Exercise 1: Online Bookstore - Se4ng Up RESTful Services

Setup Spring Boot Project:

BookstoreApiApplica.on.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);
    }
}
```

BookController.java package

```
com.example.bookstoreapi.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import lombok.Getter; import lombok.Setter; import
lombok.ToString; import java.util.List;

@RestController
@RequestMapping("/api/books")
public class BookController {

    @GetMapping     public
List<Book> getBooks() {
        // This is just a placeholder. In a real application, you'd fetch data from a database.
        return List.of(
            new Book(1L, "Effective Java", "Joshua Bloch"),
new Book(2L, "Spring Boot in Action", "Craig Walls")
        );
    }
}
```

`Book.java` (Inside `controller` package)

```
package com.example.bookstoreapi.controller;


import lombok.Getter;
import lombok.Setter;

import lombok.ToString;


@Getter

@Setter

@ToString public class

Book {     private Long

id;     private String

title;     private String

author;


    public Book(Long id, String title, String author) {

        this.id = id;

this.title = title;

this.author = author;

    }

}
```

`pom.xml` (Dependencies section)

```
<dependencies>
    <!-- Spring Boot Starter Web for building web, including
RESTful, applications using Spring MVC -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Spring Boot DevTools for automatic restart and live
reload -->     <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>1.18.24</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Project Structure:

```
BookstoreAPI/
|
├── src/
|   ├── main/
|   |   ├── java/
|   |   |   └── com/
|   |   |       └── example/
|   |   |           └── bookstoreapi/
|   |   |               ├── BookstoreApiApplicaHon.java
|   |   |               └── controller/
|   |   |                   └── BookController.java |
|   └── resources/
|   |       ├── applicaHon.properHes
|   |      └── staHc/
|   └── test/
|       └── java/
|           └── com/
|               └── example/
|                   └── bookstoreapi/
|                       └── BookstoreApiApplicaHonTests.java
└── pom.xml
```

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

Spring Boot 3 introduced several new features and improvements. Here's an overview of the key enhancements and updates:

1. Java 21 Support

Spring Boot 3 requires Java 21 or later, aligning with the latest Java LTS (Long-Term Support) version. This update allows leveraging new language features and performance improvements in Java.

2. Spring Framework 6 Integration

Spring Boot 3 is built on Spring Framework 6, which includes:

- **Jakarta EE 9+ Support**: Migrates from `javax.*` to `jakarta.*` namespaces.
- **Native Compilation Support**: Enhanced support for GraalVM native images.
- **Enhanced Configuration**: Improvements in the configuration and property binding.

3. Enhanced Observability

- **Micrometer Integration**: Improved integration with Micrometer for metrics and monitoring.
- **Observability Support**: Enhanced support for distributed tracing, metrics, and logging.

4. Improved Dependency Management

- **Dependency Upgrades**: Updated versions of key dependencies to improve security and performance.
- **Simplified Dependency Resolution**: Enhanced dependency management with clearer dependency versions.

5. Native Compilation Support

- **GraalVM Native Image**: Enhanced support for GraalVM to build native executables, improving startup times and reducing memory consumption.

Exercise 2: Online Bookstore - CreaHng Basic REST Controllers

1. Define the Book Entity

package com.example.bookstore.model;

import javax.persistence.Entity; import

javax.persistence.GeneratedValue; import

javax.persistence.GenerationType; import

javax.persistence.Id;

```
@Entity public
class Book {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;     private String title;     private String
author;     private Double price;     private String isbn;

   // Getters and Setters
public Long getId() {
      return id;
   }

   public void setId(Long id) {
      this.id = id;
   }

   public String getTitle() {
return title;
   }

   public void setTitle(String title) {
this.title = title;
   }
   public String getAuthor() {
return author;
   }
```

```
    public void setAuthor(String author) {
this.author = author;
    }

    public Double getPrice() {
return price;
    }

    public void setPrice(Double price) {
this.price = price;
    }

    public String getIsbn() {
return isbn;
    }

    public void setIsbn(String isbn) {
this.isbn = isbn;
    }
}
```

2. Create the `BookRepository`

```
package com.example.bookstore.repository; import com.example.bookstore.model.Book; import
org.springframework.data.jpa.repository.JpaRepository;

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

3. Implement the `BookController`

```
package com.example.bookstore.controller;


import com.example.bookstore.model.Book; import
com.example.bookstore.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
   @GetMapping     public
List<Book> getAllBooks() {
return bookRepository.findAll();
   }
   // GET a single 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());
   }

   // POST a new book     @PostMapping     public ResponseEntity<Book>
createBook(@RequestBody Book book) {         Book savedBook =
bookRepository.save(book);         return
ResponseEntity.status(HttpStatus.CREATED).body(savedBook);
   }

   // PUT update an existing book
@PutMapping("/{id}")
   public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book
bookDetails) {
      Optional<Book> existingBook = bookRepository.findById(id);
if (existingBook.isPresent()) {            Book book =
existingBook.get();            book.setTitle(bookDetails.getTitle());
book.setAuthor(bookDetails.getAuthor());
book.setPrice(bookDetails.getPrice());
book.setIsbn(bookDetails.getIsbn());
bookRepository.save(book);            return
ResponseEntity.ok(book);
      }
      return ResponseEntity.notFound().build();
   }
```

```
    // DELETE a book by ID     @DeleteMapping("/{id}")     public
ResponseEntity<Void> deleteBook(@PathVariable Long id) {         if
(bookRepository.existsById(id)) {
bookRepository.deleteById(id);             return
ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }
}
```

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

1. Fetch a Book by Its ID Using a Path Variable

```
// GET a single book by ID @GetMapping("/{id}") public ResponseEn.ty<Book>
getBookById(@PathVariable Long id) {     Op.onal<Book> book =
bookRepository.findById(id);     return book.map(ResponseEn.ty::ok).orElseGet(() ->
ResponseEn.ty.notFound().build());
}
```

2. Filter Books Based on Query Parameters

