**Exercise 1: Online Bookstore - Setting Up RESTful Services**

**Business Scenario:**

You are tasked with developing a RESTful service for an online bookstore. The service will manage books, authors, and customers.

**Instructions:**

1. **Setup Spring Boot Project:**
   * Initialize a new Spring Boot project named **BookstoreAPI**.
   * Add dependencies: **Spring Web, Spring Boot DevTools, Lombok**.
2. **Project Structure:**
   * Familiarize yourself with the generated project structure.
3. **What's New in Spring Boot 3:**
   * Explore and document the new features introduced in Spring Boot 3.

**BookstoreapiApplication.java:**

package com.example.bookstoreapi;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreapiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreapiApplication.class, args);

}

}

**Book.java:**

package com.example.bookstoreapi.model;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Data

@NoArgsConstructor

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private Double price;

}

**BookRepository.java:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}

**BookController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book book) {

book.setId(id);

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

application.properties:

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

pom.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 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstoreapi</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>BookstoreAPI</name>

<description>Online Bookstore RESTful API</description>

<packaging>jar</packaging>

<properties>

<java.version>21</java.version>

<spring-boot.version>3.4.0</spring-boot.version> <!-- Adjust to the latest version if needed -->

<lombok.version>1.18.24</lombok.version>

<h2.version>2.2.220</h2.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>${spring-boot.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools for development -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>${lombok.version}</version>

<scope>provided</scope>

</dependency>

<!-- H2 Database for in-memory database support -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>${h2.version}</version>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Data JPA for data access -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Exercise 2: Online Bookstore - Creating Basic REST Controllers**

**Business Scenario:**

Implement RESTful endpoints to manage books.

**Instructions:**

1. **Create Book Controller:**
   * Define a **BookController** class with request mappings for /books.
2. **Handle HTTP Methods:**
   * Implement methods to handle **GET**, **POST**, **PUT**, and **DELETE** requests.
3. **Return JSON Responses:**
   * Ensure the controller returns JSON responses.

Define the Book entity with attributes like **id, title, author, price**, and **isbn**

**Book.java:**

package com.example.bookstoreapi.model;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Data

@NoArgsConstructor

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private Double price;

private String isbn; // New attribute

}

**BookController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

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("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id) {

Optional<Book> book = bookRepository.findById(id);

return book.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

public ResponseEntity<Book> addBook(@RequestBody Book book) {

Book savedBook = bookRepository.save(book);

return ResponseEntity.status(HttpStatus.CREATED).body(savedBook);

}

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book book) {

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

book.setId(id);

Book updatedBook = bookRepository.save(book);

return ResponseEntity.ok(updatedBook);

}

@DeleteMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}

**BookRepository.java:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}

BookstoreapiApplication.java:

package com.example.bookstoreapi;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreapiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreapiApplication.class, args);

}

}

**application.properties:**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

pom.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 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstoreapi</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>BookstoreAPI</name>

<description>Online Bookstore RESTful API</description>

<packaging>jar</packaging>

<properties>

<java.version>21</java.version>

<spring-boot.version>3.4.0</spring-boot.version>

<lombok.version>1.18.24</lombok.version>

<h2.version>2.2.220</h2.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>${spring-boot.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools for development -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>${lombok.version}</version>

<scope>provided</scope>

</dependency>

<!-- H2 Database for in-memory database support -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>${h2.version}</version>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Data JPA for data access -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Exercise 3: Online Bookstore - Handling Path Variables and Query Parameters**

**Business Scenario:**

Enhance the book management endpoints to handle dynamic URLs and query parameters.

**Instructions:**

1. **Path Variables:**
   1. Implement an endpoint to fetch a book by its ID using a path variable.
2. **Query Parameters:**
   1. Implement an endpoint to filter books based on query parameters like title and author.

