**Exercise 1:**

**Online Bookstore - Setting Up RESTful Services**

**1. Setup Spring Boot Project:**

A new spring project was named BookStoreAPI.

curl https://start.spring.io/starter.zip \

-d dependencies=web,devtools,lombok \

-d name=BookstoreAPI \

-d packageName=com.example.bookstoreapi \

-d bootVersion=3.0.0 \

-d javaVersion=17 \

-d type=maven-project \

-o bookstoreapi.zip

Unzipping and navigating the project: unzip bookstoreapi.zip

cd bookstoreapi

./mvnw spring-boot:run

**2. Project Structure:**

Project Structure was analysed.

**3. New features in spring boot 3:**

**Jakarta EE Migration:** Spring Boot 3 has transitioned to Jakarta EE from Java EE, leading to changes in package names (e.g., javax to jakarta).

**GraalVM Native Image Support**: This allows you to compile your application into a native executable, improving startup time and memory usage.

**Enhanced Docker Support**: Improved Docker image creation using Spring Boot’s build-image goal in the Maven plugin.

**Exercise 2:**

**Online Bookstore - Creating Basic REST Controllers**

**1. Create Book Controller class:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.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<>();

public BookController() {

books.add(new Book(1, "Effective Java", "Joshua Bloch", 45.0, "978-0134685991"));

books.add(new Book(2, "Clean Code", "Robert C. Martin", 50.0, "978-0132350884"));

}

}

**2. Handle HTTP Methods:**

Implementing methods for GET, POST, PUT, and DELETE requests.

@GetMapping

public List<Book> getAllBooks() {

return books;

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable int id) {

return books.stream()

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

.findFirst()

.orElse(null); // Return null or handle not found

}

@PostMapping

public Book addBook(@RequestBody Book newBook) {

books.add(newBook);

return newBook;

}

@PutMapping("/{id}")

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

books.stream()

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

.forEach(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

});

return updatedBook;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable int id) {

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

}

}

**3. Return JSON Responses:**

package com.example.bookstoreapi.entity;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Book {

private int id;

private String title;

private String author;

private double price;

private String isbn;

}

**Exercise 3:**

**Online Bookstore - Handling Path Variables and Query Parameters**

**1. Path Variables:**

**Fetching a book by ID**

@GetMapping("/{id}")

public Book getBookById(@PathVariable int id) {

return books.stream()

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

.findFirst()

.orElse(null); // Return null or handle not found

}

**2. Query Parameters:**

**Fetching books based on 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**

**Request Body - Creating a New Customer:**

**Define the Customer entity:**

package com.example.bookstoreapi.entity;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Customer {

private int id;

private String name;

private String email;

private String address;

}

**Implement the POST endpoint:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.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<>();

@PostMapping

public Customer createCustomer(@RequestBody Customer newCustomer) {

customers.add(newCustomer);

return newCustomer;

}

@GetMapping

public List<Customer> getAllCustomers() {

return customers;

}

}

**Form Data - Customer Registration:**

@PostMapping("/register")

public Customer registerCustomer(@RequestParam String name,

@RequestParam String email,

@RequestParam String address) {

Customer newCustomer = new Customer(customers.size() + 1, name, email, address);

customers.add(newCustomer);

return newCustomer;

}

**Exercise 5:**

**Online Bookstore - Customizing Response Status and Headers**

**Customizing Response Status:**

import org.springframework.http.HttpStatus;

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

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public Book addBook(@RequestBody Book newBook) {

books.add(newBook);

return newBook;

}

**Adding Custom Headers using ResponseEntity:**

import org.springframework.http.ResponseEntity;

import org.springframework.http.HttpHeaders;

@PostMapping

public ResponseEntity<Book> addBookWithCustomHeader(@RequestBody Book newBook) {

books.add(newBook);

HttpHeaders headers = new HttpHeaders();

