Notification Service – Architecture & Design Overview

This notification service is designed to handle incoming events from the hiring pipeline. It uses event-based technologies like Kafka to handle these events. The Kafka consumer consumes incoming hiring events and then sends out events using NestJS EventEmitter for them to be consumed by internal handlers for processing. It sends out notifications to appropriate recipients via different channels defined in settings and logs these notifications in PostgreSQL.

Lets go over the key components of this service and explain their purpose:

- **Kafka Consumer:** Listens for hiring events such as application received, interview scheduled, offer accepted etc.
- **NotificationsController:** For testing or manually triggering notifications in case Kafka goes down, and providing swagger documentation
- **EventEmitter2:** Dispatching events locally to appropriate handlers via @OnEvent()
- **Handlers:** Listen to internal events and forward them to appropriate adapters for processing
- **NotificationsFactory:** Holds the central logic for dispatching notifications and tells how and where to route them.
- **NotificationDeliveryService:** Responsible for channel and role based notification dispatching using adapters
- **Adapters:** Contain specific implementation according to that channel for delivery of notifications extending from a single NotificationChannel interface.
- **NotificationLogService:** Logs the notification into the postgres db to persist it for future use

Application Flow

The hiring pipeline sends out events whenever there is an update. Those events are consumed by the Kafka Consumer of our service. The consumer emits the event internally using EventEmitter2. We currently have 3 handlers implemented: Push for push notifications, Email for emails and Console for internal logging. The emitted event triggers all relevant handlers. These handlers then call the deliverViaChannel method of notificationsDeliveryService. This service makes use of the rules laid out in the routing map in notificationFactory to match the channels and recipients. The appropriate adapters then send out the notification. The NotifcationLogs Service persists each notification in the db.

Design Decisions

Kafka is used as a way to transport emits as it is easily scalable across services and provides high throughput. The use of NestJS EventEmitter2 allows decoupling of the event routing logic, which increases flexibility of the service and allows for more channels to be added with minimal changes. The use of adapters provide a common interface for

extensibility (i.e., simple to extend to include channels like Slack and SMS later). RoutingMap allows configurable routing based upon the eventType and the recipient's role. DTOs along with global pipes validate the data automatically and throw error in case of mismatch.