✅ 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

Entity's

Product.java

package com.example.productorder.entity;

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="products")
public class Product {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long productId;
	@Column(name="name")
	private String name;
	@Column(name="price")
	private double price;
	@Column(name="availableQuantity")
	private int availableQuantity;
	public Product() {
		// TODO Auto-generated constructor stub
	}
	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 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;
    }
    }
```

Order.java

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

```
import java.time.LocalDateTime;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.ManyToOne;

import jakarta.persistence.Table;

@Entity

@Table(name="orders")

public class Order

   { @Id

   @GeneratedValue(strategy = GenerationType.IDENTITY)

   private Long orderId;

   @ManyToOne

   @JoinColumn(name = "product_id")

   private Product product;

   private LocalDateTime orderDate;

   @Column(name="quantityOrdered")

   private int quantityOrdered;

   public Order() {

                  // TODO Auto-generated constructor stub

           }

           public Order(Long orderId, Product product, LocalDateTime orderDate, int
quantityOrdered) {

                  this.orderId = orderId;

                  this.product = product;

                  this.orderDate = orderDate;

                  this.quantityOrdered = 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 LocalDateTime getOrderDate() {
        return orderDate;
    }
    public void setOrderDate(LocalDateTime orderDate) {
        this.orderDate = orderDate;
    }
    public int getQuantityOrdered() {
        return quantityOrdered;
    }
    public void setQuantityOrdered(int quantityOrdered) {
        this.quantityOrdered = quantityOrdered;
    }
}
```

Repositorie's

ProductReposiories.java

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

```
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.productorder.entity.Product;
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

OrderRepository.java

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

Controller's

ProcductController.java

```
package com.example.productorder.Controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import com.example.productorder.entity.Product;
import com.example.productorder.Service.ProductService;
@RestController
@RequestMapping("/api/products")
public class ProductController
        { @Autowired
          private 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 quantity) {
        return productService.updateStock(id, quantity);
    }
}
```

OrderController.java

```
package com.example.productorder.Controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import com.example.productorder.entity.Order;
import com.example.productorder.Service.OrderService;
@RestController
@RequestMapping("/api/orders")
public class OrderController
        { @Autowired
    private 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();
    }
}
```

Services

ProductService.java

```
package com.example.productorder.Service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.productorder.entity.Product;
import com.example.productorder.repository.ProductRepository;
@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 quantity) {
```

```
        Product product = productRepository.findById(productId)
                .orElseThrow(() -> new RuntimeException("Product not found"));
        product.setAvailableQuantity(quantity);
        return productRepository.save(product);
    }
}
```

OrderService.java

```
package com.example.productorder.Service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.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;
@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();
        return orderRepository.save(order);
    }

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

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>productorder</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>productorder</name>
        <description>Product Order Management System</description>
        <url/>
        <licenses>
```

```
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>17</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.mysql</groupId>
                <artifactId>mysql-connector-j</artifactId>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
</dependencies>

<build>
        <plugins>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <configuration>
                                <annotationProcessorPaths>
                                        <path>
                                                <groupId>org.projectlombok</groupId>
                                                <artifactId>lombok</artifactId>
                                        </path>
                                </annotationProcessorPaths>
                        </configuration>
```

```
                </plugin>
                <plugin>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-maven-plugin</artifactId>
                        <configuration>
                                <excludes>
                                        <exclude>
                                                <groupId>org.projectlombok</groupId>
                                                <artifactId>lombok</artifactId>
                                        </exclude>
                                </excludes>
                        </configuration>
                </plugin>
        </plugins>
    </build>

</project>
```

Application.Properties

```
spring.application.name=ProductOrderManagementSystem
spring.datasource.url=jdbc:mysql://localhost:3306/producttt_order_db
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=Sujitha@27
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
server.port=8080
# Disable H2 console if not needed
spring.h2.console.enabled=false
```

ProductApplication.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);
	}
}
```

Testing Purpose

ProductServiceTest:

```
package com.example.productorder;
import static org.assertj.core.api.Assertions.*;
import static org.mockito.ArgumentMatchers.*;
import static org.mockito.Mockito.*;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
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 com.example.productorder.entity.Product;
import com.example.productorder.repository.ProductRepository;
import com.example.productorder.Service.ProductService;
@ExtendWith(MockitoExtension.class)
```

```
public class ProductServiceTest
   { @Mock
   private ProductRepository productRepository;

   @InjectMocks
   private ProductService productService;

   @Test
   public void testAddProduct() {
      Product doveSoap = new Product(1L, "Dove Soap", 75, 120);

