eWallet Application

"GlobalPay is an established company with an existing monolithic e-wallet system. Our current system handles basic functionalities like user accounts, balance management, and simple transfers. However, we're facing challenges with scalability, adding new features, and maintaining the codebase. We've decided to modernize our system using domain-driven design and microservices architecture. As a Solution Architect, your task is to propose a strategy for this transformation.

Current System Overview:

- Monolithic Java application using Spring Framework
- Single MySQL database for all data
- Handles about 100,000 daily transactions but struggles during peak times
- Tightly coupled modules for user management, account balances, and transfers

Your task is to design a roadmap for transforming this monolithic system into a modern, scalable microservices architecture. Please address the following points:

1. Analyze the existing system and identify the core domains that could be separated into microservices. Consider aspects like user management, account management, transaction processing, payment gateways, and potential new features like multi-currency support and advanced security.
2. Propose a microservices architecture based on these domains. Explain which services you would create, how they would interact, and how they would replace parts of the monolithic system.
3. Outline a step-by-step migration strategy. How would you gradually move from the monolith to microservices while ensuring continuous operation?
4. For one key microservice of your choice, provide more detailed implementation recommendations. Discuss API design, data migration from the monolith, and any specific Java libraries or frameworks you'd recommend.
5. Describe your database design strategy for the new microservices. How would you handle data that's currently in a single MySQL database? Consider data consistency, transaction management, and performance optimization.
6. Explain how your proposed architecture would improve scalability, availability, and throughput compared to the current monolithic system. Address how you'd handle potential issues during and after the migration.

How would you approach this transformation from a monolith to a microservices-based e-wallet system? Please provide a high-level architecture diagram if possible, showing both the current monolithic state and the target microservices architecture."

While focusing on an e-wallet system. It covers:

1. Domain-driven design concepts in the context of financial services
2. Microservices architecture for a fintech application
3. Implementation details (focusing on Java)
4. Database design for financial transactions and user data
5. High availability, throughput, and scalability for a financial platform
6. Deployed into Alibaba Cloud and made sure of Alibaba Cloud Service
7. The need to maintain system stability during the migration process (zero downtime)

Please prepare an Overall Solution Analysis document with the template below:

Tips:

- **Target audience:** *Product designer, developer, and QA.*
- This document will be a guideline for the developer to prepare the detailed solution analysis and QA to prepare the test case analysis.
- Try to maximize the usage of diagrams (UML) to explain the entire design.
- If there are some requirements are not sure about, please continue with own assumptions

---

1. Requirement Analysis

<explain the requirement in whichever format, ideally is mindmap>

2. Architecture Overview

< high-level infra architecture diagram>

< high-level implementation plan>

3. Overall Process Flow

<Focus on accounting, required functionality e.g. reload, payment, refund>

<sequence diagram to explain high-level process flow>

<fund flow diagram if involves balance movement, if needed>

3.1 Application Process Flow

<sequence diagram to explain application integration>

3.2 Exception Flow / Handling

<sequence diagram to explain which integration point might introduce exceptions or fund loss if any>

<table to list down all the potential system-level exception scenarios and the SOP to follow when that happens>

<this section identifies and lists the potential exceptions and how to handle them.>

4. Implementation Analysis

4.1. Implementation Strategy

<what kind of implementation plan to roll out the new system without minimum impact on the existing user >

4.2. Monitoring

<what kind of monitoring is required, from the application, data consistency, etc>

4.3. Rollback

<how to rollback if new system rollout and found the issue>