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

Business Scenario:

Implement Create, Read, Update, and Delete opera ons for the Book and Customer en es.

CODE:

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

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

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Service;

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

import java.util.List;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = Genera onType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

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

@Min(0)

private double price;

@Version

private int version;

}

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

@Size(min = 5, max = 100) private String email;

@Version

private int version;

}

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

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

@Service public class BookService { private final BookRepository bookRepository;

public BookService(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

public List<Book> getAllBooks() { return bookRepository.findAll();

}

public Book getBookById(Long id) { return bookRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundException("Book not found"));

}

@Transactional

public Book createBook(Book book) { return bookRepository.save(book);

}

@Transactional

public Book updateBook(Long id, Book bookDetails) { Book book = getBookById(id); book.setTitle(bookDetails.getTitle()); book.setAuthor(bookDetails.getAuthor()); book.setPrice(bookDetails.getPrice());

return bookRepository.save(book);

}

@Transactional

public void deleteBook(Long id) { Book book = getBookById(id); bookRepository.delete(book);

}

}

@Service

public class CustomerService { private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) { this.customerRepository = customerRepository;

}

public List<Customer> getAllCustomers() { return customerRepository.findAll();

}

public Customer getCustomerById(Long id) { return customerRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundExcep on("Customer not found"));

}

@Transactional

public Customer createCustomer(Customer customer) { return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) { Customer customer = getCustomerById(id); customer.setName(customerDetails.getName()); customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) { Customer customer = getCustomerById(id); customerRepository.delete(customer);

}

}

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

public BookController(BookService bookService) { this.bookService = bookService;

}

@GetMapping

public List<Book> getAllBooks() { return bookService.getAllBooks();

}

@GetMapping("/{id}")

public ResponseEn ty<Book> getBookById(@PathVariable Long id){ return ResponseEn ty.ok(bookService.getBookById(id));

}

@PostMapping

public ResponseEn ty<Book> createBook(@Valid @RequestBody Book book) { return new ResponseEn ty<>(bookService.createBook(book), H pStatus.CREATED);

}

@PutMapping("/{id}")

public ResponseEn ty<Book> updateBook(@PathVariable Long id, @Valid @RequestBody Book bookDetails) {

return ResponseEn ty.ok(bookService.updateBook(id, bookDetails));

}

@DeleteMapping("/{id}")

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

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

}

}

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService;

public CustomerController(CustomerService customerService) { this.customerService = customerService;

}

@GetMapping

public List<Customer> getAllCustomers() { return customerService.getAllCustomers();

}

@GetMapping("/{id}")

public ResponseEn ty<Customer> getCustomerById(@PathVariable Long id) { return ResponseEn ty.ok(customerService.getCustomerById(id));

}

@PostMapping

public ResponseEn ty<Customer> createCustomer(@Valid @RequestBody Customer customer) {

return new ResponseEn ty<>(customerService.createCustomer(customer), HttpStatus.CREATED);

}

@PutMapping("/{id}")

public ResponseEn ty<Customer> updateCustomer(@PathVariable Long id, @Valid

@RequestBody Customer customerDetails) {

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

}

@DeleteMapping("/{id}")

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

return ResponseEntity.noContent().build();

}

}

@ResponseStatus(value = HttpStatus.NOT\_FOUND) public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) { super(message);

}

}

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

Business Scenario:

Enhance your REST API to follow HATEOAS principles for naviga on through resources.

CODE:

import jakarta.persistence.\*;

import jakarta.valida on.constraints.\*;

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

import org.springframework.hateoas.Link;

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

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Service;

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

import java.util.List;

import java.util.stream.Collectors;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

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

@Min(0)

private double price;

@Version private int version;

}

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

@Size(min = 5, max = 100) private String email;

@Version private int version;

}

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

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

@Service

public class BookService {

private final BookRepository bookRepository;

public BookService(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

public List<Book> getAllBooks() { return bookRepository.findAll();

}

public Book getBookById(Long id) { return bookRepository.findById(id)

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

}

@Transactional

public Book createBook(Book book) { return bookRepository.save(book);

}

@Transactional

public Book updateBook(Long id, Book bookDetails) { Book book = getBookById(id); book.setTitle(bookDetails.getTitle()); book.setAuthor(bookDetails.getAuthor()); book.setPrice(bookDetails.getPrice());

return bookRepository.save(book);

}

@Transactional

public void deleteBook(Long id) { Book book = getBookById(id); bookRepository.delete(book);

}

}

@Service

public class CustomerService {

private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) { this.customerRepository = customerRepository;

}

public List<Customer> getAllCustomers() { return customerRepository.findAll();

}

public Customer getCustomerById(Long id) { return customerRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundExcep on("Customer not found"));