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

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

}

**Exercise 6:**

**Online Bookstore - Exception Handling in REST Controllers**

**Creating GlobalExceptionHandler class:**

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.ResponseBody;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

@ResponseBody

public ResponseEntity<ErrorResponse> handleResourceNotFoundException(ResourceNotFoundException ex) {

ErrorResponse errorResponse = new ErrorResponse("Resource not found", ex.getMessage());

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

}

@ExceptionHandler(InvalidInputException.class)

@ResponseBody

public ResponseEntity<ErrorResponse> handleInvalidInputException(InvalidInputException ex) {

ErrorResponse errorResponse = new ErrorResponse("Invalid input", ex.getMessage());

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

}

@ExceptionHandler(Exception.class)

@ResponseBody

public ResponseEntity<ErrorResponse> handleGenericException(Exception ex) {

ErrorResponse errorResponse = new ErrorResponse("Internal server error", "An unexpected error occurred");

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

}

}

**Defining ResourceNotFoundException and InvalidInputException:**

package com.example.bookstoreapi.exception;

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

public class InvalidInputException extends RuntimeException {

public InvalidInputException(String message) {

super(message);

}

}

**Defining ErrorResponse:**

package com.example.bookstoreapi.exception;

public class ErrorResponse {

private String error;

private String message;

public ErrorResponse(String error, String message) {

this.error = error;

this.message = message;

}

public String getError() {

return error;

}

public void setError(String error) {

this.error = error;

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

}

**Exercise 7:**

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

**Creating BookDTO:**

package com.example.bookstoreapi.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class BookDTO {

private int id;

private String title;

private String author;

private double price;

private String isbn;

}

**Creating CustomerDTO classes:**

package com.example.bookstoreapi.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class CustomerDTO {

private int id;

private String name;

private String email;

private String address;

}

**Add ModelMapper dependency to pom.xml:**

<dependency>

<groupId>org.modelmapper</groupId>

<artifactId>modelmapper</artifactId>

<version>3.1.1</version>

</dependency>

**Create a configuration class for ModelMapper:**

package com.example.bookstoreapi.config;

import org.modelmapper.ModelMapper;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

@Bean

public ModelMapper modelMapper() {

return new ModelMapper();

}

}

**Using ModelMapper in your controllers to map entities and DTOs:**

package com.example.bookstoreapi.service;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.entity.Book;

import org.modelmapper.ModelMapper;

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

import org.springframework.stereotype.Service;

@Service

public class BookService {

@Autowired

private ModelMapper modelMapper;

public BookDTO convertToDTO(Book book) {

return modelMapper.map(book, BookDTO.class);

}

public Book convertToEntity(BookDTO bookDTO) {

return modelMapper.map(bookDTO, Book.class);

}}

**Custom Serialization/Deserialization with Jackson Annotations:**

package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class BookDTO {

private int id;

@JsonProperty("book\_title")

private String title;

private String author;

private double price;

@JsonIgnore

private String isbn;

}

**Exercise 8:**

**Online Bookstore - Implementing CRUD Operations**

**Defining Book Entity:**

package com.example.bookstoreapi.entity;

import lombok.Data;

import javax.persistence.\*;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

import javax.validation.constraints.Min;

@Data

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 255)

private String title;

@NotNull

@Size(min = 1, max = 255)

private String author;

@Min(0)

private double price;

@Size(min = 1, max = 13)

private String isbn;

@Version

private Integer version; // Optimistic locking field

}

**Defining Customer Entity:**

package com.example.bookstoreapi.entity;

import lombok.Data;

import javax.persistence.\*;

import javax.validation.constraints.Email;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

@Data

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 255)

private String name;

@NotNull

@Email

private String email;

@Size(max = 500)

private String address;

@Version

private Integer version; // Optimistic locking field

}

**Defining Book Repository:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.entity.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {}

**Defining Customer Repository:**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.entity.Customer;

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

public interface CustomerRepository extends JpaRepository<Customer, Long> {}

**Implement CRUD Endpoints:**

**Book Controller:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

import com.example.bookstoreapi.service.BookService;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

