German University in Cairo
Department of Computer Science
Assoc. Prof. Mervat Abu El-Kheir

Architecture of Massively Scalable Applications, Spring 2025, Spring 2025

Mini Project 1

Deadline is Sunday 09/03/2025 11:59 PM

1 Introduction

This project is a basic application using Spring Boot. It consists of four main models: **User**, **Product**, **Cart**, and **Order**. Each model has a corresponding repository, service, and controller. The objective is to implement missing methods in the service, repository, and controller files.

2 Installation

You Should clone the project from this link: https://github.com/Scalable2025/Mini-Project1-Base.git then structure the project as follows

3 Project Structure

The project follows a standard Spring Boot architecture. **Please name the packages as the same names mentioned below:**

```
|- src
|    |-main/java/com/example
|    |    |-MiniProject1
|    |    |    |-MiniProject1Application.java
|    |    |-data
|    |    |    |- users.json
|    |    |    |- products.json
|    |    |    |- carts.json
|    |    |    |- orders.json
|    |    |- model
|    |    |    |- User.java
|    |    |    |- Product.java
|    |    |    |- Cart.java
|    |    |    |- Order.java
|    |    |- repository
|    |    |    |- UserRepository.java
|    |    |    |- ProductRepository.java
|    |    |    |- CartRepository.java
|    |    |    |- OrderRepository.java
|    |    |- service
|    |    |    |- UserService.java
|    |    |    |- ProductService.java
|     |     |     |- CartService.java
|     |     |     |- OrderService.java
|     |     |- controller
|     |     |     |- UserController.java
|     |     |     |- ProductController.java
|     |     |     |- CartController.java
|     |     |     |- OrderController.java
|     |-test/java/com/example
|     |     |-MiniProject1ApplicationTests.java
```

4 Data

In folder data there are json files for every model in the project where the objects of each class should be stored. For example users.json will contain the users you add by your methods like addUser() and you will get these users also by methods like getUsers()

5 Models

All the Models must include the getters and setter for all the attributes present in the models. Also, your implementation should include 3 constructors for each class as implemented in Lab 1.

5.1 User

Represents individual customers. Each user has a unique identifier, a name, and a list of their orders.

```java
@Component
public class User {
    private UUID id;
    private String name;
    private List<Order> orders=new ArrayList<>();
}
```

5.2 Product

Defines items available for purchase, with attributes like a unique identifier, name, and price.

```java
@Component
public class Product {
    private UUID id;
    private String name;
    private double price;
}
```

5.3 Cart

Acts as a temporary storage for products that users intend to buy. Each cart is associated with a user and maintains a list of products.

```java
@Component
public class Cart {
    private UUID id;
    private UUID userId;
    private List<Product> products=new ArrayList<>();
}
```

5.4 Order

Captures finalized orders. An order includes a unique identifier, the associated user's identifier, the total price, and a list of purchased products.

```
@Component
public class Order {
    private UUID id;
    private UUID userId;
    private double totalPrice;
    private List<Product> products=new ArrayList<>();
}
```

---

6 Repositories

You should implement standard CRUD operations.

6.1 Main Repository

This is an abstract class that is provided to you in the code template. You will use for the other repositories that you will create to extend from. Think carefully why do you have to use it and what does it have to do with Dependency Injection.

Please Don't edit the implementation in the class as it has the methods that you will need to edit or read the JSON files.

6.2 UserRepository

The repository that will interact with the users JSON file.