}

@Transactional

public Customer createCustomer(Customer customer) { return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) { Customer customer = getCustomerById(id); customer.setName(customerDetails.getName()); customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) { Customer customer = getCustomerById(id); customerRepository.delete(customer);

}

}

@Component

public class ModelAssembler { public Link createBookLink(Long id) {

return

WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).getBookById(id)).wit hSelfRel();

}

public Link createCustomerLink(Long id) {

return

WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(CustomerController.class).getCustomerByI d(id)).withSelfRel();

}

}

@RestController

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

private final BookService bookService;

private final ModelAssembler modelAssembler;

public BookController(BookService bookService, ModelAssembler modelAssembler) {

this.bookService = bookService; this.modelAssembler = modelAssembler; }

@GetMapping

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

return bookService.getAllBooks().stream()

.map(book -> En tyModel.of(book, modelAssembler.createBookLink(book.getId())))

.collect(Collectors.toList());

}

@GetMapping("/{id}")

public ResponseEn ty<En tyModel<Book>> getBookById(@PathVariable Long id) {

Book book = bookService.getBookById(id);

En tyModel<Book> resource = EntityModel.of(book, modelAssembler.createBookLink(id)); return ResponseEn ty.ok(resource);

}

@PostMapping

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

Book createdBook = bookService.createBook(book);

EntityModel<Book> resource = EntityModel.of(createdBook, modelAssembler.createBookLink(createdBook.getId())); return new ResponseEntity<>(resource, H pStatus.CREATED);

}

@PutMapping("/{id}")

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

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

EntityModel<Book> resource = EntityModel.of(updatedBook, modelAssembler.createBookLink(id));

return ResponseEntity.ok(resource);

}

@DeleteMapping("/{id}")

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

return ResponseEntity.noContent().build();

}

}

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService; private final ModelAssembler modelAssembler;

public CustomerController(CustomerService customerService, ModelAssembler modelAssembler) {

this.customerService = customerService;

this.modelAssembler = modelAssembler;

}

@GetMapping

public List<EntityModel<Customer>> getAllCustomers() { return customerService.getAllCustomers().stream()

.map(customer -> En tyModel.of(customer, modelAssembler.createCustomerLink(customer.getId())))

.collect(Collectors.toList());

}

@GetMapping("/{id}")

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

Customer customer = customerService.getCustomerById(id);

EntityModel<Customer> resource = EntityModel.of(customer, modelAssembler.createCustomerLink(id));

return ResponseEntity.ok(resource);

}

@PostMapping

public ResponseEntity<EntityModel<Customer>> createCustomer(@Valid @RequestBody Customer customer) {

Customer createdCustomer = customerService.createCustomer(customer);

En tyModel<Customer> resource = EntityModel.of(createdCustomer, modelAssembler.createCustomerLink(createdCustomer.getId())); return new ResponseEntity<>(resource, HttpStatus.CREATED);

}

@PutMapping("/{id}")

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

Customer updatedCustomer = customerService.updateCustomer(id, customerDetails);

EntityModel<Customer> resource = EntityModel.of(updatedCustomer, modelAssembler.createCustomerLink(id));

return ResponseEntity.ok(resource);

}

@DeleteMapping("/{id}")

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

return ResponseEntity.noContent().build();

}

}

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundExcep on extends RuntimeException { public ResourceNotFoundException(String message) { super(message);

}

}

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

Business Scenario:

Support different media types (JSON, XML) for your bookstore's RESTful services.

CODE:

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

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

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

import org.springframework.h p.converter.xml.MappingJackson2XmlH pMessageConverter; import org.springframework.context.annotation.Bean;

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

import java.util.List;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

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

@Min(0)

private double price;

@Version private int version;

}

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

@Size(min = 5, max = 100) private String email;

@Version private int version;

}

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

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

@Service public class BookService { private final BookRepository bookRepository;

public BookService(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

public List<Book> getAllBooks() { return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

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

}

@Transactional

public Book createBook(Book book) { return bookRepository.save(book);

}

@Transactional

public Book updateBook(Long id, Book bookDetails) { Book book = getBookById(id); book.setTitle(bookDetails.getTitle()); book.setAuthor(bookDetails.getAuthor()); book.setPrice(bookDetails.getPrice());

return bookRepository.save(book);

}

@Transactional

public void deleteBook(Long id) { Book book = getBookById(id); bookRepository.delete(book);

}

}

@Service public class CustomerService {

private final CustomerRepository customerRepository; public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public List<Customer> getAllCustomers() { return customerRepository.findAll();

}

public Customer getCustomerById(Long id) { return customerRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundExcep on("Customer not found"));

}

@Transactional

public Customer createCustomer(Customer customer) { return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) { Customer customer = getCustomerById(id); customer.setName(customerDetails.getName()); customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) { Customer customer = getCustomerById(id); customerRepository.delete(customer);

