**WEEK4 EXERCISE**

**EXERCISE 1: Online Booktore-Setting Up RESTFUL Services**

**Main Application:**

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);

}

}

**Model Creation:**

package com.example.bookstoreapi.model;

import lombok.Data;

@Data

public class Book {

private Long id;

private String title;

private String author;

private String isbn;

private Double price;

}

**Controller Creation:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

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

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

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

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

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

private final List<Book> bookList = new ArrayList<>();

@GetMapping

public List<Book> getAllBooks() {

return bookList;

}

@PostMapping

public Book addBook(@RequestBody Book book) {

book.setId((long) (bookList.size() + 1));

bookList.add(book);

return book;

}

**Project Structure:**

- Once the project is generated, familiarize yourself with the following folders:

src/main/java: This is where you’ll write the main application code. The base package should contain the BookstoreAPIApplication.java file which is the entry point of your application.

src/main/resources: This is where configuration files, such as application.properties, are located. It also contains static resources, templates, and other assets.

src/test/java: Contains the unit test cases for your application.

**What's New in Spring Boot 3:**

Spring Boot 3 introduced several new features and improvements:

Java 17 Support: Spring Boot 3 requires Java 17, which includes language enhancements like pattern matching for switch, records, and sealed classes.

GraalVM Native Image Support: Spring Boot 3 provides better support for compiling Java applications into native executables using GraalVM, resulting in faster startup times and reduced memory footprint.

Micrometer Observability: Enhanced observability with the new Micrometer Observability project, which offers out-of-the-box metrics and tracing capabilities.

Jakarta EE 9: Transition to the Jakarta namespace (javax to jakarta), aligning with the changes introduced in Jakarta EE 9 and beyond.

Enhanced Security: New security features and updates, including support for OpenID Connect 1.0, OAuth 2.1, and integration with Spring Authorization Server.

Improved HTTP Interface: Introduced a new way of defining HTTP clients using HTTP interfaces, allowing for declarative client definitions similar to Spring Data repositories.

**Exercise 2 : Creating Basic Rest Controllers**

**Book Entity:**

package com.example.bookstoreapi.model;

import lombok.Data;

@Data

public class Book {

private Long id;

private String title;

private String author;

private Double price;

private String isbn;

}

**Bookstore Api Controller:**

package com.example.bookstoreapi.controller;

import com.example.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;

import java.util.Optional;

@RestController

@RequestMapping("/api/books")

public class BookController {

private final List<Book> bookList = new ArrayList<>();

@GetMapping

public List<Book> getAllBooks() {

return bookList;

}

@GetMapping("/{id}")

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

Optional<Book> book = bookList.stream().filter(b -> b.getId().equals(id)).findFirst();

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());

}

@PostMapping

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

book.setId((long) (bookList.size() + 1));

bookList.add(book);

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

}

@PutMapping("/{id}")

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

Optional<Book> existingBookOpt = bookList.stream().filter(b -> b.getId().equals(id)).findFirst();

if (existingBookOpt.isPresent()) {

Book existingBook = existingBookOpt.get();

existingBook.setTitle(updatedBook.getTitle());

existingBook.setAuthor(updatedBook.getAuthor());

existingBook.setPrice(updatedBook.getPrice());

existingBook.setIsbn(updatedBook.getIsbn());

return ResponseEntity.ok(existingBook);

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

@DeleteMapping("/{id}")

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

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

if (removed) {

return ResponseEntity.noContent().build();

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

}

**JSON Responses:**

->The ‘@RestController’ annotation ensures that the controller methods automatically serialize the return values to JSON format

->The ‘ResponseEntity’ class is used to return HTTP ststus codes along with the JSON responses.

**Exercise 3: Handling Path Variables and Query Parameters**

**Handling Path Variables:**

// GET: Retrieve a book by ID using a path variable

@GetMapping("/{id}")

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

Optional<Book> book = bookList.stream().filter(b -> b.getId().equals(id)).findFirst();

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());

}

**Handling Query Parameters:**

// GET: Retrieve all books, with optional filtering by title and/or author

@GetMapping

public List<Book> getAllBooks(@RequestParam(required = false) String title,

@RequestParam(required = false) String author) {

return bookList.stream()

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

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

.toList();

}

Book Controller with Path Variables and query Parameters:

package com.example.bookstoreapi.controller;

import com.example.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;

import java.util.Optional;

@RestController

@RequestMapping("/api/books")

public class BookController {

private final List<Book> bookList = new ArrayList<>();

// GET: Retrieve all books, with optional filtering by title and/or author

@GetMapping

public List<Book> getAllBooks(@RequestParam(required = false) String title,

@RequestParam(required = false) String author) {

return bookList.stream()

