**WEEK\_5 EXERCISES:**

**Exercise 1 :Introduction to Data Transfer Objects**

**BookDTO:**

public class BookDTO {

private Long id;

private String title;

private String author;

private String isbn;

private Double price;

}

**CustomerDTO:**

public class CustomerDTO {

private Long id;

private String name;

private String email;

private String phoneNumber;

}

**Book Mapper:**

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);

}

**Customer Mapper:**

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface CustomerMapper {

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

CustomerDTO customerToCustomerDTO(Customer customer);

Customer customerDTOToCustomer(CustomerDTO customerDTO);

}

**Configure ModelMapper:**

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();

}

}

**Mapper Usage:**

@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 CustomerDTO convertToDto(Customer customer) {

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

}

public Customer convertToEntity(CustomerDTO customerDTO) {

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

}

**Custom Price Serialization:**

import com.fasterxml.jackson.databind.annotation.JsonSerialize;

public class BookDTO {

private Long id;

private String title;

private String author;

private String isbn;

@JsonSerialize(using = CustomPriceSerializer.class)

private double price;

}

**Custom Price Deserialization:**

import com.fasterxml.jackson.databind.annotation.JsonDeserialize;

public class BookDTO {

private Long id;

private String title;

private String author;

private String isbn;

@JsonDeserialize(using = CustomPriceDeserializer.class)

private double price;

}

**Custom Price Serializer:**

import com.fasterxml.jackson.core.JsonGenerator;

import com.fasterxml.jackson.databind.SerializerProvider;

import com.fasterxml.jackson.databind.ser.std.StdSerializer;

import java.io.IOException;

public class CustomPriceSerializer extends StdSerializer<Double> {

public CustomPriceSerializer() {

this(null);

}

public CustomPriceSerializer(Class<Double> t) {

super(t);

}

@Override

public void serialize(Double value, JsonGenerator gen, SerializerProvider provider) throws IOException {

gen.writeString(String.format("%.2f USD", value));

}

}

**Exercise 2: Implementing CRUD operations**

**CRUD Endpoints:**

**BookController:**

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@PostMapping

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

return ResponseEntity.ok(bookService.createBook(bookDTO));

}

@GetMapping("/{id}")

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

return ResponseEntity.ok(bookService.getBookById(id));

}

@GetMapping

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

return ResponseEntity.ok(bookService.getAllBooks());

}

@PutMapping("/{id}")

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

return ResponseEntity.ok(bookService.updateBook(id, bookDTO));

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

**CustomerController:**

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@PostMapping

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

return ResponseEntity.ok(customerService.createCustomer(customerDTO));

}

@GetMapping("/{id}")

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

return ResponseEntity.ok(customerService.getCustomerById(id));

}

@GetMapping

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

return ResponseEntity.ok(customerService.getAllCustomers());

}

@PutMapping("/{id}")

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

return ResponseEntity.ok(customerService.updateCustomer(id, customerDTO));

}

@DeleteMapping("/{id}")

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

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

**Validating Input Data:**

**BookDTO:**

import jakarta.validation.constraints.\*;

public class BookDTO {

private Long id;

@NotNull(message = "Title is required")

@Size(min = 2, max = 100, message = "Title must be between 2 and 100 characters")

private String title;

@NotNull(message = "Author is required")

@Size(min = 2, max = 50, message = "Author must be between 2 and 50 characters")

private String author;

@NotNull(message = "ISBN is required")

@Size(min = 10, max = 13, message = "ISBN must be between 10 and 13 characters")

private String isbn;

@Min(value = 0, message = "Price must be greater than or equal to 0")

private double price;

}

**CustomerDTO:**

import jakarta.validation.constraints.\*;

public class CustomerDTO {

private Long id;

@NotNull(message = "Name is required")

@Size(min = 2, max = 50, message = "Name must be between 2 and 50 characters")

private String name;

@NotNull(message = "Email is required")

@Email(message = "Email should be valid")

private String email;

@NotNull(message = "Address is required")

@Size(min = 5, max = 100, message = "Address must be between 5 and 100 characters")

private String address;

}

**Optimistic Locking:**

**Book Entity:**

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private String isbn;

private double price;

@Version

private int version;

}

**Customer Entity:**

import jakarta.persistence.\*;

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String address;

@Version

private int version;

}

**Handling Exceptions:**

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public BookDTO updateBook(Long id, BookDTO bookDTO) {

try {

Book book = bookRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Book not found"));

book.setTitle(bookDTO.getTitle());

book.setAuthor(bookDTO.getAuthor());

book.setIsbn(bookDTO.getIsbn());

book.setPrice(bookDTO.getPrice());

book = bookRepository.save(book);

return BookMapper.INSTANCE.bookToBookDTO(book);

} catch (OptimisticLockingFailureException e) {

throw new ConcurrentUpdateException("The book was updated by another transaction");

}

}`

}

**Exercise 3: Understanding HATEOS**

**Add Spring HATEOS Dependency:**

<dependency>

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

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

</dependency>

Modify DTOs to Extend:

**BookDTO:**

import org.springframework.hateoas.RepresentationModel;

public class BookDTO extends RepresentationModel<BookDTO> {

private Long id;

private String title;

private String author;

private String isbn;

private double price;

}

**CustomerDTO:**

**Add Links in Controller or Service:**

**Book Controller:**

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

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping("/{id}")

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

BookDTO bookDTO = bookService.getBookById(id);

bookDTO.add(WebMvcLinkBuilder.linkTo(

WebMvcLinkBuilder.methodOn(BookController.class).getBook(id))

