException("Student with ID " + id + "
not found.");
        }
        studentRepository.deleteById(id);
    }
}
```

Explanation:

- Contains business logic and interacts with the `StudentRepository`.
- Throws `StudentNotFoundException` when a student is not found.

7. Controller Layer: `StudentController`

File: `StudentController.java`

```
package com.telusko.controller;

import com.telusko.entity.Student;
import com.telusko.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

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

    @Autowired
    private StudentService studentService;

    // Create a new student (POST)
```

```
    @PostMapping
    public Student createStudent(@RequestBody Student student) {
        return studentService.saveStudent(student);
    }

    // Get all students (GET)
    @GetMapping
    public List<Student> getAllStudents() {
        return studentService.getAllStudents();
    }

    // Get student by ID (GET)
    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable int id) {
        return studentService.getStudentById(id);
    }

    // Update an existing student (PUT)
    @PutMapping("/{id}")
    public Student updateStudent(@PathVariable int id, @RequestBody Student
student) {
        student.setId(id); // Ensure the ID is set to update the correct
student
        return studentService.saveStudent(student);
    }

    // Delete a student by ID (DELETE)
    @DeleteMapping("/{id}")
    public void deleteStudent(@PathVariable int id) {
        studentService.deleteStudent(id);
    }
}
```

Explanation:

- Provides endpoints for CRUD operations.
- Uses `@RequestBody` to map incoming JSON to `Student` objects.
- Uses `@PathVariable` to retrieve path parameters.

8. Exception Handling

File: `StudentNotFoundException.java`

```java
package com.telusko.exception;

public class StudentNotFoundException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public StudentNotFoundException(String message) {
        super(message);
    }
}
```

File: `GlobalExceptionHandler.java`

```java
package com.telusko.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(StudentNotFoundException.class)
    public ResponseEntity<ErrorResponse>
handleStudentNotFoundException(StudentNotFoundException ex) {
        ErrorResponse errorResponse = new ErrorResponse(ex.getMessage(),
HttpStatus.NOT_FOUND.value());
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(errorResponse);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleGenericException(Exception
ex) {
        ErrorResponse errorResponse = new ErrorResponse("An unexpected
error occurred: " + ex.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR.value());
        return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(errorResponse)
;
    }
```

```
}
```

File: `ErrorResponse.java`

```java
package com.telusko.exception;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorResponse {
    private String message;
    private int status;
}
```

Explanation:

- `GlobalExceptionHandler` handles exceptions globally using `@ControllerAdvice`.
- `ErrorResponse` standardizes the error messages sent to the client.

9. Testing CRUD Operations with Postman

Getting Started with Postman

1. **Download Postman**:
    - Go to the Postman website and download the application for your operating system.
    - Install and open Postman.
2. **Start Your Spring Boot Application**:
    - Ensure your Spring Boot application is running. You should see console output indicating that the server is started, usually at `http://localhost:9999`.

1. Create a New Student (POST)

- **Step 1**: Open Postman and create a new request.
- **Step 2**: Set the request type to `POST`.
- **Step 3**: Enter the URL: `http://localhost:9999/students`.
- **Step 4**: Go to the "Body" tab.

- **Step 5**: Select `raw` and choose `JSON` from the dropdown.

Step 6: Enter the following JSON in the body:

```
{
    "name": "Shiva Srivastava",
    "email": "shiva@gmail.com",
    "dob": "2002-11-13"
}
```

- **Step 7**: Click `Send`.
- **Step 8**: Check the response status (should be `200 OK`) and view the created student object.

2. Retrieve All Students (GET)

- **Step 1**: Create a new request.
- **Step 2**: Set the request type to `GET`.
- **Step 3**: Enter the URL: `http://localhost:9999/students`.
- **Step 4**: Click `Send`.
- **Step 5**: Check the response status (should be `200 OK`) and view the list of students.

3. Retrieve a Student by ID (GET)

- **Step 1**: Create a new request.
- **Step 2**: Set the request type to `GET`.
- **Step 3**: Enter the URL: `http://localhost:9999/students/{id}` (replace `{id}` with the actual student ID, e.g., `1`).
- **Step 4**: Click `Send`.
- **Step 5**: Check the response status (should be `200 OK`) and view the student details.

4. Update a Student (PUT)

- **Step 1**: Create a new request.
- **Step 2**: Set the request type to `PUT`.
- **Step 3**: Enter the URL: `http://localhost:9999/students/{id}` (replace `{id}` with the actual student ID).
- **Step 4**: Go to the "Body" tab.
- **Step 5**: Select `raw` and choose `JSON`.

Step 6: Enter the updated JSON in the body:

```
{
    "name": "Shiva Srivastava",
    "email": "shivasrivastava@gmail.com",
    "dob": "2002-11-13"
}
```

- **Step 7**: Click `Send`.
- **Step 8**: Check the response status (should be `200 OK`) and view the updated student object.

5. Delete a Student (DELETE)

- **Step 1**: Create a new request.
- **Step 2**: Set the request type to `DELETE`.
- **Step 3**: Enter the URL: `http://localhost:9999/students/{id}` (replace `{id}` with the actual student ID).
- **Step 4**: Click `Send`.
- **Step 5**: Check the response status (should be `204 No Content`), indicating successful deletion.

Conclusion

In this blog, we explored how to build a REST API using Spring Boot, H2 Database, and Spring Data JPA. We handled exceptions gracefully and tested our API with Postman. This robust architecture will help you efficiently manage student data and improve your skills with Spring Boot.