```
// GET books with op.onal query parameters
@GetMapping("/search") public
List<Book> searchBooks(
        @RequestParam(value = ".tle", required = false) String .tle,
        @RequestParam(value = "author", required = false) String author) {

    if (.tle != null && author != null) {        return
bookRepository.findByTitleAndAuthor(.tle, author);
```

```
   } else if (.tle != null) {        return
bookRepository.findByTitle(.tle);
   } else if (author != null) {        return
bookRepository.findByAuthor(author);
   } else {        return
bookRepository.findAll();
   }
}
```

3. Update the `BookRepository` with Custom Queries

```
package com.example.bookstore.repository;

import com.example.bookstore.model.Book; import
org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;
public interface BookRepository extends JpaRepository<Book, Long> {

   List<Book> findByTitle(String title);

   List<Book> findByAuthor(String author);

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

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

1. Define the `Customer` Entity

```
package com.example.bookstoreapi.model;

import lombok.AllArgsConstructor;
import lombok.Getter; import
lombok.NoArgsConstructor; import
lombok.Setter; import
lombok.ToString;

import javax.persistence.Entity; import
javax.persistence.GeneratedValue; import
javax.persistence.GenerationType; import
javax.persistence.Id;

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ToString public class
Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;     private String name;     private String
email;     private String phone;     private String address;
}
```

2. Create the `CustomerRepository` Interface

```
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> {
}
```

3. Create the `CustomerController`

```
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 javax.servlet.http.HttpServletRequest;
import java.util.Optional;

@RestController
@RequestMapping("/customers")
public class CustomerController {
   @Autowired     private CustomerRepository
customerRepository;

   // Create a new customer using JSON request body
   @PostMapping("/json")     public ResponseEntity<Customer>
createCustomer(@RequestBody Customer customer) {         Customer savedCustomer =
customerRepository.save(customer);         return
ResponseEntity.status(HttpStatus.CREATED).body(savedCustomer);
   }

   // Register a new customer using form data
   @PostMapping("/form")     public ResponseEntity<Customer>
registerCustomer(HttpServletRequest request) {
       String name = request.getParameter("name");
       String email = request.getParameter("email");
       String phone = request.getParameter("phone");
       String address = request.getParameter("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);
   }
}
```

Exercise 5: Online Bookstore - Customizing Response Status and Headers

1. Customizing HTTP Status Codes with `@ResponseStatus`

```
import org.springframework.http.HttpStatus; import
org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/books")
public class BookController {

   @Autowired     private BookRepository
bookRepository;     // GET a single book by
ID with custom status

   @GetMapping("/{id}")
   @ResponseStatus(HttpStatus.OK) // Default status code for successful retrieval
public Book getBookById(@PathVariable Long id) {         return
bookRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Book not found with id " + id));
   }

   // POST a new book with custom status
   @PostMapping
   @ResponseStatus(HttpStatus.CREATED) // Default status code for resource creation
public Book createBook(@RequestBody Book book) {         return
bookRepository.save(book);
   }
   // PUT update an existing book with custom status
   @PutMapping("/{id}")
   @ResponseStatus(HttpStatus.OK) // Default status code for successful update     public
Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {
       Book book = bookRepository.findById(id)
               .orElseThrow(() -> new ResourceNotFoundException("Book not found with id " + id));
book.setTitle(bookDetails.getTitle());         book.setAuthor(bookDetails.getAuthor());
book.setPrice(bookDetails.getPrice());         book.setIsbn(bookDetails.getIsbn());         return
bookRepository.save(book);
   }
   // DELETE a book by ID with custom status
   @DeleteMapping("/{id}")
   @ResponseStatus(HttpStatus.NO_CONTENT) // Default status code for successful deletion with
no content     public void deleteBook(@PathVariable Long id) {         if
(bookRepository.existsById(id)) {             bookRepository.deleteById(id);
       } else {
           throw new ResourceNotFoundException("Book not found with id " + id);
       }
   }
}
```

Custom Exception Handling

```
// Custom exception class
@ResponseStatus(HttpStatus.NOT_FOUND) public class
ResourceNotFoundException extends RuntimeException {     public
ResourceNotFoundException(String message) {         super(message);
   }
}
```

```
// Global exception handler

@ControllerAdvice public class

GlobalExceptionHandler {


   @ExceptionHandler(ResourceNotFoundException.class)

   @ResponseStatus(HttpStatus.NOT_FOUND)

   public ResponseEntity<String>
handleResourceNotFoundException(ResourceNotFoundException ex) {

       return ResponseEntity

               .status(HttpStatus.NOT_FOUND)

               .body(ex.getMessage());

   }

}
```

2. Adding Custom Headers Using `ResponseEntity`

```
import org.springframework.http.HttpHeaders;

import org.springframework.http.ResponseEntity;


// GET all books with custom headers

@GetMapping

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

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

HttpHeaders headers = new HttpHeaders();

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

   return new ResponseEntity<>(books, headers, HttpStatus.OK); // HTTP 200 OK with custom
headers

}


// POST a new book with custom headers
```

```
@PostMapping public ResponseEntity<Book>
createBook(@RequestBody Book book) {
   Book savedBook = bookRepository.save(book);
HttpHeaders headers = new HttpHeaders();
headers.add("Location", "/books/" + savedBook.getId());
return new ResponseEntity<>(savedBook, headers,
HttpStatus.CREATED); // HTTP 201 Created with Location
header

}
```

