# Take-Home Technical Challenge

## Design and Build a Notification Service

### Objective

Design and implement a simplified Notification Service that can send and manage notifications to users through multiple channels such as Email, SMS, and Push. This challenge evaluates your ability to design scalable systems, structure clean architecture, and implement maintainable, production-ready code.

### Part 1 — System Design (Conceptual)

Requirements

Design a service that:
1. Receives notification requests from other systems (for example, Order Service, Identity Service).
2. Supports multiple channels: Email, SMS, and Push.
3. Allows adding new channels in the future (for example, WhatsApp, Slack).
4. Tracks delivery status (Pending, Sent, Failed).
5. Prevents duplicate sends (idempotency).
6. Supports configurable templates for different notification types (for example, 'Order Shipped', 'Password Reset').

Deliverables

Prepare a brief system design document (1–2 pages) that covers:
- Architecture diagram showing key components and their interactions.
- Chosen technologies and reasoning (frameworks, databases, queues, etc.).
- Data model (tables or collections for notifications, templates, logs).
- Flow description of a notification from creation to delivery.
- Scalability, reliability, and fault-tolerance considerations.
- How the design allows adding new channels easily in the future.

You may use any simple diagram tool such as draw.io, Excalidraw, or Mermaid.

### Part 2 — Implementation (Practical)

Task

Build a minimal working prototype of the Notification Service using any programming language and framework of your choice.

Requirements

1. Expose an API endpoint to create a notification request:
   POST /api/notifications
   Example request body:
   ```
   {
    "user_id": "123",
    "channel": "email",
    "template": "welcome_email",
    "data": {
     "name": "Bassam",
     "link": "https://example.com"
    }
   }
   ```
2. Validate input and queue or schedule the notification for processing.
3. Simulate sending the notification for each channel (you may log to the console or store in the database instead of actually sending).
4. Store each notification with its status in a database or persistent store.
5. Provide a status endpoint:
   GET /api/notifications/:id
   Returns the current status and details of the notification.
6. Include a short README that describes:
   - How to run the project locally
   - Major design decisions
   - What could be improved or added with more time

## Bonus (Optional)

- Use a message queue for asynchronous processing (for example, RabbitMQ, Kafka, or any alternative).
- Add retry logic for failed deliveries.
- Use configuration management for secrets and environment variables.
- Containerize the application using Docker.
- Include a few unit tests.

## Submission

- A public GitHub repository link