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

Product.java

```
package com.example.productordermanagementsystem.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 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 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.productordermanagementsystem.entity;
import java.util.Date;
import jakarta.persistence.*;
@Entity
@Table(name = "orders")
public class Order {
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private Long orderId;

	@ManyToOne(cascade = CascadeType.ALL)
	@JoinColumn(name = "product_id")
	private Product product;

	@Temporal(TemporalType.TIMESTAMP)
	private Date orderDate;

	private int quantityOrdered;
	public Order() {}
	public Order(Long orderId, Product product, Date 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 Date getOrderDate() {
		return orderDate;
	}
	public void setOrderDate(Date orderDate) {
		this.orderDate = orderDate;
	}
	public int getQuantityOrdered() {
```

```
        return quantityOrdered;
    }
    public void setQuantityOrdered(int quantityOrdered) {
        this.quantityOrdered = quantityOrdered;
    }
}
```

ProductRepository.java

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

OrderRepository.java

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

ProductService.java

```
package com.example.productordermanagementsystem.service;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.productordermanagementsystem.entity.Product;
import com.example.productordermanagementsystem.repository.ProductRepository;
@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepo;

    public Product addProduct(Product product) {
        return productRepo.save(product);
    }

    public List<Product> getAllProducts(){
        return productRepo.findAll();
    }

     public Product updateStock(Long productId, int qty) {
          Optional<Product> existingProduct = productRepo.findById(productId);
```

```
        if (existingProduct.isPresent()) {
            Product product = existingProduct.get();
            product.setAvailableQuantity(qty);
            return productRepo.save(product);
        } else {
            throw new RuntimeException("Product not found with ID: " + productId);
        }
    }
}
```

OrderService.java

```
package com.example.productordermanagementsystem.service;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.productordermanagementsystem.entity.Product;
import com.example.productordermanagementsystem.repository.ProductRepository;
@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepo;

    public Product addProduct(Product product) {
        return productRepo.save(product);
    }

    public List<Product> getAllProducts(){
        return productRepo.findAll();
    }
     public Product updateStock(Long productId, int qty) {
        Optional<Product> existingProduct = productRepo.findById(productId);
        if (existingProduct.isPresent()) {
            Product product = existingProduct.get();
            product.setAvailableQuantity(qty);
            return productRepo.save(product);
        } else {
            throw new RuntimeException("Product not found with ID: " + productId);
        }
    }
}
```

ProductOrderController.java

```
package com.example.productordermanagementsystem.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.example.productordermanagementsystem.entity.Order;
import com.example.productordermanagementsystem.entity.Product;
import com.example.productordermanagementsystem.service.OrderService;
import com.example.productordermanagementsystem.service.ProductService;
@RestController
@RequestMapping("/api")
public class ProductOrderController {

	@Autowired
	private ProductService productService;

	@Autowired
	private OrderService orderService;

//	PRODUCT
	@PostMapping("/products")
	public Product createProduct(@RequestBody Product product) {
		return productService.addProduct(product);
	}

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

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

//	ORDER
	@GetMapping("/orders")
	public List<Order> getAllOrders(){
		return orderService.getAllOrders();
	}

	@PostMapping("/orders")
	public Order orderProduct(@RequestParam Long productId, @RequestParam int
quantityOrdered){
		return orderService.placeOrder(productId ,quantityOrdered);
	}
}
```

ProductServiceTest.java

```
package com.example.productordermanagementsystem;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.mockito.Mockito.when;
import java.util.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.*;
import com.example.productordermanagementsystem.entity.Product;
import com.example.productordermanagementsystem.repository.ProductRepository;
import com.example.productordermanagementsystem.service.ProductService;
import org.mockito.junit.jupiter.MockitoExtension;
@ExtendWith(MockitoExtension.class)
public class ProductServiceTest {
  @Mock
  private ProductRepository productRepo;
  @InjectMocks
  private ProductService productService;
  @Test
  void testAddProduct() {
     Product mockProduct = new Product();
     mockProduct.setName("Phone");
     when(productRepo.save(mockProduct)).thenReturn(mockProduct);
     Product result = productService.addProduct(mockProduct);
     assertThat(result.getName()).isEqualTo("Phone");
  }
  @Test
  void testGetAllProducts() {
     Product p1 = new Product();
     Product p2 = new Product();
     when(productRepo.findAll()).thenReturn(Arrays.asList(p1, p2));
     var result = productService.getAllProducts();
     assertThat(result).hasSize(2);
  }
  @Test
  void testUpdateStockSuccess() {
     Product product = new Product();
     product.setProductId(1L);
     product.setAvailableQuantity(5);
     when(productRepo.findById(1L)).thenReturn(Optional.of(product));
     when(productRepo.save(product)).thenReturn(product);
     Product result = productService.updateStock(1L, 10);
     assertThat(result.getAvailableQuantity()).isEqualTo(10);
  }
  @Test
  void testUpdateStock_ProductNotFound() {
     when(productRepo.findById(1L)).thenReturn(Optional.empty());
```

```
        Exception ex = assertThrows(RuntimeException.class, () -> {
            productService.updateStock(1L, 5);
        });
        assertThat(ex.getMessage()).contains("Product not found");
    }
}
```

OrderServiceTest.java

```
package com.example.productordermanagementsystem;
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;
import java.util.Optional;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.*;
import org.mockito.junit.jupiter.MockitoExtension;
import com.example.productordermanagementsystem.entity.*;
import com.example.productordermanagementsystem.repository.*;
import com.example.productordermanagementsystem.service.OrderService;
@ExtendWith(MockitoExtension.class)
public class OrderServiceTest {
    @Mock
    private OrderRepository orderRepo;
    @Mock
    private ProductRepository productRepo;
    @InjectMocks
    private OrderService orderService;
    @Test
    void testPlaceOrderSuccess() {
        Product product = new Product();
        product.setProductId(1L);
        product.setName("Laptop");
        product.setAvailableQuantity(10);
        when(productRepo.findById(1L)).thenReturn(Optional.of(product));
        when(productRepo.save(any())).thenReturn(product);
        when(orderRepo.save(any())).thenAnswer(inv -> inv.getArgument(0));
        Order result = orderService.placeOrder(1L, 4);
        assertThat(result.getQuantityOrdered()).isEqualTo(4);
        assertThat(result.getProduct().getName()).isEqualTo("Laptop");
        assertThat(product.getAvailableQuantity()).isEqualTo(6); // 10 - 4
    }
    @Test
    void testPlaceOrder_InsufficientStock() {
        Product product = new Product();
        product.setProductId(1L);
```

```
        product.setName("Tablet");
        product.setAvailableQuantity(2);
        when(productRepo.findById(1L)).thenReturn(Optional.of(product));
        Exception ex = assertThrows(RuntimeException.class, () -> {
            orderService.placeOrder(1L, 5);
        });
        assertThat(ex.getMessage()).contains("Insufficient stock");
    }
}
```