6.2.1 Class Defintion

```
@Repository
@SuppressWarnings("rawtypes")
public class UserRepository extends MainRepository<User>{


    public UserRepository() {

    }
}
```

6.2.2 Required Methods

6.2.2.1 Get Users

Retrieve all users from the JSON file.

```java
public ArrayList<User> getUsers();
```

6.2.2.2 Get User By ID

Fetch a specific user from the JSON file by their unique ID.

```java
public User getUserById(UUID userId);
```

6.2.2.3 Add User

Adds a new user to the users JSON file.

```java
public User addUser(User user);
```

6.2.2.4 Get The Orders of a User

Retrieve all orders for a given user ID.

```java
public List<Order> getOrdersByUserId(UUID userId);
```

6.2.2.5 Add Order to the User

Lets the User add order.

```java
public void addOrderToUser(UUID userId, Order order);
```

6.2.2.6 Remove Order from User

Let the user remove one of his/her orders.

```java
public void removeOrderFromUser(UUID userId, UUID orderId);
```

6.2.2.7 Delete User

Delete a user by passing his/her ID.

```java
public void deleteUserById(UUID userId);
```

6.3 ProductRepository

The repository that will interact with the products JSON file.

6.3.1 Class Definition

```java
@Repository
@SuppressWarnings("rawtypes")
public class ProductRepository extends MainRepository<Product> {

    public ProductRepository() {
    }
}
```

6.3.2 Required Methods

6.3.2.1 Add Product

Adding a new Product to the products JSON file

```
public Product addProduct(Product product);
```

6.3.2.2 Get All Products

To get all the products present in the JSON file.

```
public ArrayList<Product> getProducts();
```

6.3.2.3 Get Specific Product

To a specific product by passing the product ID.

```
public Product getProductById(UUID productId);
```

6.3.2.4 Update a Product

To update a product with a new name and new price.

```
public Product updateProduct(UUID productId, String newName, double newPrice);
```

6.3.2.5 Apply Discount

To apply the given discount to an array list of products' IDs. The discount will be given as integer number (60 means apply 60% discount on the given products' IDs)

```
public void applyDiscount(double discount, ArrayList<UUID> productIds);
```

6.3.2.6 Delete a Product

To delete a product by passing a specific product ID.

```
public void deleteProductById(UUID productId);
```

6.4 CartRepository

The repository that will interact with the carts JSON file.

6.4.1 Class

```
@Repository
@SuppressWarnings("rawtypes")
public class CartRepository extends MainRepository<Cart> {

    public CartRepository(){

    }
}
```

6.4.2 Required Methods

6.4.2.1 Add New Cart

To a new cart into the carts json file

```
public Cart addCart(Cart cart);
```

6.4.2.2 Get All Carts

To get all the carts from the carts JSON file

```
public ArrayList<Cart> getCarts();
```

6.4.2.3 Get Specific Cart

To a specific cart from the carts JSON file by passing its ID

```
public Cart getCartById(UUID cartId);
```

6.4.2.4 Get User's Cart

Get the cart of a specific use from the carts JSON file by passing the user's ID.

```
public Cart getCartByUserId(UUID userId);
```

6.4.2.5 Add Product to Cart

To add a new product to the cart

```
public void addProductToCart(UUID cartId, Product product);
```

6.4.2.6 Delete Product

To delete a specific product from a specific cart pass their IDs.

```
public void deleteProductFromCart(UUID cartId, Product product);
```

6.4.2.7 Delete the Cart

To delete a specific cart by passing its ID.

```
public void deleteCartById(UUID cartId);
```

6.5 OrderRepository

The repository that will interact with the orders JSON file.

6.5.1 Class Definition

```
@Repository
@SuppressWarnings("rawtypes")
public class OrderRepository extends MainRepository<Order> {

    public OrderRepository() {

    }
}
```

6.5.2 Required Methods

6.5.2.1 Add Order

To a new Order in the orders JSON file.

```
public void addOrder(Order order);
```

6.5.2.2 Get All Orders

To get all the orders inside the orders JSON file.

```
public ArrayList<Order> getOrders();
```

6.5.2.3 Get a Specific Order

To get a specific order by passing its ID.

```
public Order getOrderById(UUID orderId);
```

6.5.2.4 Delete a Specific Order

To Delete a Specific order by passing its ID

```
public void deleteOrderById(UUID orderId);
```

7 Services

All logic, aside from interacting with the JSON files, should be handled in the services.

7.1 Main Service

This is an abstract class that is provided to you in the code template. You will use for the other services that you will create to extend from. Think carefully why do you have to use it and what does it have to do with Dependency Injection.

