**Financial Master – Financial Management System**

**1. TITLE OF THE PROJECT:**

Financial Master – Financial Management System

**2. OBJECTIVE OF THE PROJECT:**

This main objective of the system is to help users to purchase, oversee and govern its income, expenses, and financial assets with the objectives of maximizing profits and ensuring sustainability.

**3. LANGUAGE AND SOFTWARE TOOL USED**:

**Server Side:** Java, Spring Boot 2.1

**Database:** MySQL 8.x, Redis

**Front End:** React 16.8, Bootstrap

**Operating System:** Windows 10

**Environment:**

Java1.8, Spring Boot 2.1, Hibernate 5.0, MySQL 8, React 16.8, Spring AOP, Spring Security, Spring Cloud, JWT, RSA, Quartz, RESTful Web Service, JSON-RPC based APIs, Choreography based Saga, Hazelcast, Active, Bootstrap3, HTML5, CSS3, GIT, Maven, JUnit 5, Mockito 2, Docker, Jenkins

**4. STRUCTURE OF THE PROJECT:**

The Server-Side of Financial Management System mainly consists of two parts, each part consists of many microservices.

The first part is Management End where financial company employee can publish, update, delete new financial assets. The second part is Seller End where venders can use it to sell financial assets products to customers and users can redeem their financial assets.

Two main parts of the Server-Side can communicate with each other through Internal APIs using RESTful APIs, JsonRPC and Messaging system.

The Client-Side of Financial Management System consists of System UI implemented by React.

**Swagger**

**Seller1**

**Management End**

**API Gateway**

**Seller2**

**Cache**

**Front End**

**ActiveMq**

**JsonRPC**

**Seller2**

**Databse**

The Overall structure of the system is like above. We have API gateway to protect our Server-Side Application and conduct load balancing. We can have many seller instances we want and seller modules can communicate with management module through JMS or JsonRPC. We also have cache in order to optimize our system. We also integrated Swagger as an API documentation tool.

**5.2. Module Description**

**5.1 Management Module:-**

This is one of the main module in the project. In Management Module, the financial company employee can publish new financial products, update and delete current products. So we need to implement Service Layer, Data Access Layer and API Controller inside this module. We need to connect to Database in Management Module too. Besides that, we also implement common error handling and unit testing for the module.

**5.2. Entity Module:-**

This module contains the definition of the data we need to process like the financial product, the order record, the payment record, the user information, etc. We will use the data defined in this module to map to database using ORM with the help of Spring Data JPA and Hibernate.

**5.3. Util Module:-**

This module contains the tools we need to use for our development. The tools includes Json Format Convertor, RSA Signature Verification, Rest API Response Format Convertor and common configuration, etc.

**5.4. API Documentation Module:-**

This projects is a microservices-based architecture. We need to define a detailed API documentation. We used Swagger API to implement API documentation. Having a good API documentation not only will increase efficiency of development process, also will make it easier for future optimization.

**5.5. Seller Module:-**

Seller Module is another main module in the project. In Seller Module, vendors can use this module to sell financial products to their customers and customers can loop up, purchase, management and redeem their financial assets. Like Management Module, we need to implement Service Layer, Data Access Layer and API Controller inside this module. Besides that, we also need to implement the communication mechanism that allows Seller Module to talk to Management Module using JsonRPC and JMS ActiveMQ or Kafka. We also need to have mechanism to implement Cache between Seller Module and Management Module and the way to update our cache.

**5.6 Scheduler Module:-**

In this project, the communication between microservices is critical and complex. We need to ensure all microservices communicate and work properly with each other. The financial business domain requirement requires us to perform certain operations on time otherwise it would be a lot loss for both company or customers. In this Financial Management System, we have two major areas that need to be updated or performed periodically. One of them is the cache refresh. We can use scheduler to update cache periodically, probably when the customer access is infrequent like at night. Another major area is reconciliation between seller module and management module (between vendors and financial company). They need to do reconciliation periodically to make sure all transaction is complete and no transaction is lost. In order to optimize the system, we only do the reconciliation periodically which is at the end of the day. In the project, we use Quartz to implement the Job Scheduler.

**5.7 API Gateway Module:-**

In order to make our system secure and load balanced, we integrated API Gateway into our system. We employed API Gateway to implement authentication and authorization, API allows us to provide services without exposing Server Side. We also used API Gateway to implement load balancing between our services. We can limit the access rate using API Gateway too in case someone use DDoS to attack our system. Those are three major reasons we used API Gateway but there are more reasons other than those three. In these project, we used TYK as API Gateway. We will update it to Spring Cloud Zuul in the future in order to better make it compatible with other Spring Cloud Service like Eurake, Hytrix, Ribbon.

**6. Database Design**

In this project, we used MySQL as our main database. One database is connected to our Management Module. Seller service also have it own database. When we are doing the reconciliation, the database in Management Module and the database in Seller Module will work with each other to make it happen. In order to optimize the database retrieve and create, we employed Read/Write Splitting technique.

We also integrated Hazelcast as our system cache in order to optimize our system performance. We need to make sure the data in cache is always up-to-date. For that, we used JMS to asynchronously update the cache when the data in the database changed. We also used Quartz to periodically update the cache.

**7. Future Improvement of the System**

We can make some of optimization for the system in the future. First, we will employ more Spring cloud technology to replace some components like TYK. Furthermore, The Financial Domain requires our service to be responsive and robust. For that, we can employed Reactive Architecture into our system and make our system become fully event-driven system. That’s where we will discuss and head for the optimization in the future.

**8. SOFTWARE REQUIREMENT**

Java 8

Spring Boot 2.1

Gradle

MySQL 8.x

AvtiveMQ/Kafka

TYK

React 16.8

IntelliJ

**9. Github Link**

[**https:/github.com/NickLi19/financial-master**](https://github.com/NickLi19/financial-master)