Exercise 6: Online Bookstore - ExcepHon Handling in REST Controllers

1. Define the Global ExcepHon Handler

```
package com.example.bookstore.excep.on;


import org.springframework.h_p.H_pStatus; import
org.springframework.h_p.ResponseEn.ty; import
org.springframework.web.bind.annota.on.ControllerAdvice; import
org.springframework.web.bind.annota.on.Excep.onHandler; import
org.springframework.web.bind.annota.on.ResponseStatus; import
org.springframework.web.bind.MethodArgumentNotValidExcep.on; import
org.springframework.web.bind.annota.on.ResponseBody;

import javax.valida.on.ConstraintViola.onExcep.on;
```

```
// Global Excep.on Handler

@ControllerAdvice public class

GlobalExcep.onHandler {


    // Handle resource not found excep.ons

    @Excep.onHandler(ResourceNotFoundExcep.on.class)

    @ResponseStatus(H_pStatus.NOT_FOUND)

    @ResponseBody

    public ResponseEn.ty<ErrorResponse>
handleResourceNotFoundExcep.on(ResourceNotFoundExcep.on ex) {

        ErrorResponse errorResponse = new ErrorResponse("Resource Not Found",
ex.getMessage());         return new ResponseEn.ty<>(errorResponse,
H_pStatus.NOT_FOUND);

    }


    // Handle valida.on excep.ons

    @Excep.onHandler(MethodArgumentNotValidExcep.on.class)

    @ResponseStatus(H_pStatus.BAD_REQUEST)

    @ResponseBody

    public ResponseEn.ty<ErrorResponse>
handleValida.onExcep.on(MethodArgumentNotValidExcep.on ex) {

        String errorMessage = ex.getBindingResult().getAllErrors().stream()

            .map(error -> error.getDefaultMessage())

            .reduce((message1, message2) -> message1 + ", " + message2)

            .orElse("Valida.on error");
        ErrorResponse errorResponse = new ErrorResponse("Valida.on Error",
errorMessage);        return new ResponseEn.ty<>(errorResponse,
H_pStatus.BAD_REQUEST);
    }


    // Handle constraint viola.ons (e.g., @Valid constraints)
    @Excep.onHandler(ConstraintViola.onExcep.on.class)
    @ResponseStatus(H_pStatus.BAD_REQUEST)
    @ResponseBody
    public ResponseEn.ty<ErrorResponse>
handleConstraintViola.onExcep.on(ConstraintViola.onExcep.on ex) {
        String errorMessage = ex.getConstraintViola.ons().stream()
            .map(viola.on -> viola.on.getMessage())
            .reduce((message1, message2) -> message1 + ", " + message2)
            .orElse("Constraint viola.on");
        ErrorResponse errorResponse = new ErrorResponse("Constraint Viola.on",
errorMessage);        return new ResponseEn.ty<>(errorResponse,
H_pStatus.BAD_REQUEST);
    }


    // Handle any other excep.ons
    @Excep.onHandler(Excep.on.class)
    @ResponseStatus(H_pStatus.INTERNAL_SERVER_ERROR)
    @ResponseBody     public ResponseEn.ty<ErrorResponse>
handleGenericExcep.on(Excep.on ex) {
        ErrorResponse errorResponse = new ErrorResponse("Internal Server Error",
ex.getMessage());

        return new ResponseEn.ty<>(errorResponse,
H_pStatus.INTERNAL_SERVER_ERROR);

    }

}
```

2. Define the `ErrorResponse` Class

```
package com.example.bookstore.exception;

public class ErrorResponse {


    private String error;

private String message;


    public ErrorResponse(String error, String message) {

this.error = error;         this.message = message;

    }


    // Getters and Setters

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: Online Bookstore - IntroducHon to Data Transfer Objects (DTOs)

1. Define DTO Classes package

```
com.example.bookstore.dto;

public class BookDTO {

    private Long id;
private String title;
private String author;
private Double price;
private String isbn;

    // Constructors
public BookDTO() {
    }

    public BookDTO(Long id, String title, String author, Double price, String isbn) {
        this.id = id;
this.title = title;
this.author = author;
this.price = price;
this.isbn = isbn;
    }

    // Getters and Setters
public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
return title;     }

    public void setTitle(String title) {
this.title = title;
    }

    public String getAuthor() {
return author;
    }

    public void setAuthor(String author) {
this.author = author;
    }
    public Double getPrice() {
return price;
    }


    public void setPrice(Double price) {
this.price = price;

    }


    public String getIsbn() {
return isbn;
    }


    public void setIsbn(String isbn) {
this.isbn = isbn;
    }
}
```

CustomerDTO

```
package com.example.bookstore.dto;


public class CustomerDTO {


    private Long id;     private
String name;     private String
email;     private String
phoneNumber;
```

```
   // Constructors     public
CustomerDTO() {
   }

   public CustomerDTO(Long id, String name, String email, String phoneNumber) {
       this.id = id;
this.name = name;
this.email = email;
this.phoneNumber =
phoneNumber;
   }

   // Getters and Setters
public Long getId() {
       return id;
   }

   public void setId(Long id) {
       this.id = id;
   }

   public String getName() {
return name;
   }

   public void setName(String name) {
this.name = name;
   }
```

```
    public String getEmail() {
return email;
    }

    public void setEmail(String email) {
this.email = email;
    }
    public String getPhoneNumber() {
return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
this.phoneNumber = phoneNumber;
    }
}
```

2. Mapping Entities to DTOs Using MapStruct or ModelMapper

```
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>1.5.5.Final</version>
</dependency>
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct-processor</artifactId>
    <version>1.5.5.Final</version>
    <scope>provided</scope>
</dependency>
```
Create

Mapper Interfaces

```
package com.example.bookstore.mapper;
import com.example.bookstore.dto.BookDTO;
import com.example.bookstore.model.Book;
import org.mapstruct.Mapper; import
org.mapstruct.factory.Mappers;

@Mapper
public interface BookMapper {
    BookMapper INSTANCE = Mappers.getMapper(BookMapper.class);

    BookDTO bookToBookDTO(Book book);

    Book bookDTOToBook(BookDTO bookDTO);
} package
com.example.bookstore.mapper;

import com.example.bookstore.dto.CustomerDTO;
import com.example.bookstore.model.Customer;
import org.mapstruct.Mapper; import
org.mapstruct.factory.Mappers;