}

}

@RestController

@RequestMapping("/api/books")

public class BookController { private final BookService bookService;

public BookController(BookService bookService) { this.bookService = bookService;

}

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

public List<Book> getAllBooks() { return bookService.getAllBooks();

}

@GetMapping(value = "/{id}",

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

public ResponseEntity<Book> getBookById(@PathVariable Long id) { Book book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

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

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

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

Book createdBook = bookService.createBook(book);

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

}

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

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

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

Book updatedBook = bookService.updateBook(id, bookDetails); return ResponseEntity.ok(updatedBook);

}

@DeleteMapping(value = "/{id}",

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

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

return ResponseEntity.noContent().build();

}

}

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService;

public CustomerController(CustomerService customerService) { this.customerService = customerService;

}

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

public List<Customer> getAllCustomers() { return customerService.getAllCustomers();

}

@GetMapping(value = "/{id}",

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

public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) { Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customer);

}

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

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

public ResponseEn ty<Customer> createCustomer(@RequestBody Customer customer) { Customer createdCustomer = customerService.createCustomer(customer);

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

}

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

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

public ResponseEntity<Customer> updateCustomer(@PathVariable Long id, @RequestBody Customer customerDetails) {

Customer updatedCustomer = customerService.updateCustomer(id, customerDetails); return ResponseEntity.ok(updatedCustomer);

}

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

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

return ResponseEntity.noContent().build();

}

}

@Configuration public class WebConfig implements WebMvcConfigurer {

@Override public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {

configurer.favorPathExtension(false) .favorParameter(false)

.ignoreAcceptHeader(false)

.defaultContentType(MediaType.APPLICATION\_JSON)

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

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

}

@Bean public MappingJackson2XmlH pMessageConverter xmlConverter() {

return new MappingJackson2XmlH pMessageConverter();

}

}

@ResponseStatus(value = HttpStatus.NOT\_FOUND) public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) { super(message);

}

}

**Exercise 11: Online Bookstore - Integra ng Spring Boot Actuator**

Business Scenario:

Monitor and manage your bookstore's RESTful services using Spring Boot Actuator.

CODE:

<!-- pom.xml -->

<dependencies>

<dependency>

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

<ar factId>spring-boot-starter-actuator</ar factId>

</dependency> </dependencies> import io.micrometer.core.instrument.Counter;

import io.micrometer.core.instrument.MeterRegistry;

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

import org.springframework.boot.actuate.endpoint.web.annotation.RestControllerEndpoint; import org.springframework.boot.actuate.health.Health;

import org.springframework.boot.actuate.health.HealthIndicator;

import org.springframework.boot.actuate.info.Info;

import org.springframework.boot.actuate.info.InfoContributor;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Service;

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

import java.util.HashMap; import java.util.List; import java.util.Map;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = Genera onType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

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

@Min(0)

private double price;

@Version private int version;

}

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = Genera onType.IDENTITY)

private Long id;

@NotNull

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

@NotNull

@Size(min = 5, max = 100) private String email;

@Version private int version;

}

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

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

@Service public class BookService { private final BookRepository bookRepository; private final Counter bookCrea onCounter;

public BookService(BookRepository bookRepository, MeterRegistry meterRegistry) { this.bookRepository = bookRepository;

this.bookCrea onCounter = meterRegistry.counter("book.crea on.count");

}

public List<Book> getAllBooks() { return bookRepository.findAll();

}

public Book getBookById(Long id) { return bookRepository.findById(id)

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

}

@Transactional

public Book createBook(Book book) { bookCreationCounter.increment(); return bookRepository.save(book);

}

@Transactional

public Book updateBook(Long id, Book bookDetails) { Book book = getBookById(id); book.setTitle(bookDetails.getTitle()); book.setAuthor(bookDetails.getAuthor()); book.setPrice(bookDetails.getPrice());

return bookRepository.save(book);

}

@Transactional

public void deleteBook(Long id) { Book book = getBookById(id); bookRepository.delete(book);

}

}

@Service public class CustomerService {

private final CustomerRepository customerRepository; private final Counter customerCreationCounter;

public CustomerService(CustomerRepository customerRepository, MeterRegistry meterRegistry) { this.customerRepository = customerRepository;

this.customerCrea onCounter = meterRegistry.counter("customer.crea on.count");

}

public List<Customer> getAllCustomers() { return customerRepository.findAll();

}

public Customer getCustomerById(Long id) { return customerRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundExcep on("Customer not found"));

}

@Transactional

public Customer createCustomer(Customer customer) { customerCreationCounter.increment();

return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) { Customer customer = getCustomerById(id); customer.setName(customerDetails.getName()); customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) { Customer customer = getCustomerById(id); customerRepository.delete(customer);

