**Summary of API Development**

**Overview**

In this project, we have developed a robust microservices architecture consisting of three key services: **Account Service**, **Transaction Service**, and **Notification Service**. This architecture leverages modern practices such as JWT for security, Eureka for service discovery, Spring Boot's load balancing capabilities, and distributed tracing with Zipkin to ensure high availability, resilience, and observability.

Developed using Spring Boot 3 and Java 17.

GitHub link: <https://github.com/shyamhan/bank-app>

**Microservices**

**1. Account Service**

**Responsibilities:**

* **Account Creation:** Handles the creation of new accounts.
* **Login:** Manages user authentication.
* **Deposit and Withdraw:** Facilitates financial transactions including deposits and withdrawals.
* **Check Balance:** Provides account balance information.

**Security:**

* **JWT Implementation:** Used for authentication and authorization, ensuring that only authorized users can access certain functionalities.
* **Inter-Service Communication:** Utilizes JWT tokens to securely communicate with other services.

**2. Transaction Service**

**Responsibilities:**

* **Transfer Money:** Provides an API to transfer funds between accounts. This operation involves:
  + **Two-Phase Commit Protocol:** A robust mechanism that first prepares the transaction by calling the Account Service to prepare withdraw and deposit operations. Once both operations are ready, it proceeds with the commit operation to finalize the transaction.
* **Transaction History:** Allows users to retrieve their transaction history.

**Key Features:**

* **Fallback Mechanism:** Ensures that the service remains functional even if some operations fail.

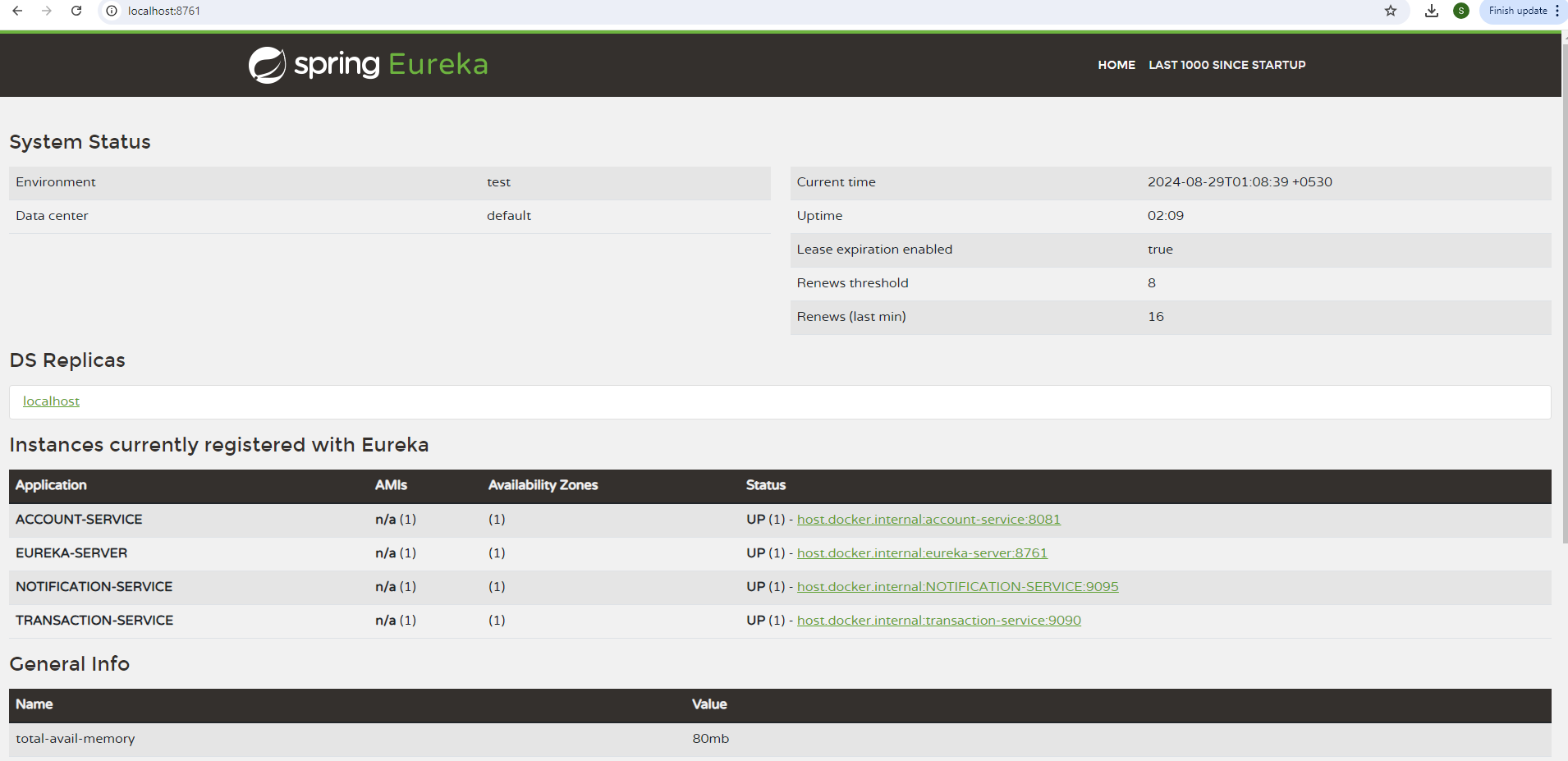
**3. Notification Service**

**Responsibilities:**

* **Send Notifications:** Sends email notifications to account holders about transaction activities using SMTP (smtp.gmail.com).

