Design a Notification System

Problem Statement:

You are tasked with designing a Notification System that supports sending notifications to users across multiple channels. The system should be scalable, extensible, and able to handle a large volume of notifications.

Core Features:

1. **User and Notification Management**:
    - Users can register to receive notifications.
    - Notifications can be sent via multiple channels, such as **Email**, **SMS**, and **Push Notifications**.
    - Notifications can be personalized for individual users.
2. **Notification Scheduling**:
    - Notifications can be sent **immediately** or **scheduled** to be sent at a specific time in the future.
    - Support recurring notifications (e.g., reminders).
3. **Prioritization and Batching**:
    - Notifications should support **priority levels** (e.g., high, medium, low).
    - Batch notifications for efficiency, especially for large-scale campaigns.
4. **Channel Extensibility**:
    - New notification channels (e.g., WhatsApp, Slack) should be easily integrated without significant changes to the existing system.
5. **Failure Handling and Retries**:
    - If sending a notification fails (e.g., network issues, invalid contact details), the system should retry with exponential backoff.
    - Provide mechanisms to log and monitor failed notifications.

Constraints and Non-Functional Requirements:

- **Scalability**: The system must handle millions of notifications, potentially across different regions.
- **Real-Time vs. Scheduled**: Real-time notifications should have low latency, while scheduled notifications should handle large backlogs efficiently.
- **Fault Tolerance**: The system should gracefully handle failures and ensure eventual delivery.
- **Extensibility**: Adding new notification channels or formats should require minimal effort

Deliverables:

1. **Class/Component Design**:
    - Clear separation of concerns between notification generation, channel delivery, scheduling, and retry logic.
    - Interfaces or abstractions for channels to ensure easy extensibility.
2. **Notification Delivery Logic**:
    - Implementation of prioritization and scheduling.
    - Retry mechanisms for handling transient failures.
3. **APIs**:
    - RESTful APIs or equivalents for sending, scheduling, and tracking notifications.
    - Support for bulk notification APIs (if applicable).
4. **Database Design**:
    - Schema to store user preferences, notification history, and scheduling information.
5. **Unit Tests**:
    - Cover priority handling, channel selection, and retry logic.