}

}

@Component

public class BookstoreHealthIndicator implements HealthIndicator {

@Override public Health health()

return Health.up().withDetail("status", "Everything is OK!").build();

}

}

@Component

public class BookstoreInfoContributor implements InfoContributor {

@Override public void contribute(Info.Builder builder) {

Map<String, Object> bookstoreDetails = new HashMap<>(); bookstoreDetails.put("applica on", "Online Bookstore"); bookstoreDetails.put("version", "1.0.0");

builder.withDetail("bookstore-info", bookstoreDetails);

}

}

@RestControllerEndpoint(id = "custom-endpoint") public class CustomActuatorEndpoint {

@GetMapping("/status")

public ResponseEntity<String> getStatus() {

return ResponseEntity.ok("Custom Actuator Endpoint is working!");

}

}

@Configuration

public class ActuatorConfig {

@Bean

public MeterRegistryCustomizer<MeterRegistry> configureMetrics() {

return registry -> registry.config().commonTags("application", "Online Bookstore");

}

}

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

public BookController(BookService bookService) { this.bookService = bookService;

}

@GetMapping

public List<Book> getAllBooks() { return bookService.getAllBooks();

}

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id) { Book book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

@PostMapping

public ResponseEntity<Book> createBook(@RequestBody Book book) { Book createdBook = bookService.createBook(book);

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

}

@PutMapping("/{id}")

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

Book updatedBook = bookService.updateBook(id, bookDetails); return ResponseEntity.ok(updatedBook);

}

@DeleteMapping("/{id}")

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

return ResponseEntity.noContent().build();

}

}

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService;

public CustomerController(CustomerService customerService) { this.customerService = customerService;

}

@GetMapping

public List<Customer> getAllCustomers() { return customerService.getAllCustomers();

}

@GetMapping("/{id}")

public ResponseEn ty<Customer> getCustomerById(@PathVariable Long id) { Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customer);

}

@PostMapping

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) { Customer createdCustomer = customerService.createCustomer(customer);

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

}

@PutMapping("/{id}")

public ResponseEntity<Customer> updateCustomer(@PathVariable Long id, @RequestBody Customer customerDetails) {

Customer updatedCustomer = customerService.updateCustomer(id, customerDetails); return ResponseEntity.ok(updatedCustomer);

}

@DeleteMapping("/{id}")

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

return ResponseEntity.noContent().build();

}

}

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) { super(message);

}

}

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

Business Scenario:

Secure your bookstore's RESTful endpoints using Spring Security with JWT-based authen ca on.

CODE:

<!-- pom.xml -->

<dependencies>

<dependency>

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

<ar factId>spring-boot-starter-security</ar factId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<ar factId>jjwt-api</ar factId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<ar factId>jjwt-impl</ar factId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<ar factId>jjwt-jackson</ar factId> <!-- or jjwt-gson, jjwt-orgjson, etc -->

<version>0.11.5</version>

</dependency>

</dependencies>

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configura on;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

import org.springframework.security.config.annotation.web.builders.H pSecurity;

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

import org.springframework.security.core.Authentication;

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

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

import org.springframework.security.core.userdetails.UsernameNotFoundExcep on;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

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

import org.springframework.web.filter.OncePerRequestFilter;

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

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

import java.io.IOException; import java.security.Key; import java.util.Date; import java.util.List;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebMvcConfigurer {

private final JwtTokenProvider jwtTokenProvider;

private final CustomUserDetailsService customUserDetailsService;

public SecurityConfig(JwtTokenProvider jwtTokenProvider, CustomUserDetailsService customUserDetailsService) {

this.jwtTokenProvider = jwtTokenProvider;

this.customUserDetailsService = customUserDetailsService;

}

@Override

protected void configure(H pSecurity h p) throws Excep on {

http.csrf().disable()

.cors().and()

.authorizeRequests()

.antMatchers("/api/auth/\*\*").permitAll()

.anyRequest().authen cated()

.and()

.sessionManagement().sessionCrea onPolicy(SessionCrea onPolicy.STATELESS)

.and()

.addFilterBefore(new JwtAuthenticationFilter(jwtTokenProvider, customUserDetailsService),

UsernamePasswordAuthenticationFilter.class);

}

@Override

public void addCorsMappings(CorsRegistry registry) { registry.addMapping("/\*\*")

.allowedOrigins("h p://allowed-origin.com")

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

.allowedHeaders("\*")

.allowCreden als(true);

}

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception { return super.authenticationManagerBean();

}

@Bean

public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder();

}

}

@Service public class JwtTokenProvider {

private final Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256); private final long validityInMilliseconds = 3600000;

public String createToken(String username, List<String> roles) { Claims claims = Jwts.claims().setSubject(username); claims.put("roles", roles);

Date now = new Date();