@Mapper
public interface CustomerMapper {package com.example.bookstore.config;
import org.modelmapper.ModelMapper; import
org.springframework.context.annotation.Bean; import
org.springframework.context.annotation.Configuration;

@Configuration
public class ModelMapperConfig {

    @Bean     public ModelMapper
modelMapper() {         return new
ModelMapper();
    }
}

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

    CustomerDTO customerToCustomerDTO(Customer customer);

    Customer customerDTOToCustomer(CustomerDTO customerDTO);
}
```

Using ModelMapper

```
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>3.1.1</version>
</dependency>
```

Configure ModelMapper package

com.example.bookstore.config;

```
import org.modelmapper.ModelMapper; import
org.springframework.context.annotation.Bean; import
org.springframework.context.annotation.Configuration;
@Configuration public class ModelMapperConfig {

   @Bean     public ModelMapper
modelMapper() {         return new
ModelMapper();
   }
}
```

Use ModelMapper for Mapping

```
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);
```

```
    }
}
```

3. Custom Serialization/Deserialization Using Jackson Annotations

```
package com.example.bookstore.dto;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonProperty;

public class BookDTO {

    private Long id;

    @JsonProperty("book_title")
private String title;

    private String author;

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "$#.00")
private Double price;

    private String isbn;

    // Constructors, Getters, and Setters
}
```

Exercise 8: Online Bookstore - ImplemenHng CRUD OperaHons

1. CRUD Endpoints

```
package com.example.bookstore.controller;

import com.example.bookstore.dto.BookDTO; import

com.example.bookstore.mapper.BookMapper; import

com.example.bookstore.model.Book; import

com.example.bookstore.repository.BookRepository; import

com.example.bookstore.excep.on.ResourceNotFoundExcep.on; import

org.springframework.beans.factory.annota.on.Autowired; import

org.springframework.h_p.ResponseEn.ty; import

org.springframework.web.bind.annota.on.*;


import javax.valida.on.Valid;

import java.u.l.List; import

java.u.l.Op.onal;


@RestController

@RequestMapping("/books") public

class BookController {


   @Autowired     private BookRepository

bookRepository;


   @Autowired
   private BookMapper bookMapper;


   // Create a new book

@PostMapping

   public ResponseEn.ty<BookDTO> createBook(@Valid @RequestBody BookDTO
bookDTO) {

       Book book = bookMapper.bookDTOToBook(bookDTO);

Book savedBook = bookRepository.save(book);

       return
ResponseEn.ty.status(201).body(bookMapper.bookToBookDTO(savedBook));

   }


   // Read all books     @GetMapping

public List<BookDTO> getAllBooks() {

return bookRepository.findAll().stream()

             .map(bookMapper::bookToBookDTO)

             .toList();

   }


   // Read a single book by ID

   @GetMapping("/{id}")

   public ResponseEn.ty<BookDTO> getBookById(@PathVariable Long id) {
        Book book = bookRepository.findById(id)

                .orElseThrow(() -> new ResourceNotFoundExcep.on("Book not found with
id " + id));

        return ResponseEn.ty.ok(bookMapper.bookToBookDTO(book));

    }



    // Update a book by ID

@PutMapping("/{id}")

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

        Book exis.ngBook = bookRepository.findById(id)

                .orElseThrow(() -> new ResourceNotFoundExcep.on("Book not found with
id " + id));



        Book book = bookMapper.bookDTOToBook(bookDTO);

        book.setId(id);

        Book updatedBook = bookRepository.save(book);        return

ResponseEn.ty.ok(bookMapper.bookToBookDTO(updatedBook));

    }



    // Delete a book by ID

    @DeleteMapping("/{id}")
```

```
   public ResponseEn.ty<Void> deleteBook(@PathVariable Long id) {

if (bookRepository.existsById(id)) {

bookRepository.deleteById(id);           return

ResponseEn.ty.noContent().build();

      } else {

         throw new ResourceNotFoundExcep.on("Book not found with id " + id);

      }

   }

}
```

CustomerController

```
package com.example.bookstore.controller;

import com.example.bookstore.dto.CustomerDTO; import
com.example.bookstore.mapper.CustomerMapper; import
com.example.bookstore.model.Customer; import
com.example.bookstore.repository.CustomerRepository; import
com.example.bookstore.excep.on.ResourceNotFoundExcep.on; import
org.springframework.beans.factory.annota.on.Autowired; import
org.springframework.h_p.ResponseEn.ty; import
org.springframework.web.bind.annota.on.*;

import javax.valida.on.Valid; import
java.u.l.List;

@RestController
@RequestMapping("/customers") public
class CustomerController {

   @Autowired
   private CustomerRepository customerRepository;
   @Autowired
   private CustomerMapper customerMapper;

   // Create a new customer
   @PostMapping
   public ResponseEn.ty<CustomerDTO> createCustomer(@Valid @RequestBody
CustomerDTO customerDTO) {
Customer customer =
customerMapper.customerDTOToCustomer(customerDTO);
      Customer savedCustomer = customerRepository.save(customer);
return
ResponseEn.ty.status(201).body(customerMapper.customerToCustomerDTO(savedC
ustomer));
   }

   // Read all customers
@GetMapping
   public List<CustomerDTO> getAllCustomers() {
return customerRepository.findAll().stream()
            .map(customerMapper::customerToCustomerDTO)
            .toList();
}

   // Read a single customer by ID
@GetMapping("/{id}")
   public ResponseEn.ty<CustomerDTO> getCustomerById(@PathVariable Long id) {
      Customer customer = customerRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundExcep.on("Customer not found
with id " + id));        return
ResponseEn.ty.ok(customerMapper.customerToCustomerDTO(customer));
   }

