**WEEK-5 Exercises**

**Exercise 8: Online Bookstore - Implementing CRUD Operations**

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.persistence.\*;

import javax.validation.constraints.Email;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

import java.util.List;

@Entity

class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Version

private Long version;

@NotNull

@Size(min = 1, max = 100)

private String title;

@NotNull

@Min(0)

private Double price;

// Getters and Setters

}

@Entity

class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 100)

private String name;

@NotNull

@Email

private String email;

// Getters and Setters

}

interface BookRepository extends JpaRepository<Book, Long> {}

interface CustomerRepository extends JpaRepository<Customer, Long> {}

@RestController

@RequestMapping("/books")

class BookController {

@Autowired

private BookRepository bookRepository;

@PostMapping

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

return new ResponseEntity<>(bookRepository.save(book), HttpStatus.CREATED);

}

@GetMapping("/{id}")

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

return bookRepository.findById(id).map(ResponseEntity::ok)

.orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

@PutMapping("/{id}")

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

book.setId(id);

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

}

@DeleteMapping("/{id}")

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

bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}

**Exercise 9: Online Bookstore - Understanding HATEOAS**

import org.springframework.hateoas.EntityModel;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.\*;

@RestController

@RequestMapping("/books")

class HATEOASBookController {

@Autowired

private BookRepository bookRepository;

@GetMapping("/{id}")

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

Book book = bookRepository.findById(id).orElseThrow();

EntityModel<Book> resource = EntityModel.of(book);

resource.add(linkTo(methodOn(HATEOASBookController.class).getBook(id)).withSelfRel());

return resource;

}

}

**Exercise 10: Online Bookstore - Configuring Content Negotiation**

import org.springframework.http.MediaType;

@RestController

@RequestMapping("/books")

class ContentNegotiationController {

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

public List<Book> getBooks() {

return bookRepository.findAll();

}

}

**Exercise 11: Online Bookstore - Integrating Spring Boot Actuator**

import org.springframework.boot.actuate.autoconfigure.web.servlet.endpoint.web.WebEndpointProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

class ActuatorConfiguration {

@Bean

public WebEndpointProperties webEndpointProperties() {

return new WebEndpointProperties();

}

}

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

import org.springframework.context.annotation.Bean;

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

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

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

@EnableWebSecurity

class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.anyRequest().authenticated()

.and().httpBasic();

}

}

**Exercise 13: Online Bookstore - Unit Testing REST Controllers**

import org.junit.jupiter.api.Test;

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

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

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

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

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@SpringBootTest

@AutoConfigureMockMvc

class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@Test

void testGetBooks() throws Exception {

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

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

}

}

**Exercise 14: Online Bookstore - Integration Testing for REST Services**

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

@SpringBootTest

class BookIntegrationTest {

}

**Exercise 15: Online Bookstore - API Documentation with Swagger**

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

import io.swagger.v3.oas.annotations.info.Info;

@OpenAPIDefinition(info = @Info(title = "Online Bookstore API", version = "1.0"))

class SwaggerConfig {

}