@Validated

public class BookController {

@Autowired

private BookService bookService;

@PostMapping

public ResponseEntity<BookDTO> createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = bookService.convertToEntity(bookDTO);

book = bookService.saveBook(book);

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

}

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return ResponseEntity.ok(bookService.convertToDTO(book));

}

@PutMapping("/{id}")

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

Book book = bookService.updateBook(id, bookDTO);

return ResponseEntity.ok(bookService.convertToDTO(book));

}

@DeleteMapping("/{id}")

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void deleteBook(@PathVariable Long id) {

bookService.deleteBook(id);

}

@GetMapping

public List<BookDTO> getAllBooks() {

return bookService.getAllBooks().stream()

.map(bookService::convertToDTO)

.collect(Collectors.toList());

}

}

**Customer Controller:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.entity.Customer;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

import com.example.bookstoreapi.service.CustomerService;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/customers")

@Validated

public class CustomerController {

@Autowired

private CustomerService customerService;

@PostMapping

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

Customer customer = customerService.convertToEntity(customerDTO);

customer = customerService.saveCustomer(customer);

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

}

@GetMapping("/{id}")

public ResponseEntity<CustomerDTO> getCustomerById(@PathVariable Long id) {

Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customerService.convertToDTO(customer));

}

@PutMapping("/{id}")

public ResponseEntity<CustomerDTO> updateCustomer(@PathVariable Long id, @Valid @RequestBody CustomerDTO customerDTO) {

Customer customer = customerService.updateCustomer(id, customerDTO);

return ResponseEntity.ok(customerService.convertToDTO(customer));

}

@DeleteMapping("/{id}")

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void deleteCustomer(@PathVariable Long id) {

customerService.deleteCustomer(id);

}

@GetMapping

public List<CustomerDTO> getAllCustomers() {

return customerService.getAllCustomers().stream()

.map(customerService::convertToDTO)

.collect(Collectors.toList());

}

}

**Implementing Book Services:**

package com.example.bookstoreapi.service;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

import com.example.bookstoreapi.repository.BookRepository;

import org.modelmapper.ModelMapper;

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

import org.springframework.stereotype.Service;

import javax.persistence.EntityNotFoundException;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

@Autowired

private ModelMapper modelMapper;

public BookDTO convertToDTO(Book book) {

return modelMapper.map(book, BookDTO.class);

}

public Book convertToEntity(BookDTO bookDTO) {

return modelMapper.map(bookDTO, Book.class);

}

public Book saveBook(Book book) {

return bookRepository.save(book);

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundException("Book with ID " + id + " not found"));

}

public Book updateBook(Long id, BookDTO bookDTO) {

Book book = getBookById(id);

modelMapper.map(bookDTO, book);

return bookRepository.save(book);

}

public void deleteBook(Long id) {

Book book = getBookById(id);

bookRepository.delete(book);

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

}

**Implementing Customer Services:**

package com.example.bookstoreapi.service;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.entity.Customer;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

import com.example.bookstoreapi.repository.CustomerRepository;

import org.modelmapper.ModelMapper;

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

import org.springframework.stereotype.Service;

import javax.persistence.EntityNotFoundException;

@Service

public class CustomerService {

@Autowired

private CustomerRepository customerRepository;

@Autowired

private ModelMapper modelMapper;

public CustomerDTO convertToDTO(Customer customer) {

return modelMapper.map(customer, CustomerDTO.class);

}

public Customer convertToEntity(CustomerDTO customerDTO) {

return modelMapper.map(customerDTO, Customer.class);

}

public Customer saveCustomer(Customer customer) {

return customerRepository.save(customer);

}

public Customer getCustomerById(Long id) {

return customerRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundException("Customer with ID " + id + " not found"));

}

public Customer updateCustomer(Long id, CustomerDTO customerDTO) {

Customer customer = getCustomerById(id);

modelMapper.map(customerDTO, customer);

return customerRepository.save(customer);