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

      Product result = productService.addProduct(doveSoap);

      assertThat(result)
         .isNotNull()
         .extracting(Product::getName, Product::getPrice)
         .containsExactly("Dove Soap", 75.0);

      verify(productRepository).save(doveSoap);
   }

   @Test
   public void testGetAllProducts() {
      Product colgate = new Product(1L, "Colgate Toothpaste", 50, 80);
      Product surfExcel = new Product(2L, "Surf Excel", 200, 40);

      when(productRepository.findAll()).thenReturn(Arrays.asList(colgate,   surfExcel));
```

```
    List<Product> result = productService.getAllProducts();

    assertThat(result)
        .hasSize(2)
        .extracting(Product::getName)
        .containsExactly("Colgate Toothpaste", "Surf Excel");
}

@Test
public void testUpdateStock() {
    Product goodDayCookies = new Product(1L, "Good Day Cookies", 30, 60);
    Product updatedProduct = new Product(1L, "Good Day Cookies", 30, 45);

    when(productRepository.findById(1L)).thenReturn(Optional.of(goodDayCookies));
    when(productRepository.save(any(Product.class))).thenReturn(updatedProduct);

    Product result = productService.updateStock(1L, 45);

    assertThat(result)
        .isNotNull()
        .extracting(Product::getAvailableQuantity)
        .isEqualTo(45);

    verify(productRepository).findById(1L);
    verify(productRepository).save(argThat(p -> p.getAvailableQuantity() == 45));
}

@Test
public void testUpdateStock_ProductNotFound() {
    when(productRepository.findById(99L)).thenReturn(Optional.empty());
```

```
        assertThatThrownBy(() -> productService.updateStock(99L, 100))
            .isInstanceOf(RuntimeException.class)
            .hasMessageContaining("Product not found");

        verify(productRepository, never()).save(any());
    }
}
```

OrderServiceTest:

```
package com.example.productorder;

import static org.assertj.core.api.Assertions.*;
import static org.mockito.Mockito.*;
import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
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 com.example.productorder.entity.Order;
import com.example.productorder.entity.Product;
import com.example.productorder.repository.OrderRepository;
import com.example.productorder.repository.ProductRepository;
import com.example.productorder.Service.OrderService;

@ExtendWith(MockitoExtension.class)
public class OrderServiceTest {
```

```
@Mock
private OrderRepository orderRepository;

@Mock
private ProductRepository productRepository;

@InjectMocks
private OrderService orderService;

@Test
public void testPlaceOrder_Success() {
    Product laptop = new Product(1L, "Laptop", 50000, 20);

    Order expectedOrder = new Order();
    expectedOrder.setOrderId(1L);
    expectedOrder.setProduct(laptop);
    expectedOrder.setQuantityOrdered(2);
    expectedOrder.setOrderDate(LocalDateTime.now());

    when(productRepository.findById(1L)).thenReturn(Optional.of(laptop));
    when(orderRepository.save(any(Order.class))).thenReturn(expectedOrder);

    Order result = orderService.placeOrder(1L, 2);

    assertThat(result).isNotNull();
    assertThat(result.getQuantityOrdered()).isEqualTo(2);
    assertThat(result.getProduct().getName()).isEqualTo("Laptop");
}

@Test
```

```
public void testPlaceOrder_InsufficientStock() {
    Product smartphone = new Product(2L, "Smartphone", 20000, 5);

    when(productRepository.findById(2L)).thenReturn(Optional.of(smartphone));

    assertThatThrownBy(() -> orderService.placeOrder(2L, 10))
        .isInstanceOf(RuntimeException.class)
       .hasMessageContaining("Insufficient stock");

    verify(orderRepository, never()).save(any());
}

@Test
public void testGetAllOrders() {
    Product laptop = new Product(1L, "Laptop", 50000, 20);
    Product smartphone = new Product(2L, "Smartphone", 20000, 5);

    Order order1 = new Order();
    order1.setOrderId(1L);
    order1.setProduct(laptop);
    order1.setQuantityOrdered(1);

    Order order2 = new Order();
    order2.setOrderId(2L);
    order2.setProduct(smartphone);
    order2.setQuantityOrdered(3);

    when(orderRepository.findAll()).thenReturn(Arrays.asList(order1,  order2));

    List<Order> result = orderService.getAllOrders();
```

```
        assertThat(result)
            .hasSize(2)
            .extracting(Order::getProduct)
            .extracting(Product::getName)
            .containsExactly("Laptop", "Smartphone");
    }
}
```

ProductOrderApplicationTest.java

```
package com.example.productorder;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest
class ProductorderApplicationTests
            { @Test
            void contextLoads() {
            }
}
```

SQL WORKBENCH

```
CREATE DATABASE productt_order_db;
USE productt_order_db;
SHOW TABLES;
INSERT INTO products VALUES (1, 2346, 'sree', 89.788);
INSERT INTO orders (order_id, order_date, product_id, quantity_ordered)
VALUES (1, NOW(), 1, 45454);
select * from products;
select * from orders;
```