BookstoreapiApplication.java:

package com.example.bookstoreapi;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreapiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreapiApplication.class, args);

}

}

Book.java:

package com.example.bookstoreapi.model;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Data

@NoArgsConstructor

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private Double price;

private String isbn; // New attribute

}

**BookRepository.java:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByTitle(String title);

List<Book> findByAuthor(String author);

List<Book> findByTitleAndAuthor(String title, String author);

}

**BookController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

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("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

// Get all books with optional filtering

@GetMapping

public List<Book> getAllBooks(

@RequestParam(required = false) String title,

@RequestParam(required = false) String author) {

if (title != null && author != null) {

return bookRepository.findByTitleAndAuthor(title, author);

} else if (title != null) {

return bookRepository.findByTitle(title);

} else if (author != null) {

return bookRepository.findByAuthor(author);

}

return bookRepository.findAll();

}

// Get book by ID

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id) {

Optional<Book> book = bookRepository.findById(id);

return book.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Add a new book

@PostMapping

public ResponseEntity<Book> addBook(@RequestBody Book book) {

Book savedBook = bookRepository.save(book);

return ResponseEntity.status(HttpStatus.CREATED).body(savedBook);

}

// Update an existing book

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book book) {

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

book.setId(id);

Book updatedBook = bookRepository.save(book);

return ResponseEntity.ok(updatedBook);

}

// Delete a book

@DeleteMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}

application.properties:

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

pom.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 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstoreapi</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>BookstoreAPI</name>

<description>Online Bookstore RESTful API</description>

<packaging>jar</packaging>

<properties>

<java.version>21</java.version>

<spring-boot.version>3.4.0</spring-boot.version>

<lombok.version>1.18.24</lombok.version>

<h2.version>2.2.220</h2.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>${spring-boot.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools for development -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>${lombok.version}</version>

<scope>provided</scope>

</dependency>

<!-- H2 Database for in-memory database support -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>${h2.version}</version>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Data JPA for data access -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Exercise 4: Online Bookstore - Processing Request Body and Form Data**

**Business Scenario:**

Create endpoints to accept and process JSON request bodies and form data for customer registrations.

**Instructions:**

1. **Request Body:**
   * Implement a POST endpoint to create a new customer by accepting a JSON request body.
2. **Form Data:**
   * Implement an endpoint to process form data for customer registrations.

**Customer.java:**

package com.example.bookstoreapi.model;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Data

@NoArgsConstructor

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String phone;

private String address;

}

**CustomerRepository.java:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.model.Customer;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

**CustomerController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Customer;

import com.example.bookstoreapi.repository.CustomerRepository;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

import java.util.Map;

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerRepository customerRepository;

// Create a new customer with JSON request body

@PostMapping

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {

Customer savedCustomer = customerRepository.save(customer);

return ResponseEntity.status(HttpStatus.CREATED).body(savedCustomer);

}

// Create a new customer with form data

@PostMapping("/register")

public ResponseEntity<Customer> registerCustomer(

@RequestParam String name,

@RequestParam String email,

@RequestParam String phone,

@RequestParam String address) {

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customer.setPhone(phone);

customer.setAddress(address);

Customer savedCustomer = customerRepository.save(customer);

return ResponseEntity.status(HttpStatus.CREATED).body(savedCustomer);

}

}

application.properties:

# Spring Boot configuration for file uploads

spring.servlet.multipart.max-file-size=2MB

spring.servlet.multipart.max-request-size=2MB

# Other existing configurations

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

pom.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 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstoreapi</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>BookstoreAPI</name>

<description>Online Bookstore RESTful API</description>

<packaging>jar</packaging>

<properties>

<java.version>21</java.version>

<spring-boot.version>3.4.0</spring-boot.version>

<lombok.version>1.18.24</lombok.version>

<h2.version>2.2.220</h2.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>${spring-boot.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools for development -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>${lombok.version}</version>

<scope>provided</scope>

</dependency>

<!-- H2 Database for in-memory database support -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>${h2.version}</version>

<scope>runtime</scope>

BookstoreapiApplication.java:

package com.example.bookstoreapi;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreapiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreapiApplication.class, args);

}

**Exercise 5: Online Bookstore - Customizing Response Status and Headers**

**Business Scenario:**

Customize the HTTP response status and headers for the book management endpoints.

**Instructions:**

1. **Response Status:**
   * Use **@ResponseStatus** to customize HTTP status codes for your endpoints.
2. **Custom Headers:**
   * Add custom headers to the response using **ResponseEntity**.