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

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

.toList();

}

// GET: Retrieve a book by ID using a path variable

@GetMapping("/{id}")

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

Optional<Book> book = bookList.stream().filter(b -> b.getId().equals(id)).findFirst();

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());

}

// POST: Add a new book

@PostMapping

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

book.setId((long) (bookList.size() + 1));

bookList.add(book);

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

}

// PUT: Update an existing book

@PutMapping("/{id}")

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

Optional<Book> existingBookOpt = bookList.stream().filter(b -> b.getId().equals(id)).findFirst();

if (existingBookOpt.isPresent()) {

Book existingBook = existingBookOpt.get();

existingBook.setTitle(updatedBook.getTitle());

existingBook.setAuthor(updatedBook.getAuthor());

existingBook.setPrice(updatedBook.getPrice());

existingBook.setIsbn(updatedBook.getIsbn());

return ResponseEntity.ok(existingBook);

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

// DELETE: Remove a book by ID

@DeleteMapping("/{id}")

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

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

if (removed) {

return ResponseEntity.noContent().build();

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

}

**Exercise 4:Processing Request Body and Form Data**

**Customer Entity:**

package com.example.bookstoreapi.model;

import lombok.Data;

@Data

public class Customer {

private Long id;

private String name;

private String email;

private String address;

}

**Customer Controller:**  
package com.example.bookstoreapi.controller;

import com.example.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("/api/customers")

public class CustomerController {

private final List<Customer> customerList = new ArrayList<>();

// POST: Create a new customer by accepting a JSON request body

@PostMapping

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

customer.setId((long) (customerList.size() + 1));

customerList.add(customer);

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

}

// POST: Process form data for customer registration

@PostMapping("/register")

public ResponseEntity<Customer> registerCustomer(

@RequestParam String name,

@RequestParam String email,

@RequestParam String address) {

Customer customer = new Customer();

customer.setId((long) (customerList.size() + 1));

customer.setName(name);

customer.setEmail(email);

customer.setAddress(address);

customerList.add(customer);

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

}

// GET: Retrieve all customers (for testing purposes)

@GetMapping

public List<Customer> getAllCustomers() {

return customerList;

}

}

**Exercise 5: Customizing Response Status and Handling:**

**Customizing HTTP Response**

import org.springframework.http.HttpStatus;

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

@RestController

@RequestMapping("/api/books")

public class BookController {

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public Book createBook(@RequestBody Book newBook) {

// Logic to save the new book

return savedBook;

}

}

**Adding Custom Headers with ‘Response Entity’:**

import org.springframework.http.HttpStatus;

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

@RestController

@RequestMapping("/api/books")

public class BookController {

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public Book createBook(@RequestBody Book newBook) {

return savedBook;

}

}

**Combining:**

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/api/books")

public class BookController {

@GetMapping("/{id}")

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

Book book = bookService.findById(id);

if (book == null) {

return ResponseEntity.notFound().build();

}

HttpHeaders headers = new HttpHeaders();

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

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

}

}

**Exercise 6: Exception Handling in Rest Controllers**

**GlobalExceptionHandler:**

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.context.request.WebRequest;

@ControllerAdvice

public class GlobalExceptionHandler

@ExceptionHandler(BookNotFoundException.class)

public ResponseEntity<String> handleBookNotFoundException(BookNotFoundException ex, WebRequest request) {

return new ResponseEntity<>("Book not found: " + ex.getMessage(), HttpStatus.NOT\_FOUND);

}

@ExceptionHandler(IllegalArgumentException.class)

public ResponseEntity<String> handleIllegalArgumentException(IllegalArgumentException ex, WebRequest request) {

return new ResponseEntity<>("Invalid argument: " + ex.getMessage(), HttpStatus.BAD\_REQUEST);

}

@ExceptionHandler(Exception.class)

public ResponseEntity<String> handleGlobalException(Exception ex, WebRequest request) {

return new ResponseEntity<>("An error occurred: " + ex.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

**Custom Exception Classes:**

public class BookNotFoundException extends RuntimeException {

public BookNotFoundException(String message) {

super(message);

}

}

**Custom Exceptions in Controllers:**

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

@RestController

@RequestMapping("/api/books")

public class BookController {

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

Book book = bookService.findById(id);

if (book == null) {

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

}

return book;

}

}

**Handling Other Exceptions:**

import org.springframework.validation.BindException;

import org.springframework.web.bind.MethodArgumentNotValidException;

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<String> handleValidationExceptions(MethodArgumentNotValidException ex) {

return new ResponseEntity<>("Validation failed: " + ex.getBindingResult().toString(), HttpStatus.BAD\_REQUEST);

}