**Service Discovery and Load Balancing**

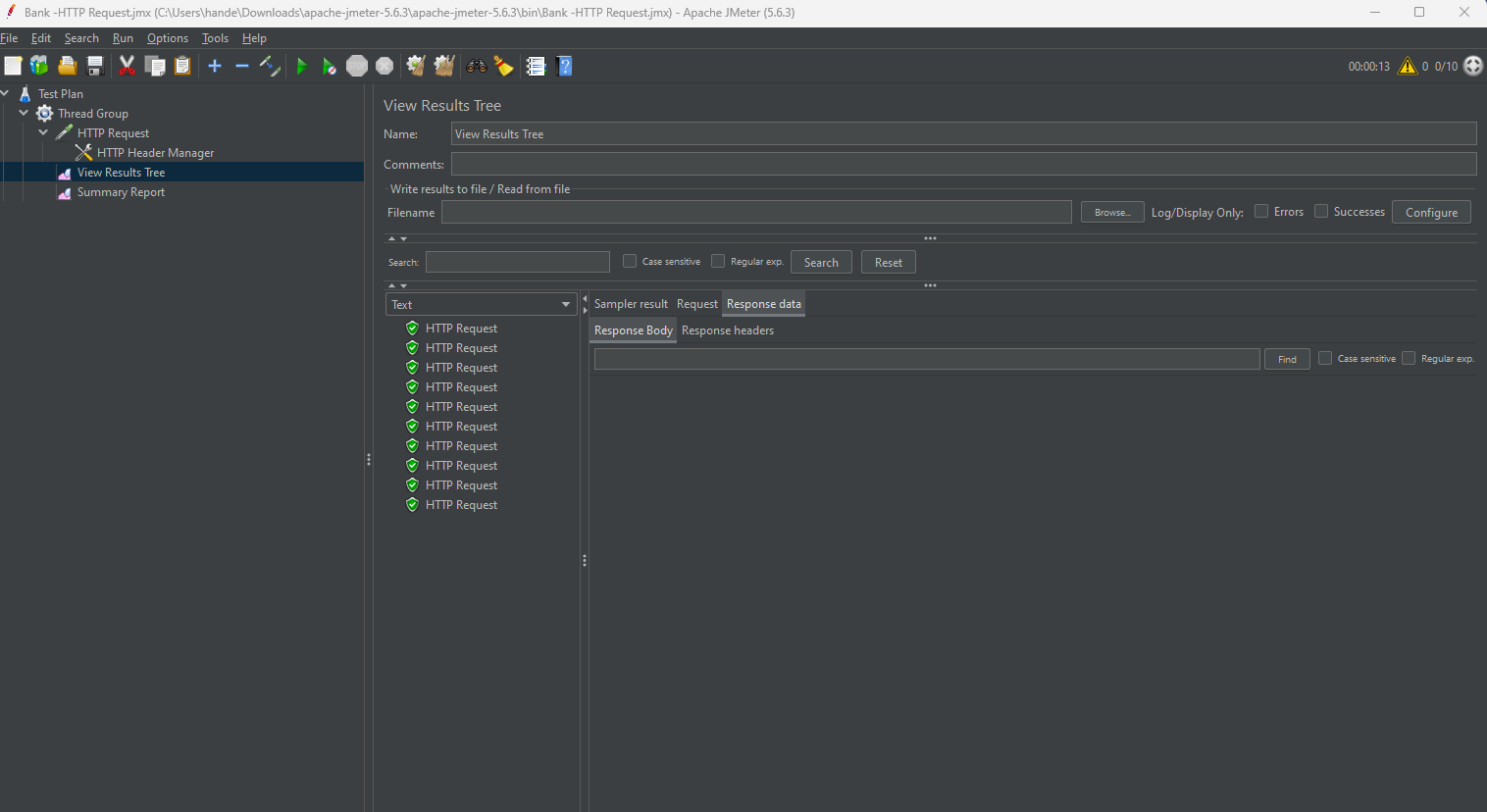
* **Eureka Server:** Acts as the service registry where all three services are registered. This enables dynamic discovery of services and load balancing.
* **Client-Side Load Balancing:** Handled by Spring Boot's load balancer library, distributing requests across multiple instances of the services to manage traffic efficiently.
* **Find below screenshot for ref:**