Date validity = new Date(now.getTime() + validityInMilliseconds);

return Jwts.builder()

.setClaims(claims) .setIssuedAt(now)

.setExpira on(validity)

.signWith(key)

.compact();

}

public boolean validateToken(String token) {

try {

Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);

return true;

} catch (Exception e) { return false;

}

}

public String getUsername(String token) {

return

Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token).getBody().getSubject();

}

}

public class JwtAuthenticationFilter extends OncePerRequestFilter {

private final JwtTokenProvider jwtTokenProvider;

private final CustomUserDetailsService customUserDetailsService;

public JwtAuthenticationFilter(JwtTokenProvider jwtTokenProvider, CustomUserDetailsService customUserDetailsService) {

this.jwtTokenProvider = jwtTokenProvider;

this.customUserDetailsService = customUserDetailsService;

}

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletExcep on, IOException { String token = resolveToken(request);

if (token != null && jwtTokenProvider.validateToken(token)) {

String username = jwtTokenProvider.getUsername(token);

UserDetails userDetails = customUserDetailsService.loadUserByUsername(username);

UsernamePasswordAuthenticationToken auth = new

UsernamePasswordAuthenticationToken(userDetails, "", userDetails.getAuthorities());

auth.setDetails(userDetails);

}

filterChain.doFilter(request, response);

}

private String resolveToken(HttpServletRequest request) { String bearerToken = request.getHeader("Authorization");

if (bearerToken != null && bearerToken.startsWith("Bearer ")) { return bearerToken.substring(7);

}

return null;

}

}

@Service public class CustomUserDetailsService implements UserDetailsService {

@Override public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

// For simplicity, using hardcoded users. In a real applica on, fetch from DB. if ("admin".equals(username)) {

return User.withUsername(username)

.password(passwordEncoder().encode("admin123"))

.roles("ADMIN")

.build();

} else if ("user".equals(username)) { return User.withUsername(username)

.password(passwordEncoder().encode("user123"))

.roles("USER")

.build();

} else {

throw new UsernameNotFoundExcep on("User not found");

}

}

@Bean

public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder();

}

}

@RestController

@RequestMapping("/api/auth")

public class AuthController {

private final JwtTokenProvider jwtTokenProvider;

public AuthController(JwtTokenProvider jwtTokenProvider) { this.jwtTokenProvider = jwtTokenProvider;

}

@PostMapping("/login") public ResponseEn ty<?> login(@RequestBody AuthRequest authRequest) {

if ("admin".equals(authRequest.getUsername()) && "admin123".equals(authRequest.getPassword())) {

String token = jwtTokenProvider.createToken(authRequest.getUsername(), List.of("ROLE\_ADMIN")); return ResponseEn ty.ok(new AuthResponse(token));

} else if ("user".equals(authRequest.getUsername()) &&

"user123".equals(authRequest.getPassword())) {

String token = jwtTokenProvider.createToken(authRequest.getUsername(), List.of("ROLE\_USER"));

return ResponseEntity.ok(new AuthResponse(token));

} else {

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

}

}

}

]

public class AuthRequest { private String username; private String password;

}

public class AuthResponse { private String token;

public AuthResponse(String token) { this.token = token;

}

}

@RestController

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

}

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

}

**Exercise 13: Online Bookstore - Unit Tes ng REST Controllers**

Business Scenario:

Write unit tests for your bookstore's REST controllers using JUnit and Mockito.

CODE:

<!-- pom.xml -->

<dependencies>

<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>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<ar factId>mockito-junit-jupiter</ar factId>

<scope>test</scope>

</dependency>

</dependencies>

// BookControllerTest.java

import com.example.bookstore.controller.BookController; import com.example.bookstore.model.Book; import com.example.bookstore.service.BookService; import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import org.junit.jupiter.api.extension.ExtendWith; import org.mockito.InjectMocks;

import org.mockito.Mock; import org.mockito.junit.jupiter.MockitoExtension; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest; import org.springframework.http.MediaType; import org.springframework.test.web.servlet.MockMvc; import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import java.util.Arrays; import java.util.List;

import static org.mockito.Mockito.\*;

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

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

@WebMvcTest(BookController.class) @ExtendWith(MockitoExtension.class) public class BookControllerTest {

@Autowired private MockMvc mockMvc;

@Mock private BookService bookService;

@InjectMocks private BookController bookController;

private Book book1; private Book book2; @BeforeEach public void setUp() { mockMvc = MockMvcBuilders.standaloneSetup(bookController).build();

book1 = new Book(); book1.setId(1L); book1.setTitle("Book One"); book1.setAuthor("Author One"); book1.setPrice(10.0);

book2 = new Book(); book2.setId(2L); book2.setTitle("Book Two"); book2.setAuthor("Author Two"); book2.setPrice(15.0);