}

public void deleteCustomer(Long id) {

Customer customer = getCustomerById(id);

customerRepository.delete(customer);

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

}

**Exercise 9:**

**Online Bookstore - Understanding HATEOAS**

**Adding Spring HATEOAS Dependency:**

<dependency>

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

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

</dependency>

**Creating Book Resource Assembler:**

package com.example.bookstoreapi.assembler;

import com.example.bookstoreapi.controller.BookController;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.dto.BookDTO;

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.Link;

import org.springframework.hateoas.server.mvc.WebMvcLinkBuilder;

import org.springframework.stereotype.Component;

@Component

public class BookModelAssembler {

public EntityModel<BookDTO> toModel(BookDTO bookDTO) {

EntityModel<BookDTO> bookModel = EntityModel.of(bookDTO);

Link selfLink = WebMvcLinkBuilder.linkTo(

WebMvcLinkBuilder.methodOn(BookController.class).getBookById(bookDTO.getId())

).withSelfRel();

Link allBooksLink = WebMvcLinkBuilder.linkTo(

WebMvcLinkBuilder.methodOn(BookController.class).getAllBooks()

).withRel("all-books");

bookModel.add(selfLink, allBooksLink);

return bookModel;

}

}

**Creating Customer Resource Assembler:**

package com.example.bookstoreapi.assembler;

import com.example.bookstoreapi.controller.CustomerController;

import com.example.bookstoreapi.entity.Customer;

import com.example.bookstoreapi.dto.CustomerDTO;

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.Link;

import org.springframework.hateoas.server.mvc.WebMvcLinkBuilder;

import org.springframework.stereotype.Component;

@Component

public class CustomerModelAssembler {

public EntityModel<CustomerDTO> toModel(CustomerDTO customerDTO) {

EntityModel<CustomerDTO> customerModel = EntityModel.of(customerDTO);

Link selfLink = WebMvcLinkBuilder.linkTo(

WebMvcLinkBuilder.methodOn(CustomerController.class).getCustomerById(customerDTO.getId())

).withSelfRel();

Link allCustomersLink = WebMvcLinkBuilder.linkTo(

WebMvcLinkBuilder.methodOn(CustomerController.class).getAllCustomers()

).withRel("all-customers");

customerModel.add(selfLink, allCustomersLink);

return customerModel;

}

}

**Modifying Book Controller:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

import com.example.bookstoreapi.assembler.BookModelAssembler;

import com.example.bookstoreapi.service.BookService;

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

import org.springframework.hateoas.EntityModel;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

@Validated

public class BookController {

@Autowired

private BookService bookService;

@Autowired

private BookModelAssembler assembler;

@PostMapping

public ResponseEntity<EntityModel<BookDTO>> createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = bookService.convertToEntity(bookDTO);

book = bookService.saveBook(book);

return new ResponseEntity<>(assembler.toModel(bookService.convertToDTO(book)), HttpStatus.CREATED);

}

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return ResponseEntity.ok(assembler.toModel(bookService.convertToDTO(book)));

}

@PutMapping("/{id}")

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

Book book = bookService.updateBook(id, bookDTO);

return ResponseEntity.ok(assembler.toModel(bookService.convertToDTO(book)));

}

@DeleteMapping("/{id}")

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void deleteBook(@PathVariable Long id) {

bookService.deleteBook(id);

}

@GetMapping

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

return bookService.getAllBooks().stream()

.map(bookService::convertToDTO)

.map(assembler::toModel)

.collect(Collectors.toList());

}

}

**Modifying Customer Controller:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.entity.Customer;

import com.example.bookstoreapi.exception.ResourceNotFoundException;

import com.example.bookstoreapi.assembler.CustomerModelAssembler;

import com.example.bookstoreapi.service.CustomerService;

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

import org.springframework.hateoas.EntityModel;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/customers")

