**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.

*Main class:*

package com.yourname.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);

}

}

Model classes:

Book.java

package com.yourname.bookstoreapi.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Book {

private Long id;

private String title;

private String isbn;

private String author;

}

Author.java

package com.yourname.bookstoreapi.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Author {

private Long id;

private String name;

private String bio;

}

Customer.java

package com.yourname.bookstoreapi.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Customer {

private Long id;

private String name;

private String email;

}

Controller classes:

BookController.java

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Book;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> books = new ArrayList<>();

@GetMapping

public List<Book> getAllBooks() {

return books;

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.orElse(null);

}

@PostMapping

public Book createBook(@RequestBody Book book) {

books.add(book);

return book;

}

@PutMapping("/{id}")

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

books.stream()

.filter(book -> book.getId().equals(id))

.forEach(book -> {

book.setTitle(updatedBook.getTitle());

book.setIsbn(updatedBook.getIsbn());

book.setAuthor(updatedBook.getAuthor());

});

return updatedBook;

}

@DeleteMapping("/{id}")

public String deleteBook(@PathVariable Long id) {

books.removeIf(book -> book.getId().equals(id));

return "Book with ID " + id + " deleted.";

}

}

AuthorController.java

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Author;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/authors")

public class AuthorController {

private List<Author> authors = new ArrayList<>();

@GetMapping

public List<Author> getAllAuthors() {

return authors;

}

@GetMapping("/{id}")

public Author getAuthorById(@PathVariable Long id) {

return authors.stream()

.filter(author -> author.getId().equals(id))

.findFirst()

.orElse(null);

}

@PostMapping

public Author createAuthor(@RequestBody Author author) {

authors.add(author);

return author;

}

@PutMapping("/{id}")

public Author updateAuthor(@PathVariable Long id, @RequestBody Author updatedAuthor) {

authors.stream()

.filter(author -> author.getId().equals(id))

.forEach(author -> {

author.setName(updatedAuthor.getName());

author.setBio(updatedAuthor.getBio());

});

return updatedAuthor;

}

@DeleteMapping("/{id}")

public String deleteAuthor(@PathVariable Long id) {

authors.removeIf(author -> author.getId().equals(id));

return "Author with ID " + id + " deleted.";

}

}

CustomerController.java

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Customer;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/customers")

public class CustomerController {

private List<Customer> customers = new ArrayList<>();

@GetMapping

public List<Customer> getAllCustomers() {

return customers;

}

@GetMapping("/{id}")

public Customer getCustomerById(@PathVariable Long id) {

return customers.stream()

.filter(customer -> customer.getId().equals(id))

.findFirst()

.orElse(null);

}

@PostMapping

public Customer createCustomer(@RequestBody Customer customer) {

customers.add(customer);

return customer;

}

@PutMapping("/{id}")

public Customer updateCustomer(@PathVariable Long id, @RequestBody Customer updatedCustomer) {

customers.stream()

.filter(customer -> customer.getId().equals(id))

.forEach(customer -> {

customer.setName(updatedCustomer.getName());

customer.setEmail(updatedCustomer.getEmail());

});

return updatedCustomer;

}

@DeleteMapping("/{id}")

public String deleteCustomer(@PathVariable Long id) {

customers.removeIf(customer -> customer.getId().equals(id));

return "Customer with ID " + id + " deleted.";

}

}

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

**Business Scenario:**

Implement RESTful endpoints to manage books.

BookController:

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Book;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> books = new ArrayList<>();

// Sample data for demonstration

public BookController() {

books.add(new Book(1L, "Book One", "Author One", 9.99, "1234567890"));

books.add(new Book(2L, "Book Two", "Author Two", 14.99, "0987654321"));

}

// GET /books - Get all books

@GetMapping

public List<Book> getAllBooks() {

return books;

}

// GET /books/{id} - Get a book by ID

@GetMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(ResponseEntity::ok)

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

}

// POST /books - Add a new book

@PostMapping

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

books.add(book);

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

}

// PUT /books/{id} - Update a book

@PutMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

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

})

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

}

// DELETE /books/{id} - Delete a book

@DeleteMapping("/{id}")

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

boolean removed = books.removeIf(book -> book.getId().equals(id));

if (removed) {

return new ResponseEntity<>("Book with ID " + id + " deleted.", HttpStatus.OK);

} else {

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

}

}

}

Implement methods to handle **GET**, **POST**, **PUT**, and **DELETE** requests.

GET:

@GetMapping

public List<Book> getAllBooks() {

return books;

}

@GetMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(ResponseEntity::ok)

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

}

JSON RESPONSE:

{

"id": 1,

"title": "Book One",

"author": "Author One",

"price": 9.99,

"isbn": "1234567890"

}

POST

@PostMapping

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

books.add(book);

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

}

JSON RESPONSE:

HTTP/1.1 201 Created

Content-Type: application/json

{

"id": 3,

"title": "Book Three",

"author": "Author Three",

"price": 19.99,

"isbn": "1122334455"

}

PUT:

@PutMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

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

})

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

}

JSON RESPONSE:

POST /books

Content-Type: application/json

{

"id": 3,

"title": "Book Three",

"author": "Author Three",

"price": 19.99,

"isbn": "1122334455"

}

DELETE:

@DeleteMapping("/{id}")

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

boolean removed = books.removeIf(book -> book.getId().equals(id));

if (removed) {

return new ResponseEntity<>("Book with ID " + id + " deleted.", HttpStatus.OK);

} else {

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

}

}

JSON RESPONSE:

HTTP/1.1 200 OK

Content-Type: application/json

"Book with ID 1 deleted."

