Case Study: Product-Order Management System (With Mockito Testing)

Objective Develop a simple Product-Order system using Spring Boot with MySQL. Test the business logic of services using Mockito. No integration testing or H2 database involved.

 Functional Requirements

1. Admin can add, view, and update products.

2. Users can place orders for available products.

3. The system reduces stock when an order is placed.

4. Each order stores order details and is linked to the product.

 Entity Design

1. Product
   - productId (PK)
   - name
   - price
   - availableQuantity
2. Order
   - orderId (PK)
   - product (ManyToOne)
   - orderDate
   - quantityOrdered

 Repository Layer

- ProductRepository extends JpaRepository
- OrderRepository extends JpaRepository

Service Layer

ProductService

- addProduct(Product p)
- getAllProducts()
- updateStock(Long productId, int qty)

OrderService

- placeOrder(Long productId, int quantity)
  - ◦ Check if stock is available
  - ◦ Create order
  - ◦ Reduce product quantity

 Controller Layer

/api/products

- POST / → Add product
- GET / → List all products
- PUT /{id}/stock → Update stock

/api/orders

- POST / → Place order
- GET / → List all orders

 Unit Testing Strategy (Mockito only)

We test only the service layer using Mockito, without real DB access.

 ProductServiceTest

- Mock ProductRepository
- Test: ◦ Adding product ◦ Fetching all products
  - ◦ Stock update logic

 OrderServiceTest

- Mock OrderRepository and ProductRepository
- Test:
  - Order placed successfully when stock is available
  - Order fails if stock is insufficient

 Database Setup (MySQL)

In your application.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/product_order_db

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

No need for test profiles or alternate configurations.

 Tools & Tech Stack

- Spring Boot 3+
- Spring Data JPA
- MySQL
- JUnit 5
- Mockito

✅ Summary of Benefits

- Clean separation of concerns (MVC + layered architecture)
- Business logic isolated for testing
- Mockito ensures fast, DB-independent testing

- MySQL used consistently in development and 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
https://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</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>
```

Controller:

OrderController.java:

```
package com.example.productordersystem.controller;

import com.example.productordersystem.entity.Order;
import com.example.productordersystem.service.OrderService;
```

```
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/orders")
public class OrderController {
    private final OrderService service;
    public OrderController(OrderService service) {
        this.service = service;
    }
    @PostMapping
    public Order placeOrder(@RequestParam Long productId,
@RequestParam int quantity) {
        return service.placeOrder(productId, quantity);
    }
    @GetMapping
    public List<Order> getAllOrders() {
        return service.getAllOrders();
    }
}
```

ProductController.java:

```
package com.example.productordersystem.controller;
import com.example.productordersystem.entity.Product;
```

```
import com.example.productordersystem.service.ProductService;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/products")
public class ProductController {
    private final ProductService service;
    public ProductController(ProductService service) {
        this.service = service;
    }
    @PostMapping
    public Product addProduct(@RequestBody Product product) {
        return service.addProduct(product);
    }
    @GetMapping
    public List<Product> getAll() {
        return service.getAllProducts();
    }
    @PutMapping("/{id}/stock")
    public Product updateStock(@PathVariable Long id,
@RequestParam int qty) {
        return service.updateStock(id, qty);
    }
```

```
}
```

Entity:

Order.java:

```
package com.example.productordersystem.entity;
import jakarta.persistence.*;
import java.time.LocalDate;
@Entity(name = "orders")
public class Order {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long orderId;
   @ManyToOne
   private Product product;
   private LocalDate orderDate;
   private int quantityOrdered;
        public Long getOrderId() {
                return orderId;
        }
        public void setOrderId(Long orderId) {
                this.orderId = orderId;
        }
        public Product getProduct() {
```

```
        return product;
    }
    public void setProduct(Product product) {
        this.product = product;
    }
    public LocalDate getOrderDate() {
        return orderDate;
    }
    public void setOrderDate(LocalDate orderDate) {
        this.orderDate = orderDate;
    }
    public int getQuantityOrdered() {
        return quantityOrdered;
    }
    public void setQuantityOrdered(int quantityOrdered) {
        this.quantityOrdered = quantityOrdered;
    }
}
```

Product.java:

```
package com.example.productordersystem.entity;

import jakarta.persistence.*;
```

```
@Entity
public class Product {

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

    private String name;
    private double price;
    private int 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 double getPrice() {
```

```
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public int getAvailableQuantity() {
        return availableQuantity;
    }
    public void setAvailableQuantity(int availableQuantity) {
        this.availableQuantity = availableQuantity;
    }
}
```

Repository:

OrderRepository:

```
package com.example.productordersystem.repository;
import com.example.productordersystem.entity.Order;
import org.springframework.data.jpa.repository.JpaRepository;
public interface OrderRepository extends JpaRepository<Order, Long> {

}
```

ProductRepository:

```
package com.example.productordersystem.repository;


import com.example.productordersystem.entity.Product;