@Validated

public class CustomerController {

@Autowired

private CustomerService customerService;

@Autowired

private CustomerModelAssembler assembler;

@PostMapping

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

Customer customer = customerService.convertToEntity(customerDTO);

customer = customerService.saveCustomer(customer);

return new ResponseEntity<>(assembler.toModel(customerService.convertToDTO(customer)), HttpStatus.CREATED);

}

@GetMapping("/{id}")

public ResponseEntity<EntityModel<CustomerDTO>> getCustomerById(@PathVariable Long id) {

Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(assembler.toModel(customerService.convertToDTO(customer)));

}

@PutMapping("/{id}")

public ResponseEntity<EntityModel<CustomerDTO>> updateCustomer(@PathVariable Long id, @Valid @RequestBody CustomerDTO customerDTO) {

Customer customer = customerService.updateCustomer(id, customerDTO);

return ResponseEntity.ok(assembler.toModel(customerService.convertToDTO(customer)));

}

@DeleteMapping("/{id}")

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void deleteCustomer(@PathVariable Long id) {

customerService.deleteCustomer(id);

}

@GetMapping

public List<EntityModel<CustomerDTO>> getAllCustomers() {

return customerService.getAllCustomers().stream()

.map(customerService::convertToDTO)

.map(assembler::toModel)

.collect(Collectors.toList());

}

}

**Exercise 10:**

**Online Bookstore - Configuring Content Negotiation**

**Adding Dependencies:**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

**Configuring Content Negotiation:**

package com.example.bookstoreapi.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.ContentNegotiationConfigurer;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {

configurer.favorPathExtension(false)

.favorParameter(true)

.parameterName("mediaType")

.ignoreAcceptHeader(false)

.defaultContentType(org.springframework.http.MediaType.APPLICATION\_JSON)

.mediaType("json", org.springframework.http.MediaType.APPLICATION\_JSON)

.mediaType("xml", org.springframework.http.MediaType.APPLICATION\_XML);

}

}

**Implement Book Controllers with Content Negotiation:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.service.BookService;

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

import org.springframework.http.MediaType;

import org.springframework.hateoas.EntityModel;

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@Autowired

private BookModelAssembler assembler;

@PostMapping(produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public EntityModel<Book> createBook(@Valid @RequestBody Book book) {

Book savedBook = bookService.saveBook(book);

return assembler.toModel(savedBook);

}

@GetMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

Book book = bookService.getBookById(id);

return assembler.toModel(book);

}

@GetMapping(produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

return bookService.getAllBooks().stream()

.map(assembler::toModel)

.collect(Collectors.toList());

}

@PutMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

Book updatedBook = bookService.updateBook(id, book);

return assembler.toModel(updatedBook);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookService.deleteBook(id);

}

}

**Exercise 11:**

**Online Bookstore - Integrating Spring Boot Actuator**

**Adding Actuator Dependency:**

<dependency>

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

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

</dependency>

**Exposing Actuator Endpoints Using application.properties:**

# Enable all Actuator endpoints

management.endpoints.web.exposure.include=\*

# Configure specific endpoint settings

management.endpoint.health.show-details=always

management.endpoints.web.exposure.include=health,info,metrics,env

# Customize path for Actuator endpoints

management.endpoints.web.base-path=/admin

**Using application.yml:**

management:

endpoints:

web:

exposure:

include: "health,info,metrics,env"

base-path: /admin

endpoint:

health:

show-details: always

**Adding Micrometer Dependency (usually included with Actuator):**

<dependency>

<groupId>io.micrometer</groupId>

<artifactId>micrometer-core</artifactId>

</dependency>

**Creating a Custom Metrics Component:**

package com.example.bookstoreapi.metrics;

import io.micrometer.core.instrument.MeterRegistry;

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

import org.springframework.stereotype.Component;

@Component

public class CustomMetrics {

@Autowired

public CustomMetrics(MeterRegistry meterRegistry) {

meterRegistry.gauge("custom.book.count", this, CustomMetrics::getBookCount);