7.2 UserService

The service handling the functionalities related to the User.

7.2.1 Class Definition

```
@Service
@SuppressWarnings("rawtypes")
public class UserService extends MainService<User>{

    //The Dependency Injection Variables

    //The Constructor with the requried variables mapping the Dependency Injection.

}
```

7.2.2 Required Methods

7.2.2.1 Add New User

To add a new User to our system

```
public User addUser(User user);
```

7.2.2.2 Get the Users

To Get all the Users present in our system.

```
public ArrayList<User> getUsers();
```

7.2.2.3 Get a Specific User

To get a specific user by passing his/her ID.

```
public User getUserById(UUID userId);
```

7.2.2.4 Get the User's Orders

To get all the orders that were created by the user. It takes the user ID as an input.

```
public List<Order> getOrdersByUserId(UUID userId);
```

7.2.2.5 Add a New Order

Here the user checks out his cart by creating a new order. The user should empty his cart and calculate everything related to his order and add the new order to his list of orders. It should call methods from CartService

Updated Method

```
public void addOrderToUser(UUID userId);
```

7.2.2.6 Empty Cart

This method should empty the cart of the user from the products present inside. It should call methods from CartService

```
public void emptyCart(UUID userId);
```

7.2.2.7 Remove Order

To remove a specific order from the list of orders of the user.

```
public void removeOrderFromUser(UUID userId, UUID orderId);
```

7.2.2.8 Delete the User

To delete a specific user by passing his ID.

```
public void deleteUserById(UUID userId);
```

7.3 ProductService

The service handling the functionalities related to the Product.

7.3.1 Class Definition

```
@Service
@SuppressWarnings("rawtypes")
public class ProductService extends MainService<Product> {

    //The Dependency Injection Variables

    //The Constructor with the requried variables mapping the Dependency Injection.

}
```

7.3.2 Required Methods

7.3.2.1 Add New Product

To add a new Product to our system.

```
public Product addProduct(Product product);
```

7.3.2.2 Get All Products

To get all the different products from the system.

```
public ArrayList<Product> getProducts();
```

7.3.2.3 Get a Specific Product

To get a specific product by passing its ID.

```
public Product getProductById(UUID productId);
```

7.3.2.4 Update a Product

To update a specific product by passing its ID and the new name and new price.

```
public Product updateProduct(UUID productId, String newName, double newPrice);
```

7.3.2.5 Apply Discount

To apply the given discount to an array list of products' IDs. The discount will be given as decimal number (60 means apply 60% discount on the given products' IDs)

```
public void applyDiscount(double discount, ArrayList<UUID> productIds);
```

7.3.2.6 Delete a Product

To delete a specific product by passing its ID.

```
public void deleteProductById(UUID productId);
```

7.4 CartService

The service handling the functionalities related to the Cart.

7.4.1 Class Definition

```
@Service
@SuppressWarnings("rawtypes")
public class CartService extends MainService<Cart>{

    //The Dependency Injection Variables

    //The Constructor with the requried variables mapping the Dependency Injection.
}
```

7.4.2 Required Methods

7.4.2.1 Add Cart

To add a new cart to the our system.

```java
public Cart addCart(Cart cart);
```

7.4.2.2 Get All Carts

To get all the carts in our system.

```java
public ArrayList<Cart> getCarts();
```

7.4.2.3 Get a Specific Cart

To get a specific cart by passing its ID.

```java
public Cart getCartById(UUID cartId);
```

7.4.2.4 Get a User's Cart

To get the cart of a specific user.

```java
public Cart getCartByUserId(UUID userId);
```

7.4.2.5 Add Product to the Cart

To add a specific product to the cart passing their IDs.

```java
public void addProductToCart(UUID cartId, Product product);
```

7.4.2.6 Delete Product from the Cart

To delete a specific product from the cart passing their IDs.

```java
public void deleteProductFromCart(UUID cartId, Product product);
```

7.4.2.7 Delete the Cart

To delete the cart by passing their IDs.

```java
public void deleteCartById(UUID cartId);
```

7.5 OrderService

The service handling the functionalities related to the Order.

