PRD: Alerting & Notification Platform

Background

Modern organizations rely on timely alerts and notifications to keep their teams informed about important events: system outages, policy changes, reminders, and announcements. However, one common problem with notifications is that users often miss or ignore them if they are not persistent enough, while admins struggle to control who sees what.

Your task is to build a **lightweight alerting and notification system** that balances admin configurability with user control, while ensuring **clean OOP design, extensibility, and code clarity**.

This system should allow **Admins** to configure alerts, define visibility (organization, team, or user), and ensure recurring reminders every 2 hours until the user explicitly snoozes the alert for the day. **End Users** should receive relevant alerts, have the ability to snooze them, and keep track of their read/unread status.

This assignment will test your ability to design clean, modular systems, apply OOP principles effectively, and implement recurring logic without creating tightly coupled code.

Functional Requirements

Admin User

Admins should be able to:

1. **Create Alerts**
    - Create unlimited alerts.
    - Each alert must include:
        - Title
        - Message body
        - Severity: Info, Warning, Critical
        - Delivery type: In-App (MVP), future-proof for Email & SMS

        - Reminder frequency: Default **every 2 hours** until snoozed or expired
    - Configure visibility:
        - Entire Organization
        - Specific Teams (e.g., Engineering, Marketing)
        - Specific Users
2. **Manage Alerts**
    - Update or archive existing alerts.
    - Set start & expiry times.
    - Enable/disable reminders for an alert.
3. **View Alerts (as Admin)**
    - List all alerts created.
    - Filter by severity, status (active/expired), and audience.
    - Track whether alerts are still recurring or have been snoozed by most users.

End User

End Users should be able to:

1. **Receive Notifications**
    - Get alerts based on assigned visibility (org/team/user).
    - Alerts **re-trigger every 2 hours** until:
        - The user snoozes it for the day, OR
        - The alert expires.
2. **Snooze Alerts**
    - Snooze an alert for the **current day**.

    - Next day, if the alert is still active, reminders resume every 2 hours.

3. **View Alerts**
    - See a list of active alerts in a dashboard.
    - Mark alerts as **read/unread**.
    - Check snoozed alerts (history).

Shared Features

- **Analytics Dashboard**
    - Show system-wide metrics:
        - Total alerts created
        - Alerts delivered vs. read
        - Snoozed counts per alert
        - Breakdown by severity (Info/Warning/Critical)

MVP Scope

1. **Notification Channels (Required):**
    - In-App delivery
2. **Visibility Management:**
    - Org-level, Team-level, and User-level audience targeting
3. **Reminder Logic:**
    - Remind every 2 hours until snoozed or expired
    - Snooze resets next day

4. **Data Models (Suggested):**
    - **Alert** (title, message, severity, start, expiry, reminder frequency)
    - **User** (id, name, team)
    - **Team** (id, name)
    - **NotificationDelivery** (log of each alert sent to a user)
    - **UserAlertPreference** (read/unread/snooze state per alert per user)

Future Scope (Not required for MVP, but code should be extensible)

- **Email & SMS delivery channels**
- **Customizable reminder frequency** (not just 2h)
- **Scheduled alerts** (cron or specific times of day)
- **Escalations:** e.g., if not acknowledged in 24h → upgrade severity
- **Role-based access control** for Admin features
- **Push Notification integration**

Key Notes for Implementation

Code Structure

- Follow **DRY** and **SRP (Single Responsibility Principle)**.
- Use **OOP design principles** to keep modules extensible:
    - **Strategy Pattern** for notification channels
    - **Observer Pattern** for user subscription to alerts

    - **State Pattern** for snooze/read/unread logic
- Design so that **adding new delivery channels** does not break existing code.

Technical Excellence

- Implement reminders in a modular way:
    - Either simulate 2h reminders (e.g., via a cron-like scheduler)
    - Or provide APIs to "trigger reminders" for demo purposes
- Ensure separation between:
    - **Alert Management** (CRUD + visibility)
    - **Notification Delivery** (channels + scheduling)
    - **User Preferences** (snooze, read/unread states)

Deliverables

- **Backend codebase** with clear README (setup + usage instructions)
- APIs:
    - Admin: Create, Update, List Alerts
    - User: Fetch Alerts, Mark Read/Unread, Snooze Alert
- Seed data: Predefined users & teams for testing visibility
- Analytics endpoint with aggregated alert data

Evaluation Criteria

- **Object-Oriented Design**: Proper use of abstraction, encapsulation, inheritance, polymorphism

- **Code Cleanliness**: Readable, modular, reusable, well-documented
- **Extensibility**: Can easily add new channels, reminder frequencies, or visibility types
- **Correctness**: Functional coverage of the PRD requirements
- **Technical Depth**: Thoughtfulness in handling reminder logic & snooze management