}

private int getBookCount() {

return 42; // Replace with actual book count logic

}

}

Access Custom Metrics: curl http://localhost:8080/admin/metrics

**Exercise 12:**

**Online Bookstore - Securing RESTful Endpoints with Spring Security**

**Adding Spring Security Dependency:**

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**Creating JWT Utility Class:**

package com.example.bookstoreapi.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtUtil {

private String secretKey = "your-secret-key"; // Replace with a secure key

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60)) // 1 hour

.signWith(SignatureAlgorithm.HS256, secretKey)

.compact();

}

public Claims extractClaims(String token) {

return Jwts.parser()

.setSigningKey(secretKey)

.parseClaimsJws(token)

.getBody();

}

public String extractUsername(String token) {

return extractClaims(token).getSubject();

}

public boolean isTokenExpired(String token) {

return extractClaims(token).getExpiration().before(new Date());

}

public boolean validateToken(String token, String username) {

return (username.equals(extractUsername(token)) && !isTokenExpired(token));

}

}

**Creating JWT Authentication Filter:**

package com.example.bookstoreapi.security;

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

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserDetailsService userDetailsService;

@Override

public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException {

String header = request.getHeader("Authorization");

String token = header != null && header.startsWith("Bearer ") ? header.substring(7) : null;

if (token != null && jwtUtil.validateToken(token, jwtUtil.extractUsername(token))) {

UserDetails userDetails = userDetailsService.loadUserByUsername(jwtUtil.extractUsername(token));

UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

return authentication;

}

return null;

}

@Override

protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain, Authentication authResult) throws IOException, ServletException {

SecurityContextHolder.getContext().setAuthentication(authResult);

chain.doFilter(request, response);

}

}

**Configure Security:**

package com.example.bookstoreapi.security;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.builders.WebSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserDetailsService userDetailsService;

@Bean

public JwtAuthenticationFilter jwtAuthenticationFilter() {

return new JwtAuthenticationFilter();

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/login").permitAll()

.anyRequest().authenticated()

.and()

.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService);

}

}

**CORS Handling:**

package com.example.bookstoreapi.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class CorsConfig implements WebMvcConfigurer {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("http://localhost:3000") // Replace with your front-end URL

.allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")

.allowedHeaders("\*")

.allowCredentials(true);

}

}

**Exercise 13:**

**Online Bookstore - Unit Testing REST Controllers**

**JUnit and Mockito Setup:**

<dependencies>

<!-- Spring Boot Starter Test (includes JUnit and Mockito) -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<!-- Mockito Core for mocking (optional, usually included in Spring Boot Starter Test) -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.0.0</version>

<scope>test</scope>

</dependency>

</dependencies>

### **MockMvc Setup**

**Create a Test Class for Your Controller:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.service.BookService;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

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

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(BookController.class)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private BookService bookService;

@Autowired

private ObjectMapper objectMapper;

@BeforeEach

public void setup() {

MockitoAnnotations.openMocks(this);

}

@Test

public void testCreateBook() throws Exception {

Book book = new Book(1L, "Title", "Author", 20.0, "ISBN123");

when(bookService.saveBook(any(Book.class))).thenReturn(book);

mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(book)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Title"))

.andExpect(jsonPath("$.author").value("Author"));

}

@Test

public void testGetBookById() throws Exception {

Book book = new Book(1L, "Title", "Author", 20.0, "ISBN123");

when(bookService.getBookById(1L)).thenReturn(book);

mockMvc.perform(get("/books/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Title"))

.andExpect(jsonPath("$.author").value("Author"));

}

@Test

public void testUpdateBook() throws Exception {

Book book = new Book(1L, "Updated Title", "Updated Author", 25.0, "ISBN123");

when(bookService.updateBook(eq(1L), any(Book.class))).thenReturn(book);

mockMvc.perform(put("/books/1")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(book)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Updated Title"))