}

@Test public void testGetAllBooks() throws Excep on { List<Book> books = Arrays.asList(book1, book2); when(bookService.getAllBooks()).thenReturn(books);

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

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(2))

.andExpect(jsonPath("$[0]. tle").value("Book One"))

.andExpect(jsonPath("$[1]. tle").value("Book Two"));

verify(bookService, mes(1)).getAllBooks();

}

@Test

public void testGetBookById() throws Excep on { when(bookService.getBookById(1L)).thenReturn(book1);

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

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$. tle").value("Book One"));

verify(bookService, mes(1)).getBookById(1L);

}

@Test public void testCreateBook() throws Exception { when(bookService.createBook(any(Book.class))).thenReturn(book1);

mockMvc.perform(post("/api/books")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\" tle\": \"Book One\", \"author\": \"Author One\", \"price\": 10.0}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$. tle").value("Book One"));

verify(bookService, mes(1)).createBook(any(Book.class));

}

@Test

public void testUpdateBook() throws Exception { when(bookService.updateBook(eq(1L), any(Book.class))).thenReturn(book1);

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

.contentType(MediaType.APPLICATION\_JSON)

.content("{\" tle\": \"Updated Book\", \"author\": \"Updated Author\", \"price\": 20.0}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$. tle").value("Book One")); // since it mocks the original

verify(bookService, mes(1)).updateBook(eq(1L), any(Book.class));

}

@Test

public void testDeleteBook() throws Exception { doNothing().when(bookService).deleteBook(1L);

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

.contentType(MediaType.APPLICATION\_JSON))

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

verify(bookService, mes(1)).deleteBook(1L);

}

}

import com.example.bookstore.controller.CustomerController; import com.example.bookstore.model.Customer; import com.example.bookstore.service.CustomerService; import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import org.junit.jupiter.api.extension.ExtendWith; import org.mockito.InjectMocks; import org.mockito.Mock; import org.mockito.junit.jupiter.MockitoExtension;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest; import org.springframework.http.MediaType; import org.springframework.test.web.servlet.MockMvc; import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import java.uttl.Arrays; import java.uttl.List;

import static org.mockito.Mockito.\*; import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(CustomerController.class) @ExtendWith(MockitoExtension.class) public class CustomerControllerTest {

@Autowired

private MockMvc mockMvc;

@Mock

private CustomerService customerService;

@InjectMocks

private CustomerController customerController;

private Customer customer1; private Customer customer2;

@BeforeEach public void setUp() { mockMvc = MockMvcBuilders.standaloneSetup(customerController).build();

customer1 = new Customer(); customer1.setId(1L); customer1.setName("Customer One"); customer1.setEmail("customer1@example.com");

customer2 = new Customer(); customer2.setId(2L); customer2.setName("Customer Two"); customer2.setEmail("customer2@example.com");

}

@Test public void testGetAllCustomers() throws Excep on {

List<Customer> customers = Arrays.asList(customer1, customer2); when(customerService.getAllCustomers()).thenReturn(customers);

mockMvc.perform(get("/api/customers")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(2))

.andExpect(jsonPath("$[0].name").value("Customer One"))

.andExpect(jsonPath("$[1].name").value("Customer Two"));

verify(customerService, mes(1)).getAllCustomers();

}

@Test

public void testGetCustomerById() throws Exception { when(customerService.getCustomerById(1L)).thenReturn(customer1); mockMvc.perform(get("/api/customers/1")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Customer One"));

verify(customerService, mes(1)).getCustomerById(1L);

}

@Test

public void testCreateCustomer() throws Exception { when(customerService.createCustomer(any(Customer.class))).thenReturn(customer1);

mockMvc.perform(post("/api/customers")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"Customer One\", \"email\": \"customer1@example.com\"}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.name").value("Customer One"));

verify(customerService, mes(1)).createCustomer(any(Customer.class));

}

@Test

public void testUpdateCustomer() throws Exception { when(customerService.updateCustomer(eq(1L), any(Customer.class))).thenReturn(customer1);

mockMvc.perform(put("/api/customers/1")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"Updated Customer\", \"email\": \"updated@example.com\"}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Customer One")); // since it mocks the original verify(customerService, mes(1)).updateCustomer(eq(1L), any(Customer.class));

}

@Test

public void testDeleteCustomer() throws Exception { doNothing().when(customerService).deleteCustomer(1L);

mockMvc.perform(delete("/api/customers/1")

.contentType(MediaType.APPLICATION\_JSON))

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

verify(customerService, mes(1)).deleteCustomer(1L);

}

}

**Exercise 14: Online Bookstore - Integra on Tes ng for REST Services**

Business Scenario:

Write integra on tests for your bookstore's RESTful services.

