Project Overview:  
The project is an e-commerce application that consists of three microservices: wallet, shop, and inventory. The wallet microservice is responsible for managing user wallets, the shop microservice is responsible for managing the online store, and the inventory microservice is responsible for managing product inventory.

Technology Stack:

* Spring Boot
* Spring Cloud
* Spring Data JPA
* MySQL
* Maven

Wallet Microservice:

* User registration and login
* Wallet creation and management
* Deposit and withdrawal of funds
* Transaction history
* RESTful API endpoints

Shop Microservice:

* Product creation and management
* Shopping cart management
* Order creation and management
* Payment processing
* RESTful API endpoints

Inventory Microservice:

* Product creation and management
* Inventory management
* RESTful API endpoints

Steps to Implementation:

1. Create a new Spring Boot project using the Spring Initializr and add the necessary dependencies for Spring Cloud, Spring Data JPA, and MySQL.
2. Set up the database schema for the three microservices.
3. Implement the wallet microservice, including the necessary entities, repositories, services, and controller. Test the microservice using Postman or a similar tool.
4. Implement the shop microservice, including the necessary entities, repositories, services, and controller. Test the microservice using Postman or a similar tool.
5. Implement the inventory microservice, including the necessary entities, repositories, services, and controller. Test the microservice using Postman or a similar tool.
6. Implement the communication between the microservices using Spring Cloud, including service discovery using Eureka server, and Feign client to call APIs between microservices.
7. Configure and run the Eureka server.
8. Configure the microservices to register themselves with the Eureka server.
9. Configure the Feign client to call APIs between microservices.
10. Implement resilience circuit breaker using Resilience4j.
11. Configure the Resilience4j dashboard to monitor the circuit breaker.
12. Test the circuit breaker by simulating a fault scenario.
13. Implement Spring API Gateway to route requests to the appropriate microservice.
14. Configure the API Gateway to use service discovery and load balancing.
15. Implement Spring Config Server to manage configuration files for the microservices.
16. Configure the microservices to use the Spring Config Server to retrieve their configuration files.
17. Test the entire application end-to-end.