**BookController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

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

import org.springframework.http.HttpHeaders;

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("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

// Get all books

@GetMapping

public ResponseEntity<List<Book>> getAllBooks() {

List<Book> books = bookRepository.findAll();

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "CustomHeaderValue");

return new ResponseEntity<>(books, headers, HttpStatus.OK);

}

// Get book by ID

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id) {

Optional<Book> book = bookRepository.findById(id);

if (book.isPresent()) {

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookFound");

return new ResponseEntity<>(book.get(), headers, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

// Add a new book

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public ResponseEntity<Book> addBook(@RequestBody Book book) {

Book savedBook = bookRepository.save(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookCreated");

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

}

// Update an existing book

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book book) {

if (!bookRepository.existsById(id)) {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

book.setId(id);

Book updatedBook = bookRepository.save(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookUpdated");

return new ResponseEntity<>(updatedBook, headers, HttpStatus.OK);

}

// Delete a book

@DeleteMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

bookRepository.deleteById(id);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookDeleted");

return new ResponseEntity<>(headers, HttpStatus.NO\_CONTENT);

}

}

**CustomerController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Customer;

import com.example.bookstoreapi.repository.CustomerRepository;

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

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerRepository customerRepository;

// Create a new customer with JSON request body

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {

Customer savedCustomer = customerRepository.save(customer);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "CustomerCreated");

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

}

// Create a new customer with form data

@PostMapping("/register")

public ResponseEntity<Customer> registerCustomer(

@RequestParam String name,

@RequestParam String email,

@RequestParam String phone,

@RequestParam String address) {

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customer.setPhone(phone);

customer.setAddress(address);

Customer savedCustomer = customerRepository.save(customer);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "CustomerRegistered");

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

}

}

BookstoreapiApplication.java:

package com.example.bookstoreapi;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreapiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreapiApplication.class, args);

}

}

Book.java:

package com.example.bookstoreapi.model;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Data

@NoArgsConstructor

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private Double price;

private String isbn;

}

**Customer.java**:

package com.example.bookstoreapi.model;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Data

@NoArgsConstructor

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String phone;

private String address;

}

**BookRepository.java:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}

**CustomerRepository.java:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.model.Customer;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

application.properties:

# Spring Boot configuration for file uploads

spring.servlet.multipart.max-file-size=2MB

spring.servlet.multipart.max-request-size=2MB

# Database configuration

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

pom.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 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstoreapi</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>BookstoreAPI</name>

<description>Online Bookstore RESTful API</description>

<packaging>jar</packaging>

<properties>

<java.version>21</java.version>

<spring-boot.version>3.4.0</spring-boot.version>

<lombok.version>1.18.24</lombok.version>

<h2.version>2.2.220</h2.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>${spring-boot.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools for development -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>${lombok.version}</version>

<scope>provided</scope>

</dependency>

<!-- H2 Database for in-memory database support -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>${h2.version}</version>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Test for testing -->

<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>

**Exercise 6: Online Bookstore - Exception Handling in REST Controllers**

**Business Scenario:**

Implement a global exception handling mechanism for the bookstore RESTful services.

**Instructions:**

1. **Global Exception Handler:**
   * Create a **GlobalExceptionHandler** class using **@ControllerAdvice**.
   * Define methods to handle various exceptions and return appropriate HTTP status codes.

**GlobalExceptionHandler.java:**

package com.example.bookstoreapi.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;

import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice

public class GlobalExceptionHandler {

// Handle Resource Not Found Exception

@ExceptionHandler(ResourceNotFoundException.class)

@ResponseStatus(HttpStatus.NOT\_FOUND)

public ResponseEntity<ErrorResponse> handleResourceNotFoundException(ResourceNotFoundException ex) {

ErrorResponse errorResponse = new ErrorResponse(HttpStatus.NOT\_FOUND.value(), ex.getMessage());

return new ResponseEntity<>(errorResponse, HttpStatus.NOT\_FOUND);

}

// Handle Bad Request Exception

@ExceptionHandler(BadRequestException.class)

@ResponseStatus(HttpStatus.BAD\_REQUEST)

public ResponseEntity<ErrorResponse> handleBadRequestException(BadRequestException ex) {

ErrorResponse errorResponse = new ErrorResponse(HttpStatus.BAD\_REQUEST.value(), ex.getMessage());

return new ResponseEntity<>(errorResponse, HttpStatus.BAD\_REQUEST);

