**Notification System – Overview**

This repository contains a **comprehensive microservices-based Notification System** built with Spring Boot. It is designed to support multiple notification channels (Email, Push, SMS, etc.) and clearly separates **user input & validation**, **content generation**, and **delivery** into distinct services.

**High-Level Flow**

[Client]

|

v

[API Gateway]

|

v

[userDetails] --(validates & enriches recipients)--> [notificationContentBuilding] --(generates content)--> [notificationSend] --(delivers)--> Email/FCM/APNs/SMS

[Service Discovery: eureka]

[Shared DTOs: input\_dto]

* **API Gateway**: Entry point for clients; handles routing to microservices.
* **Eureka**: Service discovery for dynamic registration of microservices.
* **input\_dto**: Shared library of DTOs used across services.

**Core Microservices:**

* userDetails: Accepts and validates incoming notification requests, manages user profiles, and enriches data for downstream services.
* notificationContentBuilding: Generates notification content (HTML for email, JSON for push) based on templates and payloads.
* notificationSend: Handles the actual delivery of notifications to various channels.

**Microservices Documentation**

**1. userDetails**

**Purpose**

The **first stage** in the pipeline. This service:

* Accepts notification requests from clients.
* Validates and enriches recipient/user data.
* Manages user profiles, including email addresses, phone numbers, FCM/APNs tokens.
* Returns clean, validated payloads to the content builder.

**Key Features**

* Stores user contact information securely.
* Provides recipient lookup to other services.
* Ensures only valid, existing users are notified.

**Endpoints**

* POST /v1/users — Create a new user with contact info.
* GET /v1/users/{id} — Retrieve user details.
* POST /v1/users/validate — Validate user/recipient payload before notification.

**Components**

* UserController for REST endpoints.
* UserService for business logic.
* UserRepository for persistence.

**2. notificationContentBuilding**

**Purpose**

The **second stage**. After userDetails validates the request, this service generates channel-specific content:

* HTML emails using Thymeleaf/Freemarker templates.
* JSON push notification payloads for FCM/APNs.
* Extendable to SMS, in-app, or other channels.

**Key Features**

* Accepts JSON payloads containing channel, templateId, data, and items.
* Merges dynamic data into predefined templates.
* Produces ready-to-send content objects (EmailContent, PushContent).

**Components**

* ContentGenerator interface.
* ThymeleafEmailGenerator implementation.
* PushGenerator implementation.
* REST Controller to accept payload and return generated content.

**Endpoints**

* POST /v1/notifications/content
  + **Request Body**: JSON describing channel, templateId, data, items, recipient.
  + **Response**: Generated content (HTML for email, JSON for push).

**Example Request**

{

"channel": "EMAIL",

"templateId": "order-summary",

"data": {

"subject": "Your Order Summary",

"greeting": "Hello John",

"orderId": "12345",

"total": "49.99"

},

"items": [

{"name": "Item A", "qty": 1},

{"name": "Item B", "qty": 2}

],

"recipient": {

"email": "john@example.com"

}

}

**3. notificationSend**

**Purpose**

The **final stage**. This service delivers notifications created by notificationContentBuilding to actual channels:

* SMTP email sending.
* Push notifications via FCM/APNs.
* Future SMS, WhatsApp, or Slack integration.

**Key Features**

* Receives pre-generated content and recipient details.
* Handles sending, retries, and error logging.
* Abstracted delivery channel services (EmailDeliveryService, PushDeliveryService).
* Can be extended with asynchronous queues.

**Components**

* EmailDeliveryService — sends via SMTP.
* PushDeliveryService — sends via FCM/APNs.
* Queue listener or REST controller for incoming send requests.

**Endpoints**

* POST /v1/notifications/send
  + **Request Body**: JSON containing channel content + recipient.
  + **Response**: Status of delivery attempt.

**Example Request**

{

"type": "EMAIL",

"subject": "Your Order Summary",

"htmlBody": "<!doctype html>...",

"recipient": {

"email": "john@example.com"

}

}

**Example Response**

{

"status": "SENT",

"timestamp": "2025-09-22T15:00:00Z"

}

**API Reference Summary**

* **/v1/users** — Manage and validate users.
* **/v1/notifications/content** — Generate content.
* **/v1/notifications/send** — Send notifications.

**Quick Start Guide**

This guide explains how to run and test the Notification System locally.

**Prerequisites**

* Java 17+
* Maven 3.8+
* MySQL/PostgreSQL (if persistence enabled)
* SMTP credentials (for email sending)
* FCM Server Key / APNs credentials (for push)

**Running Services Locally**

1. **Start Eureka Service Discovery:**
2. cd eureka
3. mvn spring-boot:run
4. **Start UserDetails service:**
5. cd userDetails
6. mvn spring-boot:run
7. **Start Notification Content Building:**
8. cd notificationContentBuilding
9. mvn spring-boot:run
10. **Start Notification Send service:**
11. cd notificationSend
12. mvn spring-boot:run
13. **Start API Gateway:**
14. cd APIGateway
15. mvn spring-boot:run

**Testing Notification Creation**

Send a test request to create content:

curl -X POST http://localhost:8080/v1/notifications/content \

-H "Content-Type: application/json" \

-d @sample\_request.json

**Testing Notification Send**

curl -X POST http://localhost:8080/v1/notifications/send \

-H "Content-Type: application/json" \

-d @sample\_send.json

**Adding a New Channel**

1. Implement a new ContentGenerator for your channel.
2. Add a DeliveryService for sending.
3. Register both in Spring context.
4. Update templates or payload format accordingly.

**Best Practices & Extension Points**

* Keep userDetails responsible only for validation and user data, not business logic of notifications.
* Use DTOs from input\_dto for consistent request/response objects.
* Add new templates in notificationContentBuilding to support more email formats.
* Use asynchronous queues (e.g., Kafka) between notificationContentBuilding and notificationSend for scalability.
* Add monitoring/logging for delivery success/failure in notificationSend.

This documentation gives a **holistic view** of the notification system, its microservices, and how to run, test, and extend it.