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


public interface ProductRepository extends JpaRepository<Product,
Long> {

}
```

Service:

OrderService.java:

```
package com.example.productordersystem.service;

import com.example.productordersystem.entity.Order;

import com.example.productordersystem.entity.Product;

import
com.example.productordersystem.repository.OrderRepository;

import
com.example.productordersystem.repository.ProductRepository;

import org.springframework.stereotype.Service;

import java.time.LocalDate;

import java.util.List;

@Service

public class OrderService {
```

```
    private final OrderRepository orderRepo;
    private final ProductRepository productRepo;
    public OrderService(OrderRepository orderRepo,
ProductRepository productRepo) {
        this.orderRepo = orderRepo;
        this.productRepo = productRepo;
    }
    public Order placeOrder(Long productId, int quantity) {
        Product product =
productRepo.findById(productId).orElseThrow();
        if (product.getAvailableQuantity() < quantity) {
            throw new RuntimeException("Not enough stock");
        }
        product.setAvailableQuantity(product.getAvailableQuantity() -
quantity);
        productRepo.save(product);
        Order order = new Order();
        order.setProduct(product);
        order.setOrderDate(LocalDate.now());
        order.setQuantityOrdered(quantity);
        return orderRepo.save(order);
    }
    public List<Order> getAllOrders() {
```

```
        return orderRepo.findAll();
    }
}
```

ProductService:

```
package com.example.productordersystem.service;
import com.example.productordersystem.entity.Product;
import
com.example.productordersystem.repository.ProductRepository;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class ProductService {
    private final ProductRepository repo;
    public ProductService(ProductRepository repo) {
        this.repo = repo;
    }
    public Product addProduct(Product product) {
        return repo.save(product);
    }
    public List<Product> getAllProducts() {
        return repo.findAll();
    }
    public Product updateStock(Long productId, int qty) {
```

```
        Product p = repo.findById(productId).orElseThrow();
        p.setAvailableQuantity(qty);
        return repo.save(p);
    }
}
```

ProductorderSystemApplication:

```
package com.example.productordersystem;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ProductordersystemApplication {

        public static void main(String[] args) {

        SpringApplication.run(ProductordersystemApplication.class,
args);
        }

}
```

Application.properties:

```
spring.application.name=productordersystem

spring.datasource.url=jdbc:mysql://localhost:3306/product_order_db

spring.datasource.username=root

spring.datasource.password=Swetha@57

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

server.port=8080
```

Service:

OrderServiceTest:

```
package com.example.productordersystem.service;

import com.example.productordersystem.entity.Order;

import com.example.productordersystem.entity.Product;

import
com.example.productordersystem.repository.OrderRepository;

import
com.example.productordersystem.repository.ProductRepository;

import org.junit.jupiter.api.Test;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
```

```
import static org.mockito.Mockito.*;

class OrderServiceTest {

    private final ProductRepository productRepo =
mock(ProductRepository.class);

    private final OrderRepository orderRepo =
mock(OrderRepository.class);

    private final OrderService service = new OrderService(orderRepo,
productRepo);

    @Test

    void testPlaceOrderSuccess() {

        Product product = new Product();

        product.setProductId(1L);

        product.setAvailableQuantity(10);


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

        when(orderRepo.save(any())).thenReturn(new Order());

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

        verify(productRepo).save(product);

        assertNotNull(order);

    }

    @Test

    void testPlaceOrderFailsDueToInsufficientStock() {

        Product product = new Product();

        product.setAvailableQuantity(2);
```

```
when(productRepo.findById(1L)).thenReturn(Optional.of(product));
        RuntimeException ex = assertThrows(RuntimeException.class, () -
> {
            service.placeOrder(1L, 5);
        });
        assertEquals("Not enough stock", ex.getMessage());
    }
}
```

ProductServiceTest:

```
package com.example.productordersystem.service;
import com.example.productordersystem.entity.Product;
import
com.example.productordersystem.repository.ProductRepository;
import org.junit.jupiter.api.Test;
import java.util.List;
import java.util.Optional;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
class ProductServiceTest {
    private final ProductRepository repo =
mock(ProductRepository.class);
    private final ProductService service = new ProductService(repo);
    @Test
```

```
    void testAddProduct() {
        Product p = new Product();
        p.setName("Laptop");
        when(repo.save(p)).thenReturn(p);
        Product result = service.addProduct(p);
        assertEquals("Laptop", result.getName());
    }
    @Test
    void testGetAllProducts() {
        when(repo.findAll()).thenReturn(List.of(new Product()));
        assertEquals(1, service.getAllProducts().size());
    }
    @Test
    void testUpdateStock() {
        Product p = new Product();
        p.setAvailableQuantity(10);
        when(repo.findById(1L)).thenReturn(Optional.of(p));
        when(repo.save(any())).thenReturn(p);
        Product updated = service.updateStock(1L, 15);
        assertEquals(15, updated.getAvailableQuantity());
    }
}
```