   // Update a customer by ID
@PutMapping("/{id}")
   public ResponseEn.ty<CustomerDTO> updateCustomer(@PathVariable Long id,
@Valid @RequestBody CustomerDTO customerDTO) {
      Customer exis.ngCustomer = customerRepository.findById(id)
```

```
            .orElseThrow(() -> new ResourceNotFoundExcep.on("Customer not found
with id " + id));

        Customer customer =
customerMapper.customerDTOToCustomer(customerDTO);
customer.setId(id);
        Customer updatedCustomer = customerRepository.save(customer);
return
ResponseEn.ty.ok(customerMapper.customerToCustomerDTO(updatedCustomer));
    }

    // Delete a customer by ID
@DeleteMapping("/{id}")
    public ResponseEn.ty<Void> deleteCustomer(@PathVariable Long id) {
if (customerRepository.existsById(id)) {
customerRepository.deleteById(id);
            return ResponseEn.ty.noContent().build();
        } else {
            throw new ResourceNotFoundExcep.on("Customer not found with id " + id);
        }
    }
}
```

2. ValidaHng Input Data

```
package com.example.bookstore.dto;


import javax.valida.on.constraints.NotNull;

import javax.valida.on.constraints.Size; import

javax.valida.on.constraints.Min;


public class BookDTO {


    private Long id;
```

```
   @NotNull
   @Size(min = 1, max = 100)
private String .tle;


   @NotNull
   @Size(min = 1, max = 50)
private String author;


   @NotNull
@Min(0)    private
Double price;


   @NotNull
   @Size(min = 10, max = 13)
private String isbn;


   // Constructors, Ge_ers, and Se_ers
}
```

CustomerDTO Example:

```
package com.example.bookstore.dto;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.validation.constraints.Email;
public class CustomerDTO {
    private Long id;
    @NotNull
    @Size(min = 1, max = 50)
private String name;

    @NotNull
    @Email     private
String email;

    @Size(min = 10, max = 15)
private String phoneNumber;

    // Constructors, Getters, and Setters
}
```

3. Implementing Optimistic Locking

```
package com.example.bookstore.model;

import javax.persistence.*; import
javax.validation.constraints.NotNull; import
javax.validation.constraints.Size; import
javax.validation.constraints.Min;

@Entity public
class Book {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

   @Version     private
Long version;
@NotNull

   @Size(min = 1, max = 100)
   private String title;


   @NotNull
   @Size(min = 1, max = 50)
private String author;


   @NotNull
@Min(0)     private
Double price;


   @NotNull
   @Size(min = 10, max = 13)
private String isbn;


   // Constructors, Getters, and Setters
}
```

Customer Entity Example:

```
package com.example.bookstore.model;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.validation.constraints.Email;

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

    @Version     private
Long version;
```

```
    @NotNull
    @Size(min = 1, max = 50)
private String name;

    @NotNull
    @Email     private
String email;

    @Size(min = 10, max = 15)
private String phoneNumber;

    // Constructors, Getters, and Setters
}
```

Exercise 9: Online Bookstore - Understanding HATEOAS

1. Add Spring HATEOAS Dependency

```
<dependency>

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

   <ar.factId>spring-boot-starter-hateoas</ar.factId>

</dependency>
```

2. Create Resource Assemblers

```
package com.example.bookstore.assembler;


import com.example.bookstore.controller.BookController;

import com.example.bookstore.dto.BookDTO; import

com.example.bookstore.model.Book; import

org.springframework.hateoas.En.tyModel; import
org.springframework.hateoas.Link; import

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

org.springframework.stereotype.Component;


@Component

public class BookResourceAssembler {


   public En.tyModel<BookDTO> toModel(BookDTO bookDTO) {

       En.tyModel<BookDTO> bookResource = En.tyModel.of(bookDTO);

       Link selfLink =
WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).get
BookById(bookDTO.getId())).withSelfRel();

       Link allBooksLink =
WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).get

AllBooks()).withRel("all-books");        bookResource.add(selfLink, allBooksLink);

return bookResource;

   }

}
```

CustomerResourceAssembler

```
package com.example.bookstore.assembler;

import com.example.bookstore.controller.CustomerController;
import com.example.bookstore.dto.CustomerDTO;
import com.example.bookstore.model.Customer;
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 CustomerResourceAssembler {

    public EntityModel<CustomerDTO> toModel(CustomerDTO
customerDTO) {
        EntityModel<CustomerDTO> customerResource =
EntityModel.of(customerDTO);
        Link selfLink =
WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(CustomerC
ontroller.class).getCustomerById(customerDTO.getId())).withSe
lfRel();
        Link allCustomersLink =
WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(CustomerC
ontroller.class).getAllCustomers()).withRel("all-customers");
customerResource.add(selfLink, allCustomersLink);
return customerResource;
    }
}
```

3. Modify Controllers to Include Links

```
package com.example.bookstore.controller;

import com.example.bookstore.dto.BookDTO; import
com.example.bookstore.assembler.BookResourceAssembler; import
com.example.bookstore.model.Book; import
com.example.bookstore.repository.BookRepository; import
com.example.bookstore.exception.ResourceNotFoundException; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.hateoas.EntityModel; import
org.springframework.http.ResponseEntity;

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

import javax.validation.Valid;
import java.util.List; import
java.util.Optional;
```

```
@RestController

@RequestMapping("/books")

public class BookController {


   @Autowired    private BookRepository

bookRepository;


   @Autowired    private BookResourceAssembler

bookResourceAssembler;


   @PostMapping

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

       Book book = new Book(); // Assuming you have a method to convert DTO to Entity

book.setTitle(bookDTO.getTitle());        book.setAuthor(bookDTO.getAuthor());

book.setPrice(bookDTO.getPrice());        book.setIsbn(bookDTO.getIsbn());

       Book savedBook = bookRepository.save(book);