}

// Handle Generic Exception

@ExceptionHandler(Exception.class)

@ResponseStatus(HttpStatus.INTERNAL\_SERVER\_ERROR)

public ResponseEntity<ErrorResponse> handleException(Exception ex) {

ErrorResponse errorResponse = new ErrorResponse(HttpStatus.INTERNAL\_SERVER\_ERROR.value(), "An unexpected error occurred");

return new ResponseEntity<>(errorResponse, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

**ErrorResponse.java:**

package com.example.bookstoreapi.exception;

import lombok.AllArgsConstructor;

import lombok.Data;

@Data

@AllArgsConstructor

public class ErrorResponse {

private int status;

private String message;

}

ResourceNotFoundException.java:

package com.example.bookstoreapi.exception;

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

BadRequestException.java:

package com.example.bookstoreapi.exception;

public class BadRequestException extends RuntimeException {

public BadRequestException(String message) {

super(message);

}

}

**BookController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.exception.BadRequestException;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

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

import org.springframework.http.HttpHeaders;

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("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

// Get all books

@GetMapping

public ResponseEntity<List<Book>> getAllBooks() {

List<Book> books = bookRepository.findAll();

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "CustomHeaderValue");

return new ResponseEntity<>(books, headers, HttpStatus.OK);

}

// Get book by ID

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id) {

Optional<Book> book = bookRepository.findById(id);

if (book.isPresent()) {

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookFound");

return new ResponseEntity<>(book.get(), headers, HttpStatus.OK);

} else {

throw new ResourceNotFoundException("Book with ID " + id + " not found");

}

}

// Add a new book

@PostMapping

public ResponseEntity<Book> addBook(@RequestBody Book book) {

if (book.getTitle() == null || book.getAuthor() == null) {

throw new BadRequestException("Book title and author are required");

}

Book savedBook = bookRepository.save(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookCreated");

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

}

// Update an existing book

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book book) {

if (!bookRepository.existsById(id)) {

throw new ResourceNotFoundException("Book with ID " + id + " not found");

}

if (book.getTitle() == null || book.getAuthor() == null) {

throw new BadRequestException("Book title and author are required");

}

book.setId(id);

Book updatedBook = bookRepository.save(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookUpdated");

return new ResponseEntity<>(updatedBook, headers, HttpStatus.OK);

}

// Delete a book

@DeleteMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

throw new ResourceNotFoundException("Book with ID " + id + " not found");

}

bookRepository.deleteById(id);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookDeleted");

return new ResponseEntity<>(headers, HttpStatus.NO\_CONTENT);

}

}

**Exercise 7: Online Bookstore - Introduction to Data Transfer Objects (DTOs)**

**Business Scenario:**

Use DTOs to transfer data between the client and server for books and customers.

**Instructions:**

1. **Create DTOs:**
   * Define BookDTO and CustomerDTO classes.
2. **Mapping Entities to DTOs:**
   * Use a library like **MapStruct** or **ModelMapper** to map entities to DTOs and vice versa.
3. **Custom Serialization/Deserialization:**
   * Customize JSON serialization and deserialization using Jackson annotations.

**BookDTO.java:**

package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data

public class BookDTO {

private Long id;

@JsonProperty("book\_title")

private String title;

private String author;

private Double price;

@JsonIgnore

private String isbn;

}

**CustomerDTO.java:**

package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data

public class CustomerDTO {

private Long id;

@JsonProperty("full\_name")

private String name;

private String email;

private String phone;

private String address;

}

**BookMapper.java:**

package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface BookMapper {

BookMapper INSTANCE = Mappers.getMapper(BookMapper.class);

BookDTO toDTO(Book book);

Book toEntity(BookDTO bookDTO);

}

**CustomerMapper.java:**

package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.model.Customer;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface CustomerMapper {

CustomerMapper INSTANCE = Mappers.getMapper(CustomerMapper.class);

CustomerDTO toDTO(Customer customer);

Customer toEntity(CustomerDTO customerDTO);

}

**BookDTO.java** (Customized):

package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data

public class BookDTO {

private Long id;

@JsonProperty("book\_title")

private String title;

private String author;

private Double price;

@JsonIgnore

private String isbn;