CODE:

<!-- pom.xml -->

<dependencies>

<!-- Other dependencies -->

<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>

</dependencies>

// Integra onTests.java

import com.example.bookstore.BookstoreApplication; import com.example.bookstore.model.Book; import com.example.bookstore.model.Customer; import com.example.bookstore.repository.BookRepository; import com.example.bookstore.repository.CustomerRepository; import org.junit.jupiter.api.BeforeEach; import org.junit.jupiter.api.Test; import org.junit.jupiter.api.extension.ExtendWith; 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.http.MediaType; import org.springframework.test.context.junit.jupiter.SpringExtension; import org.springframework.test.web.servlet.MockMvc; import org.springframework.transaction.annotation.Transactional;

import java.util.Op onal;

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

@SpringBootTest(classes = BookstoreApplication.class)

@ExtendWith(SpringExtension.class)

@AutoConfigureMockMvc @Transactional

public class IntegrationTests {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@Autowired

private CustomerRepository customerRepository;

private Book book1; private Customer customer1;

@BeforeEach

public void setUp() {

book1 = new Book(); book1.setTitle("Integra on Test Book"); book1.setAuthor("Test Author"); book1.setPrice(29.99); bookRepository.save(book1);

customer1 = new Customer(); customer1.setName("Integra on Test Customer"); customer1.setEmail("testcustomer@example.com"); customerRepository.save(customer1);

}

@Test

public void testGetAllBooks() throws Exception { mockMvc.perform(get("/api/books")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(1))

.andExpect(jsonPath("$[0]. tle").value("Integra on Test Book"));

}

@Test

public void testGetBookById() throws Exception { mockMvc.perform(get("/api/books/" + book1.getId())

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$. tle").value("Integra on Test Book"));

}

@Test

public void testCreateBook() throws Exception { mockMvc.perform(post("/api/books")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\" tle\": \"New Book\", \"author\": \"New Author\", \"price\": 19.99}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$. tle").value("New Book"));

}

@Test

public void testUpdateBook() throws Exception { mockMvc.perform(put("/api/books/" + book1.getId())

.contentType(MediaType.APPLICATION\_JSON)

.content("{\" tle\": \"Updated Book\", \"author\": \"Updated Author\", \"price\": 39.99}")) .andExpect(status().isOk())

.andExpect(jsonPath("$. tle").value("Updated Book"));

}

@Test

public void testDeleteBook() throws Exception { mockMvc.perform(delete("/api/books/" + book1.getId())

.contentType(MediaType.APPLICATION\_JSON))

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

Op onal<Book> deletedBook = bookRepository.findById(book1.getId()); assert(deletedBook.isEmpty());

}

@Test

public void testGetAllCustomers() throws Exception { mockMvc.perform(get("/api/customers")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(1))

.andExpect(jsonPath("$[0].name").value("Integra on Test Customer"));

}

@Test

public void testGetCustomerById() throws Exception { mockMvc.perform(get("/api/customers/" + customer1.getId())

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Integra on Test Customer"));

}

@Test

public void testCreateCustomer() throws Exception { mockMvc.perform(post("/api/customers")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"New Customer\", \"email\": \"newcustomer@example.com\"}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.name").value("New Customer"));

}

@Test

public void testUpdateCustomer() throws Exception { mockMvc.perform(put("/api/customers/" + customer1.getId())

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"Updated Customer\", \"email\": \"updatedcustomer@example.com\"}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Updated Customer"));

}

@Test

public void testDeleteCustomer() throws Exception { mockMvc.perform(delete("/api/customers/" + customer1.getId())

.contentType(MediaType.APPLICATION\_JSON))

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

Op onal<Customer> deletedCustomer = customerRepository.findById(customer1.getId()); assert(deletedCustomer.isEmpty());

}

}

**Scenario 15: Online Bookstore - API Documenta on with Swagger**

Business Scenario:

Document your bookstore's REST APIs using Swagger and Springdoc.

CODE:

package com.example.bookstore;

import com.example.bookstore.model.Book; import com.example.bookstore.model.Customer; import com.example.bookstore.repository.BookRepository; import com.example.bookstore.repository.CustomerRepository; import org.springframework.beans.factory.annota on.Autowired; import org.springframework.boot.SpringApplica on; import org.springframework.boot.autoconfigure.SpringBootApplica on; import org.springframework.context.annota on.Bean; import org.springframework.web.bind.annota on.\*; import io.swagger.v3.oas.annota ons.Opera on; import io.swagger.v3.oas.annota ons.responses.ApiResponse; import io.swagger.v3.oas.annota ons.responses.ApiResponses; import io.swagger.v3.oas.annota ons.parameters.RequestBody as SwaggerRequestBody; import org.springframework.h p.H pStatus; import org.springframework.h p.ResponseEn ty; import org.springframework.web.servlet.config.annota on.WebMvcConfigurer; import org.springframework.web.servlet.config.annota on.ResourceHandlerRegistry; import org.springframework.web.servlet.config.annota on.CorsRegistry; import org.springframework.web.servlet.config.annota on.WebMvcConfigurerAdapter; import org.springdoc.core.GroupedOpenApi; import org.springdoc.webmvc.ui.SwaggerConfig;