7.5.1 Class Definition

```java
@Service
@SuppressWarnings("rawtypes")
public class OrderService extends MainService<Order> {

    //The Dependency Injection Variables
    //The Constructor with the requried variables mapping the Dependency Injection.
}
```

7.5.2 Required Methods

7.5.2.1 Add Order

To add a new order to our system.

```
public void addOrder(Order order);
```

7.5.2.2 Get All Orders

To get all the orders in our System.

```
public ArrayList<Order> getOrders();
```

7.5.2.3 Get a specific order

To get a specific order by passing its ID.

```
public Order getOrderById(UUID orderId);
```

7.5.2.4 Delete a Specific Order

To delete a specific order by passing its ID.

```
public void deleteOrderById(UUID orderId) throws IllegalArgumentException;
```

---

8 Controllers

You should implement RESTful endpoints. **And Please Do not change the RequestMapping paths keep it as it is**

8.1 UserController

TO handle the user's endpoints (The user router)

8.1.1 Class Definition

```
@RestController
@RequestMapping("/user")
public class UserController {

    //The Dependency Injection Variables

    //The Constructor with the requried variables mapping the Dependency Injection.

}
```

8.1.2 Required Endpoints

8.1.2.1 Add User

Post Request to add a new user

```
@PostMapping("/")
public User addUser(@RequestBody User user);
```

8.1.2.2 Get All Users

Get Request to get all users

```
@GetMapping("/")
public ArrayList<User> getUsers();
```

8.1.2.3 Get Specific User

Get Request to get a specific user by passing his/her ID in the URL

```
@GetMapping("/{userId}")
public User getUserById(@PathVariable UUID userId);
```

8.1.2.4 Get a User's Orders

Get Request to get the orders of a specific user by passing his/her ID in the URL

```
@GetMapping("/{userId}/orders")
public List<Order> getOrdersByUserId(@PathVariable UUID userId);
```

8.1.2.5 Check Out

Post Request to issue a new order for the user.

```
@PostMapping("/{userId}/checkout")
public String addOrderToUser(@PathVariable UUID userId);
```

8.1.2.6 Remove Order

Post Request to remove a specific order from the user.

```
@PostMapping("/{userId}/removeOrder")
public String removeOrderFromUser(@PathVariable UUID userId, @RequestParam UUID orderId);
```

8.1.2.7 Empty Cart

Delete Request to empty the cart of the user.

```
@DeleteMapping("/{userId}/emptyCart")
public String emptyCart(@PathVariable UUID userId);
```

8.1.2.8 Add Product To the Cart

Put Request to add a specific product to the cart by passing their IDs in the request body.

```
@PutMapping("/addProductToCart")
public String addProductToCart(@RequestParam UUID userId, @RequestParam UUID productId);
```

8.1.2.9 Delete Product from the Cart

Put Request to delete a specific product from the Cart

```
@PutMapping("/deleteProductFromCart")
public String deleteProductFromCart(@RequestParam UUID userId, @RequestParam UUID productId);
```

8.1.2.10 Delete User

Delete Request to delete a specific user.

```
@DeleteMapping("/delete/{userId}")
public String deleteUserById(@PathVariable UUID userId);
```

8.2 ProductController

To handle the products's endpoints (The Product router)

8.2.1 Class

```
@RestController
@RequestMapping("/product")
public class ProductController {

    //The Dependency Injection Variables

    //The Constructor with the requried variables mapping the Dependency Injection.

}
```

8.2.2 Required Endpoints

8.2.2.1 Add Product

Post Request to add a new Product to the system.

```
@PostMapping("/")
public Product addProduct(@RequestBody Product product);
```

8.2.2.2 Get All Products

Get Request to get all the products from the system.

```
@GetMapping("/")
public ArrayList<Product> getProducts();
```

8.2.2.3 Get a Specific Product

Get Request to get a specific product by passing its ID in the URL.

```
@GetMapping("/{productId}")
public Product getProductById(@PathVariable UUID productId);
```