**Security and Resilience**

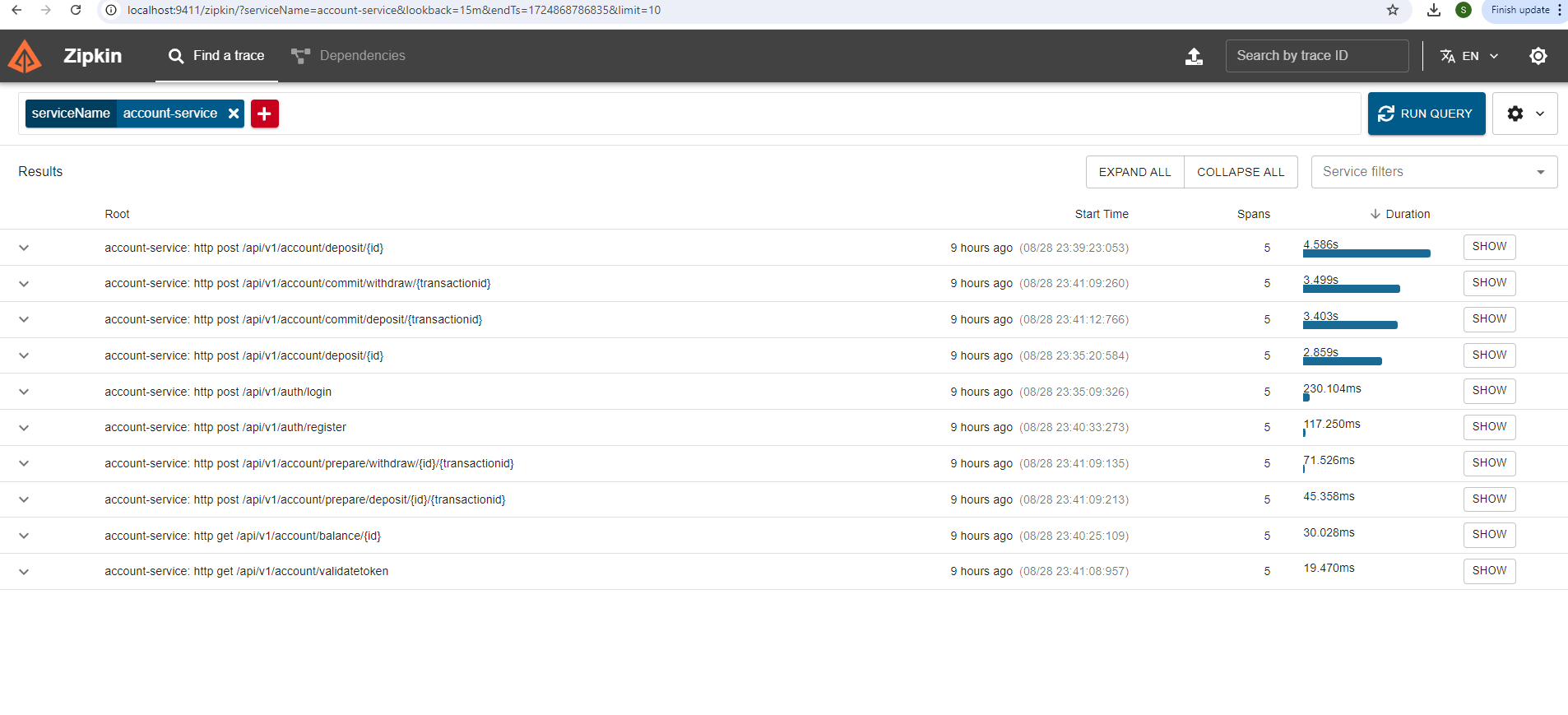
* **JWT Tokens:** Ensure secure communication between services by embedding authentication and authorization information in the tokens.
* **Two-Phase Commit Protocol:** Guarantees data consistency during financial transactions by preparing and committing operations in a coordinated manner.

**Testing and Validation**

* **JUnit Tests:** Basic JUnit tests have been implemented for services to ensure that the core functionalities and edge cases are handled correctly. These tests cover scenarios such as account creation, transaction processing, and email notifications.
* **Load Balancing Tests:** The load balancing capabilities of the system have been validated using JMeter. This testing tool helps simulate various load conditions and ensures that the client-side load balancing effectively distributes traffic across multiple service instances, maintaining performance and reliability.
* Used Postman tool to validate the API working and its available for demo from local system.
* Sample summary result from JMeter, executed 10 requests in 5 seconds timespan:
* 

**Distributed Tracing**

* **Zipkin Integration:** Distributed tracing is enabled using Zipkin to monitor and trace the flow of requests across microservices. This helps in identifying performance bottlenecks, tracking request paths, and understanding service dependencies. The tracing information is collected and visualized using the Zipkin UI, allowing for efficient debugging and performance monitoring.



**Conclusion**

This microservices architecture is designed to be scalable, resilient, and secure, with a focus on robust transaction management, secure inter-service communication, and comprehensive observability. The integration of JWT for security, Eureka for service discovery, Spring Boot's load balancing capabilities, and Zipkin for distributed tracing ensures that the system can handle varying loads, maintain high availability, and provide deep insights into its operation. Additionally, the implementation of JUnit tests and load balancing tests using JMeter ensures the reliability and performance of the system under real-world conditions.