FINAL BOOKCONTROLLER CLASS:

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Book;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> books = new ArrayList<>();

// Sample data for demonstration

public BookController() {

books.add(new Book(1L, "Book One", "Author One", 9.99, "1234567890"));

books.add(new Book(2L, "Book Two", "Author Two", 14.99, "0987654321"));

}

// GET /books - Get all books

@GetMapping

public List<Book> getAllBooks() {

return books;

}

// GET /books/{id} - Get a book by ID

@GetMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(ResponseEntity::ok)

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

}

// POST /books - Add a new book

@PostMapping

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

books.add(book);

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

}

// PUT /books/{id} - Update a book

@PutMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

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

})

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

}

// DELETE /books/{id} - Delete a book

@DeleteMapping("/{id}")

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

boolean removed = books.removeIf(book -> book.getId().equals(id));

if (removed) {

return new ResponseEntity<>("Book with ID " + id + " deleted.", HttpStatus.OK);

} else {

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

}

}

}

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

**Business Scenario:**

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

Fetching a Book by ID:

@GetMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(ResponseEntity::ok)

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

}

Eg:

GET /books/1

RESPONSE:

{

"id": 1,

"title": "Book One",

"author": "Author One",

"price": 9.99,

"isbn": "1234567890"

}

Filtering Books by Title and/or Author:

In BookController:

@GetMapping("/search")

public List<Book> searchBooks(

@RequestParam(required = false) String title,

@RequestParam(required = false) String author) {

return books.stream()

.filter(book -> (title == null || book.getTitle().equalsIgnoreCase(title)) &&

(author == null || book.getAuthor().equalsIgnoreCase(author)))

.toList();

}

Eg:

GET /books/search?title=Book One

RESPONSE:

[

{

"id": 1,

"title": "Book One",

"author": "Author One",

"price": 9.99,

"isbn": "1234567890"

}

]

BookController class:

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Book;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> books = new ArrayList<>();

// Sample data for demonstration

public BookController() {

books.add(new Book(1L, "Book One", "Author One", 9.99, "1234567890"));

books.add(new Book(2L, "Book Two", "Author Two", 14.99, "0987654321"));

}

// GET /books - Get all books

@GetMapping

public List<Book> getAllBooks() {

return books;

}

// GET /books/{id} - Get a book by ID

@GetMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(ResponseEntity::ok)

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

}

// GET /books/search - Search books by title and/or author

@GetMapping("/search")

public List<Book> searchBooks(

@RequestParam(required = false) String title,

@RequestParam(required = false) String author) {

return books.stream()

.filter(book -> (title == null || book.getTitle().equalsIgnoreCase(title)) &&

(author == null || book.getAuthor().equalsIgnoreCase(author)))

.toList();

}

// POST /books - Add a new book

@PostMapping

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

books.add(book);

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

}

// PUT /books/{id} - Update a book

@PutMapping("/{id}")

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

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

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

})

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

}

// DELETE /books/{id} - Delete a book

@DeleteMapping("/{id}")

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

boolean removed = books.removeIf(book -> book.getId().equals(id));

if (removed) {

return new ResponseEntity<>("Book with ID " + id + " deleted.", HttpStatus.OK);

} else {

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

}

}

}

**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.

Customer Entity:

package com.yourname.bookstoreapi.model;

public class Customer {

private Long id;

private String name;

private String email;

private String address;

// Constructors, getters, and setters

public Customer(Long id, String name, String email, String address) {

this.id = id;

this.name = name;

this.email = email;

this.address = address;

}

// Getters and setters...

}

POST Endpoint to Create a Customer (JSON Request Body)

@PostMapping("/customers")

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

customers.add(customer);

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

}

POST Endpoint to Register Customer (Form Data):

@PostMapping("/customers/register")

public ResponseEntity<Customer> registerCustomer(

@RequestParam("name") String name,

@RequestParam("email") String email,

@RequestParam("address") String address) {

Customer customer = new Customer(null, name, email, address);

customers.add(customer);

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

}

Eg:

POST /customers/register

Content-Type: application/x-www-form-urlencoded

name=John Doe&email=johndoe@example.com&address=123 Main St

Response:

{

"id": null,

"name": "John Doe",

"email": "johndoe@example.com",

"address": "123 Main St"

}

CustomerController class:

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Customer;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/customers")

public class CustomerController {

private List<Customer> customers = new ArrayList<>();

// POST /customers - Create a new customer (JSON Request Body)

@PostMapping

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

customers.add(customer);

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

}

// POST /customers/register - Register a customer (Form Data)

@PostMapping("/register")

public ResponseEntity<Customer> registerCustomer(

@RequestParam("name") String name,

@RequestParam("email") String email,

@RequestParam("address") String address) {

Customer customer = new Customer(null, name, email, address);

customers.add(customer);

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

}

}