**Case Study: Product-Order Management System (With**

**Mockito Testing)**

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance> xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.2.4-SNAPSHOT</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>springtest</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springtest</name>

<description>Demo project for Spring Test</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>21</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springdoc</groupId>

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

<version>2.1.0</version>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<releases>

<enabled>false</enabled>

</releases>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<releases>

<enabled>false</enabled>

</releases>

</pluginRepository>

</pluginRepositories>

</project>

**application.properties**

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

spring.datasource.url=jdbc:h2:mem:testdb

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

spring.datasource.username=sa

spring.datasource.password=

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**Product.java**

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long productId;

private String name;

private double price;

private int availableQuantity;

public Product() {}

public Product(Long productId, String name, double price, int availableQuantity) {

this.productId = productId;

this.name = name;

this.price = price;

this.availableQuantity = availableQuantity;

}

public Long getproductId() {

return productId;

}

public void setproductId(Long productId) {

this.productId = productId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPrice() {

return price;

}

public void setprice(String price) {

this.price = price;

}

public int availableQuantity() {

return availableQuantity;

}

public void setavailableQuantity(int availableQuantity) {

this.availablequantity = availableQuantity;

}

}

**Order.java**

@Entity

@Table(name = "orders")

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long orderId;

@ManyToOne

private Product product;

private LocalDate orderDate;

private int quantityOrdered;

**ProductRepository.java**

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

import com.example.springtest.entity.User;

public interface ProductRepository extends JpaRepository<Product, Long> {}

**OrderRepository.java**

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

import com.example.springtest.entity.User;

public interface OrderRepository extends JpaRepository<Order, Long> {}

**ProductService.java**

@Service

public class ProductService {

@Autowired

private ProductRepository productRepository;

public Product addProduct(Product product) {

return productRepository.save(product);

}

public List<Product> getAllProducts() {

return productRepository.findAll();

}

public Product updateStock(Long productId, int qty) {

Product product = productRepository.findById(productId)

.orElseThrow(() -> new RuntimeException("Product not found"));

product.setAvailableQuantity(qty);

return productRepository.save(product);

}

}

**OrderService.java**

@Service

public class OrderService {

@Autowired

private OrderRepository orderRepository;

@Autowired

private ProductRepository productRepository;

public Order placeOrder(Long productId, int quantity) {

Product product = productRepository.findById(productId)

.orElseThrow(() -> new RuntimeException("Product not found"));

if (product.getAvailableQuantity() < quantity) {

throw new RuntimeException("Insufficient stock");

}

product.setAvailableQuantity(product.getAvailableQuantity() - quantity);

productRepository.save(product);

Order order = new Order();

order.setProduct(product);

order.setQuantityOrdered(quantity);

order.setOrderDate(LocalDate.now());

return orderRepository.save(order);

}

public List<Order> getAllOrders() {

return orderRepository.findAll();

}

}

**ProductController.java**

@RestController

@RequestMapping("/api/products")

public class ProductController {

@Autowired

private ProductService productService;

@PostMapping

public ResponseEntity<Product> add(@RequestBody Product p) {

return ResponseEntity.ok(productService.addProduct(p));

}

@GetMapping

public List<Product> getAll() {

return productService.getAllProducts();

}

@PutMapping("/{id}/stock")

public ResponseEntity<Product> updateStock(@PathVariable Long id, @RequestParam int qty) {

return ResponseEntity.ok(productService.updateStock(id, qty));

}

}

**OrderController.java**

@RestController

@RequestMapping("/api/orders")

public class OrderController {

@Autowired

private OrderService orderService;

@PostMapping

public ResponseEntity<Order> placeOrder(@RequestParam Long productId, @RequestParam int quantity) {

return ResponseEntity.ok(orderService.placeOrder(productId, quantity));

}

@GetMapping

public List<Order> getAll() {

return orderService.getAllOrders();

}

}

**ProductServiceTest.java**

@ExtendWith(MockitoExtension.class)

public class ProductServiceTest {

@InjectMocks

private ProductService productService;

@Mock

private ProductRepository productRepository;

@Test

void testAddProduct() {

Product product = new Product();

product.setName("Test Product");

when(productRepository.save(any(Product.class))).thenReturn(product);

Product saved = productService.addProduct(product);

assertEquals("Test Product", saved.getName());

}

@Test

void testGetAllProducts() {

when(productRepository.findAll()).thenReturn(List.of(new Product(), new Product()));

List<Product> products = productService.getAllProducts();

assertEquals(2, products.size());

}

@Test

void testUpdateStock() {

Product product = new Product();

product.setProductId(1L);

product.setAvailableQuantity(5);

when(productRepository.findById(1L)).thenReturn(Optional.of(product));

when(productRepository.save(any())).thenReturn(product);

Product updated = productService.updateStock(1L, 10);

assertEquals(10, updated.getAvailableQuantity());

}

}

**OrderServiceTest.java**

@ExtendWith(MockitoExtension.class)

public class OrderServiceTest {

@InjectMocks

private OrderService orderService;

@Mock

private OrderRepository orderRepository;

@Mock

private ProductRepository productRepository;

@Test

void testPlaceOrderSuccess() {

Product product = new Product();

product.setProductId(1L);

product.setAvailableQuantity(10);

when(productRepository.findById(1L)).thenReturn(Optional.of(product));

when(productRepository.save(any())).thenReturn(product);

when(orderRepository.save(any())).thenAnswer(invocation -> invocation.getArgument(0));

Order order = orderService.placeOrder(1L, 5);

assertEquals(5, order.getQuantityOrdered());

assertEquals(5, product.getAvailableQuantity());

}

@Test

void testPlaceOrderInsufficientStock() {

Product product = new Product();

product.setProductId(1L);

product.setAvailableQuantity(2);

when(productRepository.findById(1L)).thenReturn(Optional.of(product));

RuntimeException exception = assertThrows(RuntimeException.class, () -> {

orderService.placeOrder(1L, 5);

});

assertEquals("Insufficient stock", exception.getMessage());

}

}