       BookDTO savedBookDTO = new BookDTO(savedBook.getId(), savedBook.getTitle(),
savedBook.getAuthor(), savedBook.getPrice(), savedBook.getIsbn());        return
ResponseEntity.status(201).body(bookResourceAssembler.toModel(savedBookDTO));

   }

   @GetMapping    public

List<EntityModel<BookDTO>> getAllBooks() {

return bookRepository.findAll().stream()

               .map(book -> bookResourceAssembler.toModel(new BookDTO(book.getId(),
book.getTitle(), book.getAuthor(), book.getPrice(), book.getIsbn())))

               .toList();

   }
   @GetMapping("/{id}")     public ResponseEntity<EntityModel<BookDTO>>
getBookById(@PathVariable Long id) {
        Book book = bookRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Book not found with id " + id));
        BookDTO bookDTO = new BookDTO(book.getId(), book.getTitle(), book.getAuthor(),
book.getPrice(), book.getIsbn());          return
ResponseEntity.ok(bookResourceAssembler.toModel(bookDTO));
    }


   @PutMapping("/{id}")
   public ResponseEntity<EntityModel<BookDTO>> updateBook(@PathVariable Long id, @Valid
@RequestBody BookDTO bookDTO) {
        Book existingBook = bookRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Book not found with id " + id));


        existingBook.setTitle(bookDTO.getTitle());
existingBook.setAuthor(bookDTO.getAuthor());
existingBook.setPrice(bookDTO.getPrice());
existingBook.setIsbn(bookDTO.getIsbn());

        Book updatedBook = bookRepository.save(existingBook);
        BookDTO updatedBookDTO = new BookDTO(updatedBook.getId(), updatedBook.getTitle(),
updatedBook.getAuthor(), updatedBook.getPrice(), updatedBook.getIsbn());          return
ResponseEntity.ok(bookResourceAssembler.toModel(updatedBookDTO));
    }


   @DeleteMapping("/{id}")     public ResponseEntity<Void>
deleteBook(@PathVariable Long id) {          if
(bookRepository.existsById(id)) {
```

```
bookRepository.deleteById(id);             return
ResponseEntity.noContent().build();
        } else {
            throw new ResourceNotFoundException("Book not found with id " + id);
        }
    }
}
```

CustomerController

```
package com.example.bookstore.controller;


import com.example.bookstore.dto.CustomerDTO; import
com.example.bookstore.assembler.CustomerResourceAssembler; import
com.example.bookstore.model.Customer; import
com.example.bookstore.repository.CustomerRepository; import
com.example.bookstore.excep.on.ResourceNotFoundExcep.on; import
org.springframework.beans.factory.annota.on.Autowired; import
org.springframework.hateoas.En.tyModel;

import org.springframework.h_p.ResponseEn.ty; import
org.springframework.web.bind.annota.on.*;


import javax.valida.on.Valid; import
java.u.l.List;


@RestController
```

```
@RequestMapping("/customers") public

class CustomerController {


   @Autowired     private CustomerRepository

customerRepository;


   @Autowired     private CustomerResourceAssembler

customerResourceAssembler;


   @PostMapping

   public ResponseEn.ty<En.tyModel<CustomerDTO>> createCustomer(@Valid
@RequestBody CustomerDTO customerDTO) {

      Customer customer = new Customer(); // Assuming you have a method to

convert DTO to En.ty         customer.setName(customerDTO.getName());

customer.setEmail(customerDTO.getEmail());

customer.setPhoneNumber(customerDTO.getPhoneNumber());

      Customer savedCustomer = customerRepository.save(customer);
CustomerDTO savedCustomerDTO = new CustomerDTO(savedCustomer.getId(),
savedCustomer.getName(), savedCustomer.getEmail(),
savedCustomer.getPhoneNumber());

      return
ResponseEn.ty.status(201).body(customerResourceAssembler.toModel(savedCusto
merDTO));

   }
   @GetMapping     public List<En.tyModel<CustomerDTO>>
getAllCustomers() {        return
customerRepository.findAll().stream()
            .map(customer -> customerResourceAssembler.toModel(new
CustomerDTO(customer.getId(), customer.getName(), customer.getEmail(),
customer.getPhoneNumber())))
            .toList();
   }


   @GetMapping("/{id}")
   public ResponseEn.ty<En.tyModel<CustomerDTO>>
getCustomerById(@PathVariable Long id) {
      Customer customer = customerRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundExcep.on("Customer not found
with id " + id));
      CustomerDTO customerDTO = new CustomerDTO(customer.getId(),
customer.getName(), customer.getEmail(), customer.getPhoneNumber());
      return
ResponseEn.ty.ok(customerResourceAssembler.toModel(customerDTO));
   }
   @PutMapping("/{id}")
   public ResponseEn.ty<En.tyModel<CustomerDTO>>
updateCustomer(@PathVariable Long id, @Valid @RequestBody CustomerDTO
customerDTO) {
      Customer exis.ngCustomer = customerRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundExcep.on("Customer not found
with id " + id));
```

```
        exis.ngCustomer.setName(customerDTO.getName());
exis.ngCustomer.setEmail(customerDTO.getEmail());
exis.ngCustomer.setPhoneNumber(customerDTO.getPhoneNumber());


        Customer updatedCustomer = customerRepository.save(exis.ngCustomer);
        CustomerDTO updatedCustomerDTO = new
CustomerDTO(updatedCustomer.getId(), updatedCustomer.getName(),
updatedCustomer.getEmail(), updatedCustomer.getPhoneNumber());
        return
ResponseEn.ty.ok(customerResourceAssembler.toModel(updatedCustomerDTO));
    }


    @DeleteMapping("/{id}")     public ResponseEn.ty<Void>
deleteCustomer(@PathVariable Long id) {         if
(customerRepository.existsById(id)) {
customerRepository.deleteById(id);           return
ResponseEn.ty.noContent().build();
        } else {
            throw new ResourceNotFoundExcep.on("Customer not found with id " + id);
        }
    }
}
```

Exercise 10: Online Bookstore - Configuring Content NegoHaHon

1. Configure Content NegoHaHon in Spring Boot

