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

**CODE:**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

</dependencies>

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

**CODE:**

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> books = new ArrayList<>(); // Temporary in-memory list to simulate a database

// GET: Retrieve all books

@GetMapping

public List<Book> getAllBooks() {

return books;

}

// GET: Retrieve a book by ID

@GetMapping("/{id}")

public Book getBookById(@PathVariable int id) {

return books.stream().filter(book -> book.getId() == id).findFirst().orElse(null);

}

// POST: Add a new book

@PostMapping

public Book addBook(@RequestBody Book book) {

books.add(book);

return book;

}

// PUT: Update an existing book

@PutMapping("/{id}")

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

Book book = books.stream().filter(b -> b.getId() == id).findFirst().orElse(null);

if (book != null) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

}

return book;

}

// DELETE: Remove a book by ID

@DeleteMapping("/{id}")

public String deleteBook(@PathVariable int id) {

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

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

}

}

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

**CODE:**

package com.bookstoreapi.controller;

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

import com.bookstoreapi.model.Book;

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

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

// GET: Retrieve all books

@GetMapping

public List<Book> getAllBooks() {

return books;

}

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

@GetMapping("/{id}")

public Book getBookById(@PathVariable int id) {

return books.stream().filter(book -> book.getId() == id).findFirst().orElse(null);

}

// POST: Add a new book

@PostMapping

public Book addBook(@RequestBody Book book) {

books.add(book);

return book;

}

// PUT: Update an existing book

@PutMapping("/{id}")

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

Book book = books.stream().filter(b -> b.getId() == id).findFirst().orElse(null);

if (book != null) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

}

return book;

}

// DELETE: Remove a book by ID

@DeleteMapping("/{id}")

public String deleteBook(@PathVariable int id) {

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

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

}

// GET: Search books by title and/or author using query parameters

@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();

}

}

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

**CODE:**

package com.bookstoreapi.controller;

import com.bookstoreapi.model.Customer;

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

@RestController

@RequestMapping("/customers")

public class CustomerController {

// POST: Create a new customer from JSON request body

@PostMapping

public Customer createCustomer(@RequestBody Customer customer) {

// Logic to save customer (for now, just return the customer object)

return customer;

}

// POST: Process form data for customer registration

@PostMapping("/register")

public Customer registerCustomer(

@RequestParam("name") String name,

@RequestParam("email") String email,

@RequestParam("password") String password) {

// Create and return customer object from form data

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customer.setPassword(password);

return customer;

}

}

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

**CODE:**

package com.bookstoreapi.controller;

import com.bookstoreapi.model.Book;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/books")

public class BookController {

// POST: Create a new book with custom status

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public Book createBook(@RequestBody Book book) {

// Logic to save book (for now, just return the book object)

return book;

}

// GET: Retrieve a book with custom headers

@GetMapping("/{id}")

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

Book book = new Book(); // Example book retrieval logic

HttpHeaders headers = new HttpHeaders();

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

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

}

}

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

**CODE:**

package com.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 exceptions

@ExceptionHandler(ResourceNotFoundException.class)

@ResponseStatus(HttpStatus.NOT\_FOUND)

public ResponseEntity<String> handleResourceNotFound(ResourceNotFoundException ex) {

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

}

// Handle generic exceptions

@ExceptionHandler(Exception.class)

@ResponseStatus(HttpStatus.INTERNAL\_SERVER\_ERROR)

public ResponseEntity<String> handleGenericException(Exception ex) {

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

}

}

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

**CODE:**

package com.bookstoreapi.dto;

// 1. BookDTO and CustomerDTO classes

public class BookDTO {

private int id;

private String title;

private String author;

private double price;

// Getters and Setters

}

public class CustomerDTO {

private String name;

private String email;

// Custom serialization for hiding sensitive data

@JsonProperty(access = JsonProperty.Access.WRITE\_ONLY)

private String password;

// Getters and Setters

}

// 2. Entity-to-DTO mapping using MapStruct

@Mapper(componentModel = "spring")

public interface BookMapper {

BookDTO toBookDTO(Book book);

Book toBook(BookDTO bookDTO);

}

@Mapper(componentModel = "spring")

public interface CustomerMapper {

CustomerDTO toCustomerDTO(Customer customer);

Customer toCustomer(CustomerDTO customerDTO);

}

// 3. Example usage in a controller

@RestController

@RequestMapping("/books")

public class BookController {

private final BookMapper bookMapper;

public BookController(BookMapper bookMapper) {

this.bookMapper = bookMapper;

}

@GetMapping("/{id}")

public BookDTO getBook(@PathVariable int id) {

Book book = new Book(); // Retrieve book entity

return bookMapper.toBookDTO(book); // Map entity to DTO

}

@PostMapping

public BookDTO createBook(@RequestBody BookDTO bookDTO) {

Book book = bookMapper.toBook(bookDTO); // Map DTO to entity

// Save book logic

return bookMapper.toBookDTO(book); // Return mapped DTO

}

}