.andExpect(jsonPath("$.author").value("Updated Author"));

}

@Test

public void testDeleteBook() throws Exception {

mockMvc.perform(delete("/books/1"))

.andExpect(status().isOk());

}

}

**Exercise 14:**

**Online Bookstore - Integration Testing for REST Services**

**Dependencies for Integration Testing:**

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>test</scope>

</dependency>

### **MockMvc Integration Testing**

**Create Integration Test Class:**

package com.example.bookstoreapi;

import com.example.bookstoreapi.controller.BookController;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.service.BookService;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.context.annotation.Import;

import org.springframework.http.MediaType;

import org.springframework.test.context.jdbc.Sql;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.annotation.WebMvcTest;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest

@Sql(scripts = "/test-schema.sql") // Optional: Load schema or data scripts

public class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private ObjectMapper objectMapper;

@Autowired

private BookService bookService;

@BeforeEach

public void setup() {

}

@Test

public void testCreateBook() throws Exception {

Book book = new Book(null, "Title", "Author", 20.0, "ISBN123");

mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(book)))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.title").value("Title"))

.andExpect(jsonPath("$.author").value("Author"));

}

@Test

public void testGetBookById() throws Exception {

// Assume a book with ID 1 exists in the database

mockMvc.perform(get("/books/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Title"))

.andExpect(jsonPath("$.author").value("Author"));

}

@Test

public void testUpdateBook() throws Exception {

Book book = new Book(1L, "Updated Title", "Updated Author", 25.0, "ISBN123");

mockMvc.perform(put("/books/1")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(book)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Updated Title"))

.andExpect(jsonPath("$.author").value("Updated Author"));

}

@Test

public void testDeleteBook() throws Exception {

mockMvc.perform(delete("/books/1"))

.andExpect(status().isNoContent());

}

}

### **Database Integration with H2**

**Configure H2 in application-test.properties:**

spring.datasource.url=jdbc:h2:mem:testdb;DB\_CLOSE\_DELAY=-1

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

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=create-drop

**Scenario 15:**

**Online Bookstore - API Documentation with Swagger**

**Add Swagger Dependency:**

<dependencies>

<!-- Springdoc OpenAPI for Swagger documentation -->

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>2.0.0</version>

</dependency>

</dependencies>

**Document Endpoints:**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.Book;

import com.example.bookstoreapi.service.BookService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.responses.ApiResponse;

import io.swagger.v3.oas.annotations.responses.ApiResponses;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import javax.validation.Valid;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@Operation(summary = "Create a new book", description = "Adds a new book to the bookstore")

@ApiResponse(responseCode = "201", description = "Book created successfully")

@PostMapping

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

Book createdBook = bookService.saveBook(book);

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

}

@Operation(summary = "Get a book by ID", description = "Fetches a book by its ID")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Book found"),

@ApiResponse(responseCode = "404", description = "Book not found")

})

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

@Operation(summary = "Update an existing book", description = "Updates the details of an existing book")

@ApiResponse(responseCode = "200", description = "Book updated successfully")

@PutMapping("/{id}")

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

Book updatedBook = bookService.updateBook(id, book);

return ResponseEntity.ok(updatedBook);

}

@Operation(summary = "Delete a book by ID", description = "Removes a book from the bookstore")

@ApiResponse(responseCode = "204", description = "Book deleted successfully")

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

**API documentation:**

**Running the Spring Boot application and accessing the Swagger UI:**

* Start the Spring Boot application.
* Open the browser and navigate to http://localhost:8080/swagger-ui.html or http://localhost:8080/swagger-ui/index.html (URL might vary based on your configuration).

**Swagger UI or Springdoc UI:**

* **Swagger UI**: Provides an interactive interface to view and test your API endpoints.
* **Springdoc UI**: Similar interface provided by Springdoc OpenAPI, usually accessible at /swagger-ui/index.html.