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

import java.u l.Op onal;

@SpringBootApplica on public class BookstoreApplica on {

public sta c void main(String[] args) {

SpringApplica on.run(BookstoreApplica on.class, args);

}

@Bean public WebMvcConfigurer configurer() { return new WebMvcConfigurerAdapter() {

@Override public void addResourceHandlers(ResourceHandlerRegistry registry) { registry.addResourceHandler("swagger-ui.html")

.addResourceLoca ons("classpath:/META-INF/resources/"); registry.addResourceHandler("/webjars/\*\*")

.addResourceLoca ons("classpath:/META-INF/resources/webjars/");

}

};

}

}

@RestController

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

@Autowired private BookRepository bookRepository;

@Opera on(summary = "Get all books", descrip on = "Retrieve a list of all books") @GetMapping public List<Book> getAllBooks() { return bookRepository.findAll();

}

@Opera on(summary = "Get book by ID", descrip on = "Retrieve a book by its ID")

@ApiResponses({

@ApiResponse(responseCode = "200", descrip on = "Book found"),

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

})

@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());

}

@Opera on(summary = "Create a new book", descrip on = "Add a new book to the catalog")

@PostMapping

public ResponseEn ty<Book> createBook(@Valid @SwaggerRequestBody(descrip on = "Book object to be created") @RequestBody Book book) { Book savedBook = bookRepository.save(book); return ResponseEn ty.status(H pStatus.CREATED).body(savedBook);

}

@Opera on(summary = "Update an exis ng book", descrip on = "Update the details of an exis ng book")

@PutMapping("/{id}")

public ResponseEn ty<Book> updateBook(@PathVariable Long id, @Valid @RequestBody Book book) { if (!bookRepository.existsById(id)) { return ResponseEn ty.notFound().build();

}

book.setId(id);

Book updatedBook = bookRepository.save(book); return ResponseEn ty.ok(updatedBook);

}

@Opera on(summary = "Delete a book", descrip on = "Remove a book from the catalog")

@DeleteMapping("/{id}") public ResponseEn ty<Void> deleteBook(@PathVariable Long id) { if (!bookRepository.existsById(id)) { return ResponseEn ty.notFound().build();

}

bookRepository.deleteById(id); return ResponseEn ty.noContent().build();

}

}

@RestController

@RequestMapping("/api/customers") class CustomerController {

@Autowired private CustomerRepository customerRepository;

@Opera on(summary = "Get all customers", descrip on = "Retrieve a list of all customers")

@GetMapping public List<Customer> getAllCustomers() { return customerRepository.findAll();

}

@Opera on(summary = "Get customer by ID", descrip on = "Retrieve a customer by its ID")

@ApiResponses({

@ApiResponse(responseCode = "200", descrip on = "Customer found"),

@ApiResponse(responseCode = "404", descrip on = "Customer not found")

})

@GetMapping("/{id}") public ResponseEn ty<Customer> getCustomerById(@PathVariable Long id) { Op onal<Customer> customer = customerRepository.findById(id); return customer.map(ResponseEn ty::ok).orElseGet(() -> ResponseEn ty.notFound().build());

}

@Opera on(summary = "Create a new customer", descrip on = "Add a new customer to the system") @PostMapping

public ResponseEn ty<Customer> createCustomer(@Valid @SwaggerRequestBody(descrip on = "Customer object to be created") @RequestBody Customer customer) { Customer savedCustomer = customerRepository.save(customer); return ResponseEn ty.status(H pStatus.CREATED).body(savedCustomer);

}

@Opera on(summary = "Update an exis ng customer", descrip on = "Update the details of an exis ng customer") @PutMapping("/{id}")

public ResponseEn ty<Customer> updateCustomer(@PathVariable Long id, @Valid

@RequestBody Customer customer) {

if (!customerRepository.existsById(id)) { return ResponseEn ty.notFound().build();

}

customer.setId(id);

Customer updatedCustomer = customerRepository.save(customer); return ResponseEn ty.ok(updatedCustomer);

}

@Opera on(summary = "Delete a customer", descrip on = "Remove a customer from the system")

@DeleteMapping("/{id}") public ResponseEn ty<Void> deleteCustomer(@PathVariable Long id) { if (!customerRepository.existsById(id)) { return ResponseEn ty.notFound().build();

}

customerRepository.deleteById(id); return ResponseEn ty.noContent().build();

}

}