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<Product, Long>
• OrderRepository extends JpaRepository<Order, Long>

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

//ProductOrderApplication.java

```
package com.example.productorder;

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

@SpringBootApplication
public class ProductOrderApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProductOrderApplication.class, args);
    }
}
```

//Product.java

```
package com.example.productorder.entity;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```java
    private Long productId;

    private String name;
    private double price;
    private int availableQuantity;
}
```

//Order.java

```java
package com.example.productorder.entity;

import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDate;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Table(name = "orders") // 'order' is reserved in SQL
public class Order {

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

    @ManyToOne
    @JoinColumn(name = "product_id", nullable = false)
    private Product product;

    private LocalDate orderDate;
    private int quantityOrdered;
}
```

//ProductRepository.java

```java
package com.example.productorder.repository;

import com.example.productorder.entity.Product;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Long> {

}
```

//ProductService.java

```
package com.example.productorder.service;

import com.example.productorder.entity.Product;
import com.example.productorder.repository.ProductRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class ProductService {

    private final 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) {
        return productRepository.findById(productId).map(product -> {
            product.setAvailableQuantity(qty);
            return productRepository.save(product);
        }).orElseThrow(() -> new RuntimeException("Product not found"));
    }
}
```

//OrderService.java

```
package com.example.productorder.service;

import com.example.productorder.entity.Order;
import com.example.productorder.entity.Product;
import com.example.productorder.repository.OrderRepository;
import com.example.productorder.repository.ProductRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.util.List;

@Service
@RequiredArgsConstructor
```

```java
public class OrderService {

    private final OrderRepository orderRepository;
    private final 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 = Order.builder()
                .product(product)
                .orderDate(LocalDate.now())
                .quantityOrdered(quantity)
                .build();

        return orderRepository.save(order);
    }

    public List<Order> getAllOrders() {
        return orderRepository.findAll();
    }
}
```

//ProductController.java

```java
package com.example.productorder.controller;

import com.example.productorder.entity.Product;
import com.example.productorder.service.ProductService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/products")
@RequiredArgsConstructor
public class ProductController {

    private final ProductService productService;

    @PostMapping
    public Product addProduct(@RequestBody Product product) {
        return productService.addProduct(product);
    }

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    @PutMapping("/{id}/stock")
    public Product updateStock(@PathVariable Long id, @RequestParam int qty) {
        return productService.updateStock(id, qty);
    }
}
```

//OrderController.java

```java
package com.example.productorder.controller;

import com.example.productorder.entity.Order;
import com.example.productorder.service.OrderService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/orders")
@RequiredArgsConstructor
public class OrderController {

    private final OrderService orderService;

    @PostMapping
    public Order placeOrder(@RequestParam Long productId, @RequestParam int quantity) {
        return orderService.placeOrder(productId, quantity);
    }

    @GetMapping
    public List<Order> getAllOrders() {
        return orderService.getAllOrders();
    }
}
```

//Unit Tests (Mockito)

//ProductServiceTest.java

```
package com.example.productorder.service;

import com.example.productorder.entity.Product;
import com.example.productorder.repository.ProductRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import java.util.Arrays;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class ProductServiceTest {

    @Mock
    private ProductRepository productRepository;

    @InjectMocks
    private ProductService productService;

    public ProductServiceTest() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testAddProduct() {
        Product product = new Product(1L, "Laptop", 1000.0, 10);
        when(productRepository.save(product)).thenReturn(product);

        Product savedProduct = productService.addProduct(product);
        assertEquals("Laptop", savedProduct.getName());
        verify(productRepository, times(1)).save(product);
    }

    @Test
    void testGetAllProducts() {
        when(productRepository.findAll()).thenReturn(Arrays.asList(
                new Product(1L, "Laptop", 1000.0, 10),
                new Product(2L, "Phone", 500.0, 20)
        ));

        assertEquals(2, productService.getAllProducts().size());
    }

    @Test
    void testUpdateStock() {
        Product product = new Product(1L, "Laptop", 1000.0, 10);
        when(productRepository.findById(1L)).thenReturn(Optional.of(product));
        when(productRepository.save(product)).thenReturn(product);

        Product updated = productService.updateStock(1L, 15);
        assertEquals(15, updated.getAvailableQuantity());
    }
}
```

//OrderServiceTest.java

```
package com.example.productorder.service;

import com.example.productorder.entity.Order;
import com.example.productorder.entity.Product;
import com.example.productorder.repository.OrderRepository;
import com.example.productorder.repository.ProductRepository;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

class OrderServiceTest {

    @Mock
    private OrderRepository orderRepository;

    @Mock
    private ProductRepository productRepository;

    @InjectMocks
    private OrderService orderService;

    public OrderServiceTest() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testPlaceOrder_Success() {
        Product product = new Product(1L, "Laptop", 1000.0, 10);
        when(productRepository.findById(1L)).thenReturn(Optional.of(product));
        when(orderRepository.save(any(Order.class))).thenAnswer(invocation ->
invocation.getArgument(0));

        Order order = orderService.placeOrder(1L, 5);
        assertEquals(5, order.getQuantityOrdered());
        assertEquals(5, product.getAvailableQuantity());
        verify(productRepository, times(1)).save(product);
    }

    @Test
    void testPlaceOrder_InsufficientStock() {
        Product product = new Product(1L, "Laptop", 1000.0, 2);
        when(productRepository.findById(1L)).thenReturn(Optional.of(product));

        RuntimeException exception = assertThrows(RuntimeException.class, () ->
                orderService.placeOrder(1L, 5));

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