.withSelfRel());

bookDTO.add(WebMvcLinkBuilder.linkTo(

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

.withRel("all-books"));

return ResponseEntity.ok(bookDTO);

}

@GetMapping

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

List<BookDTO> books = bookService.getAllBooks();

books.forEach(book -> book.add(WebMvcLinkBuilder.linkTo(

WebMvcLinkBuilder.methodOn(BookController.class).getBook(book.getId()))

.withSelfRel()));

return ResponseEntity.ok(books);

}

}

**Explore the APIs with HATEOS:**

{

"id": 1,

"title": "Effective Java",

"author": "Joshua Bloch",

"isbn": "978-0134685991",

"price": 45.00,

"\_links": {

"self": {

"href": "http://localhost:8080/api/books/1"

},

"all-books": {

"href": "http://localhost:8080/api/books"

}

}

}

**Exercise 4: Configure Content Negotiation**

Add Dependencies for XML Support

**Configure Negotiation in Spring Boot**

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(MediaType.APPLICATION\_JSON)

.mediaType("json", MediaType.APPLICATION\_JSON)

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

}

}

**Modify Controllers to return DTOs**

**Book Controller:**

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

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

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

BookDTO bookDTO = bookService.getBookById(id);

return ResponseEntity.ok(bookDTO);

}

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

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

List<BookDTO> books = bookService.getAllBooks();

return ResponseEntity.ok(books);

}

}

**Customer Controller:**

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

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

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

CustomerDTO customerDTO = customerService.getCustomerById(id);

return ResponseEntity.ok(customerDTO);

}

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

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

List<CustomerDTO> customers = customerService.getAllCustomers();

return ResponseEntity.ok(customers);

}

}

**Exercise 5:Integrating Spring Boot Actuator**

Add Actuator Dependency

Expose actuator endpoints

Customize actuator Endpoints

**Define Custom metrices:**

import io.micrometer.core.instrument.Counter;

import io.micrometer.core.instrument.MeterRegistry;

import org.springframework.stereotype.Service;

@Service

public class CustomMetricService {

private final Counter bookCreatedCounter;

public CustomMetricService(MeterRegistry meterRegistry) {

this.bookCreatedCounter = meterRegistry.counter("custom.book.created.count");

}

public void incrementBookCreatedCounter() {

bookCreatedCounter.increment();

}

}

Use custom metrices in services

**BookService:**

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

import org.springframework.stereotype.Service;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

@Autowired

private CustomMetricService customMetricService;

public BookDTO createBook(BookDTO bookDTO) {

Book book = new Book();

book.setTitle(bookDTO.getTitle());

book.setAuthor(bookDTO.getAuthor());

book.setIsbn(bookDTO.getIsbn());

book.setPrice(bookDTO.getPrice());

bookRepository.save(book);

customMetricService.incrementBookCreatedCounter();

return BookMapper.INSTANCE.bookToBookDTO(book);

}

}

**Exercise 6:Unit Testing Rest Controllers:**

JUnit and Mockito Setup

MockMvc for Unit Testing

**Book Controller:**

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping("/{id}")

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

BookDTO bookDTO = bookService.getBookById(id);

return ResponseEntity.ok(bookDTO);

}

}

**Unit Test:**

@WebMvcTest(BookController.class)

class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private BookService bookService;

@Test

void testGetBookById() throws Exception {

Long bookId = 1L;

BookDTO mockBook = new BookDTO(bookId, "Test Title", "Test Author", 29.99);

Mockito.when(bookService.getBookById(bookId)).thenReturn(mockBook);

mockMvc.perform(get("/books/{id}", bookId))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(bookId))

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

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

.andExpect(jsonPath("$.price").value(29.99));

}

}

**Exercise 7:Integration Testing for Rest Services**

Spring Test Setup for Integration Testing

Configure the application for the testing

**MockMvc for Integrtaion Testing:**

@SpringBootTest

@AutoConfigureMockMvc

class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@BeforeEach

void setup() {

bookRepository.deleteAll();

@Test

void testCreateAndGetBook() throws Exception {

String bookJson = "{\"title\": \"Spring Boot Guide\", \"author\": \"John Doe\", \"price\": 39.99}";

mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(bookJson))

.andExpect(status().isCreated())

.andExpect(header().string("Location", containsString("/books/")));

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

.andExpect(status().isOk())

.andExpect(jsonPath("$[0].title").value("Spring Boot Guide"))

.andExpect(jsonPath("$[0].author").value("John Doe"))

.andExpect(jsonPath("$[0].price").value(39.99));

}

}

Database Integration

Run Integration Testing

Mnv test

**Excercise 8 : API Documentation with Swagger**

Add Swagger or Springdoc Dependencies

Document Endpoints

**Book Controller:**

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

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

import io.swagger.v3.oas.annotations.tags.Tag;

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

@RestController

@RequestMapping("/books")

@Tag(name = "Book API", description = "Operations pertaining to books in the bookstore")

public class BookController {

@Operation(summary = "Get a book by ID", description = "Retrieve a book from the bookstore by its ID")

@GetMapping("/{id}")

public ResponseEntity<BookDTO> getBookById(

@Parameter(description = "ID of the book to be retrieved") @PathVariable Long id) {

}

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

@PostMapping

public ResponseEntity<BookDTO> createBook(

@RequestBody BookDTO bookDTO) {

}

}

**Review Api Documentation:**

import org.springdoc.core.GroupedOpenApi;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class OpenApiConfig {

@Bean

public GroupedOpenApi publicApi() {

return GroupedOpenApi.builder()

.group("public-api")

.pathsToMatch("/")

.build();

}

}