}

**CustomerDTO.java** (Customized):

package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data

public class CustomerDTO {

private Long id;

@JsonProperty("full\_name")

private String name;

private String email;

private String phone;

private String address;

}

**BookController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.mapper.BookMapper;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

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

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

private final BookMapper bookMapper = BookMapper.INSTANCE;

@GetMapping

public ResponseEntity<List<BookDTO>> getAllBooks() {

List<BookDTO> books = bookRepository.findAll()

.stream()

.map(bookMapper::toDTO)

.collect(Collectors.toList());

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "CustomHeaderValue");

return new ResponseEntity<>(books, headers, HttpStatus.OK);

}

@GetMapping("/{id}")

public ResponseEntity<BookDTO> getBookById(@PathVariable Long id) {

Optional<Book> book = bookRepository.findById(id);

if (book.isPresent()) {

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookFound");

return new ResponseEntity<>(bookMapper.toDTO(book.get()), headers, HttpStatus.OK);

} else {

throw new ResourceNotFoundException("Book with ID " + id + " not found");

}

}

@PostMapping

public ResponseEntity<BookDTO> addBook(@RequestBody BookDTO bookDTO) {

Book book = bookMapper.toEntity(bookDTO);

Book savedBook = bookRepository.save(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookCreated");

return new ResponseEntity<>(bookMapper.toDTO(savedBook), headers, HttpStatus.CREATED);

}

@PutMapping("/{id}")

public ResponseEntity<BookDTO> updateBook(@PathVariable Long id, @RequestBody BookDTO bookDTO) {

if (!bookRepository.existsById(id)) {

throw new ResourceNotFoundException("Book with ID " + id + " not found");

}

Book book = bookMapper.toEntity(bookDTO);

book.setId(id);

Book updatedBook = bookRepository.save(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookUpdated");

return new ResponseEntity<>(bookMapper.toDTO(updatedBook), headers, HttpStatus.OK);

}

@DeleteMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

throw new ResourceNotFoundException("Book with ID " + id + " not found");

}

bookRepository.deleteById(id);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookDeleted");

return new ResponseEntity<>(headers, HttpStatus.NO\_CONTENT);

}

}

**CustomerController.java:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.mapper.CustomerMapper;

import com.example.bookstoreapi.model.Customer;

import com.example.bookstoreapi.repository.CustomerRepository;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

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

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerRepository customerRepository;

private final CustomerMapper customerMapper = CustomerMapper.INSTANCE;

@PostMapping

public ResponseEntity<CustomerDTO> createCustomer(@RequestBody CustomerDTO customerDTO) {

Customer customer = customerMapper.toEntity(customerDTO);

Customer savedCustomer = customerRepository.save(customer);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "CustomerCreated");

return new ResponseEntity<>(customerMapper.toDTO(savedCustomer), headers, HttpStatus.CREATED);

}

@PostMapping("/register")

public ResponseEntity<CustomerDTO> registerCustomer(

@RequestParam String name,

@RequestParam String email,

@RequestParam String phone,

@RequestParam String address) {

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customer.setPhone(phone);

customer.setAddress(address);

Customer savedCustomer = customerRepository.save(customer);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "CustomerRegistered");

return new ResponseEntity<>(customerMapper.toDTO(savedCustomer), headers, HttpStatus.CREATED);

}

}

Pom.xml:

<project>

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstoreapi</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<java.version>21</java.version>

<mapstruct.version>1.5.3.Final</mapstruct.version>

<lombok.version>1.18.24</lombok.version>

<spring.boot.version>3.3.1</spring.boot.version>

</properties>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>${lombok.version}</version>

<scope>provided</scope>

</dependency>

<!-- MapStruct for mapping DTOs -->

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct</artifactId>

<version>${mapstruct.version}</version>

</dependency>

<!-- MapStruct annotation processor -->

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

<version>${mapstruct.version}</version>

<scope>provided</scope>

</dependency>

<!-- Jackson for JSON serialization/deserialization -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- H2 database for development -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- JUnit Jupiter API for testing -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- Spring Boot Starter 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>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<annotationProcessorPaths>

<path>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

</path>

</annotationProcessorPaths>

</configuration>

</plugin>

</plugins>

</build>

</project>