**Case Study Document: Spring Boot REST API with JUnit, Integration with Kafka, and Jenkins**

**Table of Contents**

1. Introduction

- 1.1 Purpose

- 1.2 Scope

- 1.3 Technologies Used

2. Business Scenario

- 2.1 Background

- 2.2 Problem Statement

- 2.3 Objectives

3. System Architecture

- 3.1 High-Level Overview

- 3.2 Components

- 3.3 Data Flow

4. Features

5. Implementation Steps

6. Conclusion

**1. Introduction**

**1.1 Purpose**

The purpose of this case study is to build an Ordering API for Consumer Customer using Spring Boot, integrate it with Kafka for real-time messaging, and set up continuous integration with Jenkins. The API will allow consumers to create, modify, list, and view orders associated with their customer ID.

**1.2 Scope**

The scope of this project includes the development of RESTful endpoints for managing orders related to consumer customers. Additionally, integration with Kafka will enable real-time event processing for order-related events.

**1.3 Technologies Used**

The technologies used in this project are as follows:

**- Spring Boot**: For building the RESTful API.

**- JUnit:** For writing unit tests to ensure code quality.

**- Apache Kafka:** For real-time messaging and event processing.

**- Jenkins:** For continuous integration and automated builds.

**2. Business Scenario**

**2.1 Background**

In the context of an e-commerce platform, consumer customers often need to place and manage their orders efficiently. The Ordering API will provide the necessary functionalities to create new orders, modify existing orders, and view order details for a specific consumer customer.

**2.2 Problem Statement**

The existing system lacks a centralized and efficient way for consumer customers to manage their orders. There is a need for a robust and scalable Ordering API that can handle real-time order-related events and provide seamless interactions with the e-commerce platform.

**2.3 Objectives**

**The main objectives of the project are as follows:**

1. Design and implement RESTful endpoints for order management.

2. Write comprehensive unit tests using JUnit to ensure the correctness of the API.

3. Integrate Kafka to handle real-time order events and notifications.

4. Automate the build and deployment process using Jenkins for continuous integration.

**3. System Architecture**

The system architecture will follow a layered approach:

- Presentation Layer: Handles HTTP requests and responses for RESTful endpoints.

- Business Logic Layer: Contains business logic for order management operations.

- Data Access Layer: Communicates with the database to retrieve and update order data.

- Kafka Integration Layer: Facilitates communication with Kafka topics for real-time events.

**4. Features**

The Ordering API will support the following features:

1. Create an order for a specific consumer customer.

2. Update an existing order associated with a customer ID and order ID.

3. Retrieve all orders for a specific consumer customer.

4. View details of a specific order by order ID.

5. Check the status of an order by order ID.

**5. Implementation Steps**

1. Create a new Spring Boot project and set up the necessary dependencies, including Kafka integration and JUnit testing.

2. Design and implement the required RESTful endpoints for order management.

3. Write comprehensive unit tests using JUnit to verify the correctness of each endpoint.

4. Integrate Kafka to handle order-related events and notifications.

5. Implement event publishing and consumption using Kafka topics.

6. Set up Jenkins for continuous integration and automated builds.

7. Configure Jenkins to trigger builds and tests on every code commit or pull request.

8. Monitor the build status and code coverage through Jenkins dashboard.

By following these implementation steps, we will achieve a robust and efficient Ordering API with real-time event processing capabilities and automated continuous integration for enhanced code quality and deployment efficiency.

**Conclusion**

In this case study, we have explored the process of building a Spring Boot REST API for order management, integrating it with Kafka for real-time event processing, and setting up continuous integration using Jenkins. The Ordering API will empower consumer customers to manage their orders seamlessly while enabling the e-commerce platform to handle real-time order events effectively. This implementation demonstrates the power of Spring Boot, Kafka, and Jenkins in developing reliable and scalable APIs for modern applications.