Polyglot Distributed SMS Service

1. Goal

The objective of this assignment is to refactor an existing monolithic SMS Notification Service into two separate, communicating microservices: an **SMS Sender** written in **Java (Spring Boot and Kafka Event Producer)**, and an **SMS Store** written in **GoLang (Standard HTTP and Kafka Event Consumer)**.

This exercise provides hands-on experience with:

1. Designing and implementing **RESTful APIs** in two different programming languages.
2. Implementing **inter-service communication** (Java client calling GoLang server).
3. Working with language-specific frameworks (Spring Boot) and standard libraries (net/http).
4. Understanding service contracts, data flow, and polyglot persistence in a distributed system.

---

2. Architecture and Flow

The system will consist of two core services interacting to complete the SMS sending and logging process.

Architecture Overview

Service Name	Primary Language / Framework	Role	Core Function
SMS Sender	**Java / Spring Boot**	Client/Gateway	Receives requests, calls the 3rd party SMS API, and sends events to the SMS Store service to log the event.
SMS Store	**GoLang / net/http**	Data Persistence	Receives SMS records from the Java service and stores them. Provides APIs for users to retrieve their history.

Data Flow

1. A client calls the **Java SMS Sender** API (`POST /v1/sms/send`).
2. The **Java Service** calls the 3rd party vendor API (e.g., `https://api.imiconnect.in/...`) to send the SMS.
3. The **Java Service** then calls kafka to send SMS event to the **GoLang SMS Store** to persist the record along with the status.
4. The **GoLang Service** stores the record and acks event.
5. The **Java Service** returns the final response after pushing to kafka.

---

3. Service 1: SMS Sender (Java)

This service acts as the gateway, handling external requests, interfacing with the external SMS vendor, and coordinating with the internal GoLang service and blocking SMS to any blocked users.

Technology Requirements

- **Language:** Java
- **Framework: Spring Boot**

Required APIs

API	Method	Endpoint	Description
Send SMS	`POST`	`/v1/sms/send`	Receives `phoneNumber` and `message`. Calls the 3rd party SMS API, then sends to the GoLang Store.

Implementation Details

- Use Spring Boot's **`RestTemplate`** **or** **`WebClient`** for making the synchronous HTTP call to the GoLang service.
- The service must correctly serialize the message data into a JSON format expected by the GoLang service.
- Implement comprehensive error handling for both the 3rd party vendor call and the GoLang service connection(Since 3P will not be available for this exercise mock such calls to 3P vendor with result status as SUCCESS or FAIL).
- A user block list will be kept in redis and before sending SMS this list needs to be checked to see if user is already blocked.

4. Service 2: SMS Store (GoLang)

This service is the core system of record for all messages, providing persistence and retrieval capabilities.

Technology Requirements

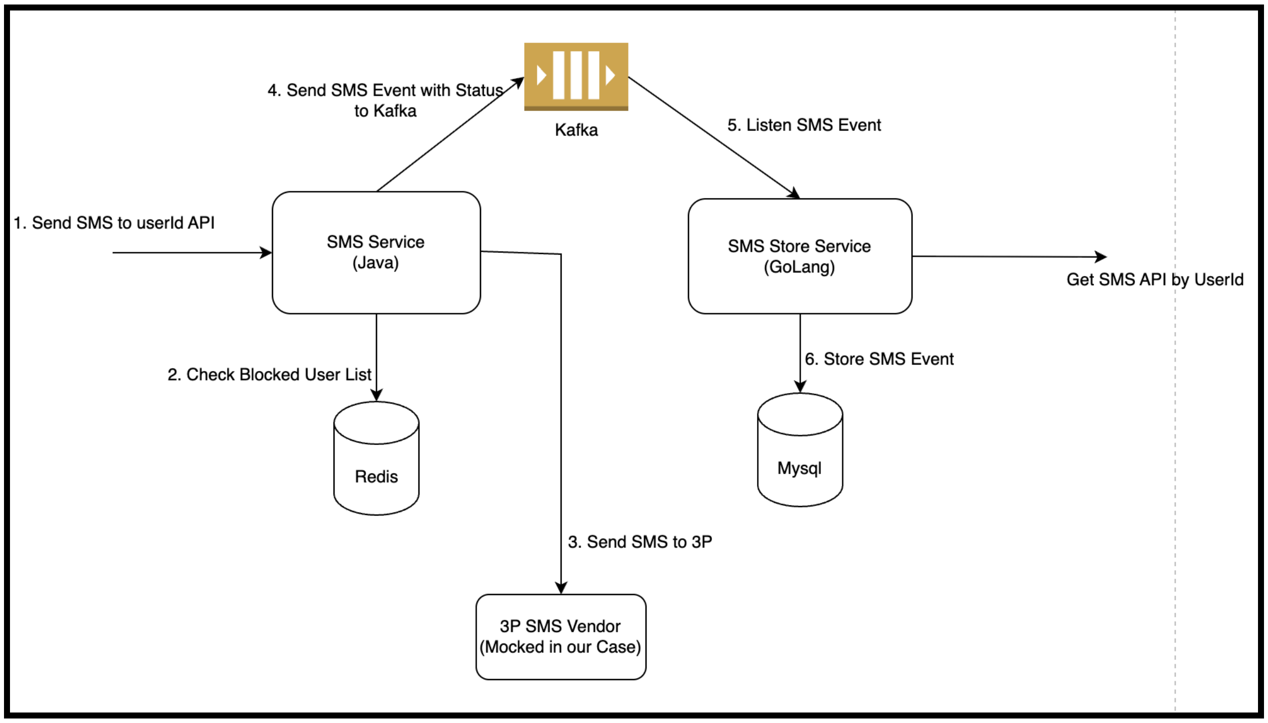
- **Language:** GoLang
- **Server:** Standard **net/http** package for building the server.
- **Persistence:** Use mongoDb to store the messages

Required APIs (Internal and External)

API	Method	Endpoint	Description
Get SMS History	GET	/v1/user/{User_id}/messages	**External/App-facing API.** Fetches all messages stored for a specific userId.

Implementation Details

- The Go service must use the standard library to define HTTP handlers.
- Define a clear Go struct to represent the SMS record to be stored.
- Ensure the service is well-structured using **idiomatic Go practices** (e.g., proper error handling, package structure).


4. Send SMS Event with Status to Kafka
Kafka
5. Listen SMS Event
1. Send SMS to userId API
SMS Service (Java)
SMS Store Service (GoLang)
Get SMS API by UserId
2. Check Blocked User List
6. Store SMS Event
Redis
Mysql
3. Send SMS to 3P
3P SMS Vendor (Mocked in our Case)

Note: While the diagram mentions mysql for the purpose of this exercise we will use mongoDb instead of mysql

5. Deliverables & Recommended Practices

Deliverables

1. **API Documentation:** A simple README.md file detailing the endpoints for both services and instructions on how to run them locally.
2. **Demonstration:** A script or documented steps demonstrating the full, end-to-end flow: Call the Java service, check the GoLang service's logs for the internal call, and finally retrieve the record using the GoLang service's history API.

Recommended Practices

- **Code Structure:** Organize code logically (handlers, services, models/structs).
- **Error Handling:** Implement robust logging and error handling, especially for inter-service communication timeouts or failures.
- **Testing:** Include basic **Unit Tests** for core business logic in both services.
- **Best Practices:** Follow language-specific best practices (Spring Boot conventions for Java and idiomatic Go for the Go service).