Case Study: Simple eCommerce Application (No Database)

1. Application Overview:

- **Functionality**:
  - Display list of products.
  - Add products to a cart.
  - Place an order.
  - View order details.
- **Architecture**:
  - Use Spring Boot as the framework.
  - Data will be stored in-memory (using a list or map).
  - No integration with a real database.

2. Key Components:

- **Product**: Represents an item available for sale.
- **Cart**: Holds the products added by the user.
- **Order**: Represents a transaction with the selected products.

3. Spring Boot Setup:

Dependencies:

In pom.xml, you need only a few basic dependencies for this:

xml

Copy code

```xml
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
</dependencies>
```

4. Model Classes:

Product.java:

java

Copy code

```
public class Product {
    private Long id;
    private String name;
    private double price;

    // Constructors, Getters, Setters
}
```

CartItem.java:

java

Copy code

```
public class CartItem {
    private Product product;
    private int quantity;

    // Constructors, Getters, Setters
}
```

Order.java:

java

Copy code

```
public class Order {
    private Long orderId;
    private List<CartItem> items;
    private double totalPrice;

    // Constructors, Getters, Setters
}
```

5. Service Layer:

ProductService.java:

java

Copy code

```
import org.springframework.stereotype.Service;
import java.util.*;

@Service
```

```
public class ProductService {
    private List<Product> products = new ArrayList<>();

    public ProductService() {
        // Sample products
        products.add(new Product(1L, "Laptop", 1200));
        products.add(new Product(2L, "Smartphone", 800));
    }

    public List<Product> getAllProducts() {
        return products;
    }

    public Product getProductById(Long id) {
        return products.stream().filter(p -> p.getId().equals(id)).findFirst().orElse(null);
    }
}
```

OrderService.java:

java

Copy code

```
import org.springframework.stereotype.Service;
import java.util.*;

@Service
public class OrderService {
    private Map<Long, Order> orders = new HashMap<>();
    private Long nextOrderId = 1L;

    public Order placeOrder(List<CartItem> items) {
        double total = items.stream().mapToDouble(item -> item.getProduct().getPrice() * item.getQuantity()).sum();
        Order order = new Order(nextOrderId++, items, total);
        orders.put(order.getOrderId(), order);
        return order;
    }
    public Order getOrderById(Long id) {
        return orders.get(id);
    }
}
```

6. Controller Layer:

ProductController.java:

java

Copy code

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@Controller
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping("/products")
    public String showProducts(Model model) {
        model.addAttribute("products", productService.getAllProducts());
        return "products";
    }

    @GetMapping("/products/{id}")
    public String showProductDetails(@PathVariable("id") Long id, Model model) {
        Product product = productService.getProductById(id);
        model.addAttribute("product", product);
        return "product-details";
    }
}
```

OrderController.java:

java

Copy code

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.*;

@Controller
@RequestMapping("/orders")
public class OrderController {

    @Autowired
    private OrderService orderService;

    private List<CartItem> cart = new ArrayList<>();

    @PostMapping("/add-to-cart")
    public String addToCart(@RequestParam("productId") Long productId, @RequestParam("quantity") int quantity) {
        Product product = productService.getProductById(productId);
        cart.add(new CartItem(product, quantity));
        return "redirect:/cart";
    }

    @GetMapping("/cart")
    public String showCart(Model model) {
        model.addAttribute("cart", cart);
        return "cart";
    }

    @PostMapping("/checkout")
    public String checkout(Model model) {
        Order order = orderService.placeOrder(cart);
        cart.clear();  // Empty the cart after placing the order
        model.addAttribute("order", order);
        return "order-confirmation";
    }
}
```

7. Thymeleaf Views:

- products.html: Displays the list of products.
- product-details.html: Shows details of a single product.
- cart.html: Displays items added to the cart.
- order-confirmation.html: Shows the confirmation of the placed order.

Here is a simple example of products.html:

html

Copy code

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Products</title>
</head>
<body>
    <h1>Product List</h1>
    <ul>
        <li th:each="product : ${products}">
            <a th:href="@{/products/{id}(id=${product.id})}"><span th:text="${product.name}"></span></a> -
            <span th:text="${product.price}"></span> USD
        </li>
    </ul>
</body>
</html>
```

8. Running the Application:

You can run the Spring Boot application using mvn spring-boot:run, and access the product list at http://localhost:8080/products.

This setup simulates an eCommerce application with basic functionality and in-memory data handling, which can be useful for learning purposes without involving a database.