Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

This document outlines the software requirements for a Microservices Architecture project using Spring Boot, Docker, and Kubernetes. The goal is to develop a scalable, resilient, and production-ready application that adheres to microservices principles and follows a modular approach to service management.

1.2 Scope

The project will encompass the following:

- Creation and management of microservices for **Accounts**, **Cards**, and **Loans** services, among others.
- Containerization using Docker for deployment and orchestration with Kubernetes.
- Service discovery, routing, and security management across microservices.
- Enhanced monitoring, observability, and resilience of services in production.

1.3 Definitions, Acronyms, and Abbreviations

- **Microservices**: Architectural style that organizes an application as a collection of loosely coupled services.
- **Spring Boot**: A framework for building production-ready, stand-alone Spring applications.
- **Docker**: A containerization platform for building and deploying applications in containers.
- **Kubernetes (K8s)**: System for automating the deployment, scaling, and management of containerized applications.
- **RabbitMQ**: Message broker for inter-service communication.
- **Kafka**: A distributed event streaming platform.

2. Overall Description

2.1 Product Perspective

This project will be deployed as a cloud-native application with Kubernetes orchestrating microservices. The architecture leverages Spring Boot and Spring Cloud to facilitate seamless integration, scalability, and modularity across services like **Accounts**, **Cards**, and **Loans**.

2.2 Product Functions

- Microservice setup with **Accounts**, **Cards**, **Loans**, and other components.
- Configuration management using **configserver**.
- Service discovery and registration via **eurekaserver**.
- Routing and gateway functionality managed by **gatewayserver**.
- Audit logging, exception handling, and transaction processing for each service.
- Deployment of microservices in Docker containers and Kubernetes clusters.
- Event-driven communication through RabbitMQ and Kafka.

2.3 User Classes and Characteristics

- **Developers**: Develop and maintain microservices, manage Git repositories for each service.
- **DevOps Engineers**: Deploy, scale, and manage container orchestration in Kubernetes.
- **System Administrators**: Oversee application security and monitoring.

2.4 Operating Environment

The application will run in a cloud environment with the following requirements:

- Docker version 20.10 or later.
- Kubernetes version 1.18 or later.
- Java Development Kit (JDK) version 11 or later.
- Spring Boot version 2.5 or later.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Microservices Development

- **FR-1**: The system shall provide separate repositories (accounts, cards, loans) and core folders: audit, constants, controller, dto, entity, exception, functions, mapper, repository, and service.
- **FR-2**: The controller layer shall handle incoming requests and route them to the appropriate service layer.
- **FR-3**: The repository layer shall handle database operations.
- **FR-4**: The dto and entity layers shall define data models and transfer objects.

3.1.2 Containerization

- **FR-5**: The system shall allow Docker images to be built for each service.
- **FR-6**: Docker containers shall be managed using Docker Compose for local development.

3.1.3 Configuration Management

- **FR-7**: The configserver shall manage externalized configuration for each service.

3.1.4 Service Discovery

- **FR-8**: The eurekaserver shall enable service registration and discovery.
- **FR-9**: Each microservice shall register itself with the Eureka server.

3.1.5 Event-Driven Architecture

- **FR-10**: RabbitMQ and Kafka shall be integrated for asynchronous event handling between microservices.

3.1.6 Security

- **FR-11**: Keycloak shall provide authentication and authorization across services.

3.1.7 Monitoring and Observability

- **FR-12**: Prometheus shall collect metrics across services.
- **FR-13**: Grafana dashboards shall visualize key metrics for system health.

3.1.8 Deployment

- **FR-14**: The system shall deploy microservices to a Kubernetes cluster using Helm charts.
- **FR-15**: The application shall scale microservices dynamically based on load.

3.2 Non-Functional Requirements

3.2.1 Performance Requirements

- **NFR-1**: The application shall support at least 1000 concurrent users across services.

3.2.2 Security Requirements

- **NFR-2**: All APIs shall be secured with OAuth2.

3.2.3 Usability Requirements

- **NFR-3**: Each service shall include API documentation for usage and deployment.

3.2.4 Reliability Requirements

- **NFR-4**: The application shall provide 99.9% uptime.

4. External Interface Requirements

4.1 User Interfaces

- The application shall provide web-based and API-based interfaces for interacting with microservices.

4.2 Hardware Interfaces

- The application will run in a cloud-based environment, requiring no specific hardware interfaces.

4.3 Software Interfaces

- Integrations with the following:
    - **Spring Boot**: For microservice creation.
    - **Docker**: For containerization.
    - **Kubernetes**: For orchestration and scalability.
    - **RabbitMQ**: For event-driven messaging.
    - **Kafka**: For distributed event streaming.

5. Appendices

5.1 References

- Spring Boot Documentation
- Docker Documentation
- Kubernetes Documentation
- RabbitMQ Documentation
- Kafka Documentation