```
<dependency>

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

   <ar.factId>spring-boot-starter-web</ar.factId>

</dependency>

<dependency>

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

   <ar.factId>spring-boot-starter-xml</ar.factId>

</dependency> import

org.springframework.context.annota.on.Bean; import

org.springframework.context.annota.on.Configura.on;

import
org.springframework.web.servlet.config.annota.on.ContentNego.a.onConfigurer;
import org.springframework.web.servlet.config.annota.on.WebMvcConfigurer;


@Configura.on public class WebConfig implements

WebMvcConfigurer {


   @Override

   public void configureContentNego.a.on(ContentNego.a.onConfigurer
configurer) {        configurer.favorParameter(false)

              .ignoreAcceptHeader(false)

              .defaultContentType(org.springframework.h_p.MediaType.APPLICATION_J
SON)

              .mediaType("json",
org.springframework.h_p.MediaType.APPLICATION_JSON)

              .mediaType("xml",
org.springframework.h_p.MediaType.APPLICATION_XML);
    }
}
```

2. Implement Logic Based on `Accept` Header

```
import org.springframework.http.MediaType; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RestController; import
java.util.HashMap; import java.util.Map;

@RestController
@RequestMapping("/books")
public class BookController {

    @GetMapping(value = "/list", produces = { MediaType.APPLICATION_JSON_VALUE,
MediaType.APPLICATION_XML_VALUE })
    public Map<String, String> getBooks() {
        Map<String, String> books = new HashMap<>();
books.put("1", "Spring Boot in Action");
books.put("2", "Effective Java");        return books;
    }
}
```

Exercise 11: Online Bookstore - IntegraHng Spring Boot Actuator

1. Add Actuator Dependency <dependency>

```
    <groupId>org.springframework.boot</groupId>
    <ar.factId>spring-boot-starter-actuator</ar.factId>
```

```
</dependency>
```

2. Expose Actuator Endpoints

```
# Enable all actuator endpoints management.endpoints.web.exposure.include=*

# Enable specific actuator endpoints
management.endpoints.web.exposure.include=health,info,metrics management:
  endpoints:
    web:
exposure:
        include: health,info,metrics
management.endpoints.web.base-path=/actuator
management.endpoints.web.path-mapping.health=health-check management:

  endpoints:
web:
      base-path: /actuator       path-
mapping:
        health: health-check
```

4. Expose Custom Metrics

```
import io.micrometer.core.instrument.MeterRegistry; import
org.springframework.stereotype.Component;

@Component
```

```
public class CustomMetrics {

    private final MeterRegistry meterRegistry;

    public CustomMetrics(MeterRegistry meterRegistry) {
this.meterRegistry = meterRegistry;        this.registerCustomMetrics();
    }

    private void registerCustomMetrics() {
meterRegistry.gauge("custom.metric", 42);  // Register a simple gauge metric
    }
}
```

Exercise 12: Online Bookstore - Securing RESTful Endpoints with Spring Security

1. Add Spring Security to Your Project

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <arHfactId>spring-boot-starter-security</arHfactId>
</dependency>
```

b. Create a Security Configuration Class:

```
import org.springframework.context.annotation.Bean; import
org.springframework.context.annotation.Configuration;
import
```

```
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuil
der; 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.config.annotation.web.builders.HttpSecurity; import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableWebSecurity public class SecurityConfig extends
WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
http.csrf().disable()
         .authorizeRequests()
         .antMatchers("/public/**").permitAll()  // Allow public endpoints
         .anyRequest().authenticated()          // Secure all other endpoints
         .and()
         .addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
   }

   @Override     protected void configure(AuthenticationManagerBuilder auth)
throws Exception {         // Configure authentication provider
   }
   @Bean     public JwtAuthenticationFilter
jwtAuthenticationFilter() {         return new
JwtAuthenticationFilter();
   }
}
```

2. Implement JWT-Based Authentication

```
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";  // Use a strong secret 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));
    }
}
```

b. Create JWT Authentication Filter:

```
import org.springframework.security.core.context.SecurityContextHolder; import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter; import
javax.servlet.FilterChain; import javax.servlet.ServletException; import
javax.servlet.ServletRequest; import javax.servlet.ServletResponse; import
javax.servlet.http.HttpServletRequest; import java.io.IOException;

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
   private JwtUtil jwtUtil;


   public JwtAuthenticationFilter(JwtUtil jwtUtil) {
this.jwtUtil = jwtUtil;
   }


   @Override     public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)             throws IOException, ServletException {
       HttpServletRequest httpRequest = (HttpServletRequest) request;
       String authHeader = httpRequest.getHeader("Authorization");


       if (authHeader != null && authHeader.startsWith("Bearer ")) {
String token = authHeader.substring(7);             if
(jwtUtil.validateToken(token, jwtUtil.extractUsername(token))) {
               SecurityContextHolder.getContext().setAuthentication(jwtUtil.getAuthentication(token));
           }
       }
       chain.doFilter(request, response);
   }
}
```

3. Configure CORS Handling

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

@Configuration
public class WebConfig implements WebMvcConfigurer {

   @Override
   public void addCorsMappings(CorsRegistry registry) {
       registry.addMapping("/**")
               .allowedOrigins("*")
               .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
```

```
            .allowedHeaders("*")
            .allowCredentials(true);
    }
}
```

Exercise 13: Online Bookstore - Unit TesHng REST Controllers

1. JUnit and Mockito Setup

```
<dependency>

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

   <ar.factId>spring-boot-starter-test</ar.factId>

   <scope>test</scope>

</dependency>

<dependency>

   <groupId>org.mockito</groupId>

   <ar.factId>mockito-core</ar.factId>

   <version>4.0.0</version> <!-- or the latest version -->

   <scope>test</scope>

</dependency>
```

2. Use MockMvc to Write Unit Tests

```
import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import

org.mockito.InjectMocks; import org.mockito.Mock; import

org.mockito.MockitoAnnota.ons; import

org.springframework.beans.factory.annota.on.Autowired; import
org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest; import