8.2.2.4 Update the Product

Put Request to update the product by passing its ID in the URL with specific body in the request body.

```
@PutMapping("/update/{productId}")
public Product updateProduct(@PathVariable UUID productId, @RequestBody Map<String,Object>
    body);
```

8.2.2.5 Apply Discount to Products

Put Request that takes the discount amount in percentage and a list of product Ids. It should update the price of these products

```
@PutMapping("/applyDiscount")
public String applyDiscount(@RequestParam double discount,@RequestBody ArrayList<UUID>
    productIds)
```

8.2.2.6 Delete Product

Delete Request to delete a specific product by passing its ID in the URL.

```
@DeleteMapping("/delete/{productId}")
public String deleteProductById(@PathVariable UUID productId);
```

8.3 CartController

To handle the cart's endpoints (The Product router)

8.3.1 Class

```
@RestController
@RequestMapping("/cart")
public class CartController {

    //The Dependency Injection Variables
    //The Constructor with the requried variables mapping the Dependency Injection.

}
```

8.3.2 Required Endpoints

8.3.2.1 Add Cart

Post Request to add a new Cart in the system.

```
@PostMapping("/")
public Cart addCart(@RequestBody Cart cart);
```

8.3.2.2 Get All Carts

Get Request to get all the carts in the system.

```
@GetMapping("/")
public ArrayList<Cart> getCarts();
```

8.3.2.3 Get a Specific Cart

Get Request to get a specific cart by passing its ID in the URL.

```
@GetMapping("/{cartId}")
public Cart getCartById(@PathVariable UUID cartId);
```

8.3.2.4 Add Product to Cart

Put Request to add a product that is passed through the request body to the cart with its ID passed in the URL.

```
@PutMapping("/addProduct/{cartId}")
public String addProductToCart(@PathVariable UUID cartId, @RequestBody Product product);
```

8.3.2.5 Delete Cart

Delete Request to delete a cart by passing its ID in the URL.

```
@DeleteMapping("/delete/{cartId}")
public String deleteCartById(@PathVariable UUID cartId);
```

8.4 OrderController

To handle the order's endpoints (The Order Router)

8.4.1 Class Definition

```
@RestController
@RequestMapping("/order")
public class OrderController {

    //The Dependency Injection Variables

    //The Constructor with the requried variables mapping the Dependency Injection.

}
```

8.4.2 Required Endpoints

8.4.2.1 Add Order

Post Request to add a new Order to the system

```
@PostMapping("/")
public void addOrder(@RequestBody Order order);
```

8.4.2.2 Get a Specific Order

Get Request to get a specific order by passing its ID in the URL

```
@GetMapping("/{orderId}")
public Order getOrderById(@PathVariable UUID orderId);
```

8.4.2.3 Get All Orders

Get Request to get all the orders in our system

```
@GetMapping("/")
public ArrayList<Order> getOrders();
```

8.4.2.4 Delete a Specific Order

Delete Request to delete a specific order by passing its ID in the URL

```
@DeleteMapping("/delete/{orderId}")
public String deleteOrderById(@PathVariable UUID orderId);
```

---

9 Docker

Create dockerfile for your project and handle mounting the data.json files as explained in the labs. Also you should change the environment variables in the Dockerfile to be the new path of the json files in the container

---

10 Test Cases

For each API that you will write you have to write ***3 Unit Test cases for each Service that you will implement. You have in the project 25 Service which means you are required to implement 75 unit test*** cases in a new test case file other than the public commented ones. To run the public API tests, uncomment the code in the test file then run mvn test in terminal

---

---

11 Submission

Create a repository for your Project with separate branches for each team member with the Main branch for integrating your work. Your repository name should be Your Team Number - your Team name (e.g. 100-Random_01). Then add *****Scalable-Submissions***** OR *****Scalable2025@gmail.com***** as colaborator to your repository. Your history of commits will be monitored as an indication for the participation for all the team members so make sure you divide the work equally among you.