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

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

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

org.springframework.web.context.WebApplica.onContext;

import sta.c
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import sta.c
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import sta.c
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import sta.c
org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;


@WebMvcTest(BookController.class) public

class BookControllerTest {


   @Autowired     private

MockMvc mockMvc;


   @MockBean     private

BookService bookService;


   @BeforeEach

void setUp() {
        MockitoAnnota.ons.openMocks(this);

    }


    @Test     void testGetBookById() throws

Excep.on {

        // Mock the service layer
when(bookService.getBookById(1L)).thenReturn(new Book(1L, "Effec.ve Java",
"Joshua Bloch"));


        // Perform the request and verify the response

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

                .andExpect(status().isOk())

                .andExpect(jsonPath("$..tle").value("Effec.ve Java"))

                .andExpect(jsonPath("$.author").value("Joshua Bloch"));

    }


    @Test     void testCreateBook() throws

Excep.on {

        Book book = new Book(1L, "Clean Code", "Robert C. Mar.n");


        // Mock the service layer

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


        // Perform the request and verify the response

mockMvc.perform(post("/books")
                 .contentType("applica.on/json")
                 .content("{\".tle\":\"Clean Code\",\"author\":\"Robert C. Mar.n\"}"))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$..tle").value("Clean Code"))
                .andExpect(jsonPath("$.author").value("Robert C. Mar.n"));
    }
}
```

Exercise 14: Online Bookstore - IntegraHon TesHng for REST Services

1. Set Up Spring Test

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <ar.factId>spring-boot-starter-test</ar.factId>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>com.h2database</groupId>
   <ar.factId>h2</ar.factId>
   <scope>test</scope>
</dependency>
```

2. MockMvc IntegraHon

```
import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import
org.springframework.beans.factory.annota.on.Autowired; import
org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest; import

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

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

org.springframework.h_p.MediaType; import

org.springframework.test.context.Ac.veProfiles;

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

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

org.springframework.web.context.WebApplica.onContext;


import sta.c
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import sta.c
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import sta.c
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import sta.c
org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import sta.c
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import sta.c
org.springframework.test.web.servlet.result.MockMvcResultMatchers.header;


@SpringBootTest

@AutoConfigureMockMvc

@Ac.veProfiles("test") public class

BookControllerIntegra.onTest {
   @Autowired     private
MockMvc mockMvc;


   @Autowired     private BookRepository bookRepository; // Assuming you use
Spring Data JPA     @BeforeEach     public void setUp() {
       // Clear the database before each test if necessary
bookRepository.deleteAll();
   }


   @Test     void testGetBookById() throws
Excep.on {
       // Arrange: Set up your test data
       Book book = new Book(1L, "Effec.ve Java", "Joshua Bloch");
bookRepository.save(book);


       // Act & Assert: Perform the request and verify the response
mockMvc.perform(get("/books/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$..tle").value("Effec.ve Java"))
            .andExpect(jsonPath("$.author").value("Joshua Bloch"));
   }


   @Test     void testCreateBook() throws
Excep.on {
        // Act & Assert: Perform the request and verify the response
mockMvc.perform(post("/books")
.contentType(MediaType.APPLICATION_JSON)

             .content("{\".tle\":\"Clean Code\",\"author\":\"Robert C. Mar.n\"}"))
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$..tle").value("Clean Code"))
            .andExpect(jsonPath("$.author").value("Robert C. Mar.n"));
   }
}
```

3. Database IntegraHon

```
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.database-
platorm=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

Scenario 15: Online Bookstore - API DocumentaHon with Swagger

1. Add Swagger or Springdoc Dependency

```
<dependency>
   <groupId>org.springdoc</groupId>
   <ar.factId>springdoc-openapi-ui</ar.factId>
   <version>2.0.0</version> <!-- Use the latest version -->
```

```
</dependency>
```

2. Document Endpoints

```
import org.springframework.web.bind.annota.on.*; import
io.swagger.v3.oas.annota.ons.Opera.on; import
io.swagger.v3.oas.annota.ons.Parameter; import
io.swagger.v3.oas.annota.ons.responses.ApiResponse; import
io.swagger.v3.oas.annota.ons.tags.Tag;

@RestController
@RequestMapping("/books")
@Tag(name = "Book Controller", descrip.on = "APIs for managing books") public
class BookController {

   @GetMapping("/{id}")
   @Opera.on(summary = "Get a book by ID",
descrip.on = "Retrieve the details of a book by its ID",
responses = {
               @ApiResponse(responseCode = "200", descrip.on = "Book found"),
               @ApiResponse(responseCode = "404", descrip.on = "Book not found")
           })
   public Book getBookById(
       @Parameter(descrip.on = "ID of the book to be retrieved") @PathVariable Long
id) {
       // Implementa.on
   }
   @PostMapping
   @Opera.on(summary = "Create a new book",
descrip.on = "Add a new book to the collec.on",
            responses = {
               @ApiResponse(responseCode = "201", descrip.on = "Book created"),
               @ApiResponse(responseCode = "400", descrip.on = "Invalid input")
            })
   public Book createBook(
      @RequestBody Book book) {
      // Implementa.on
   }
}
```

b. Document Models:

```
import io.subhransu.v3.oas.annotations.media.Schema;

@Schema(description = "Book model")
public class Book {

   @Schema(description = "ID of the book", example = "1")
private Long id;

   @Schema(description = "Title of the book", example = "Effective Java")
private String title;
```

```
    @Schema(description = "Author of the book", example = "Joshua Bloch")
private String author;

    // Getters and Setters
}
```

3. Generate and Review API Documentation

http://localhost:8080/subhransu-ui.html