Group Design Activity: Modeling a Real-time Notification system for a community event app

Group Members:

Mike Muyambango 2420983

Robert Sichizya 2410264

Israel Mumba 2420971

Member Roles:

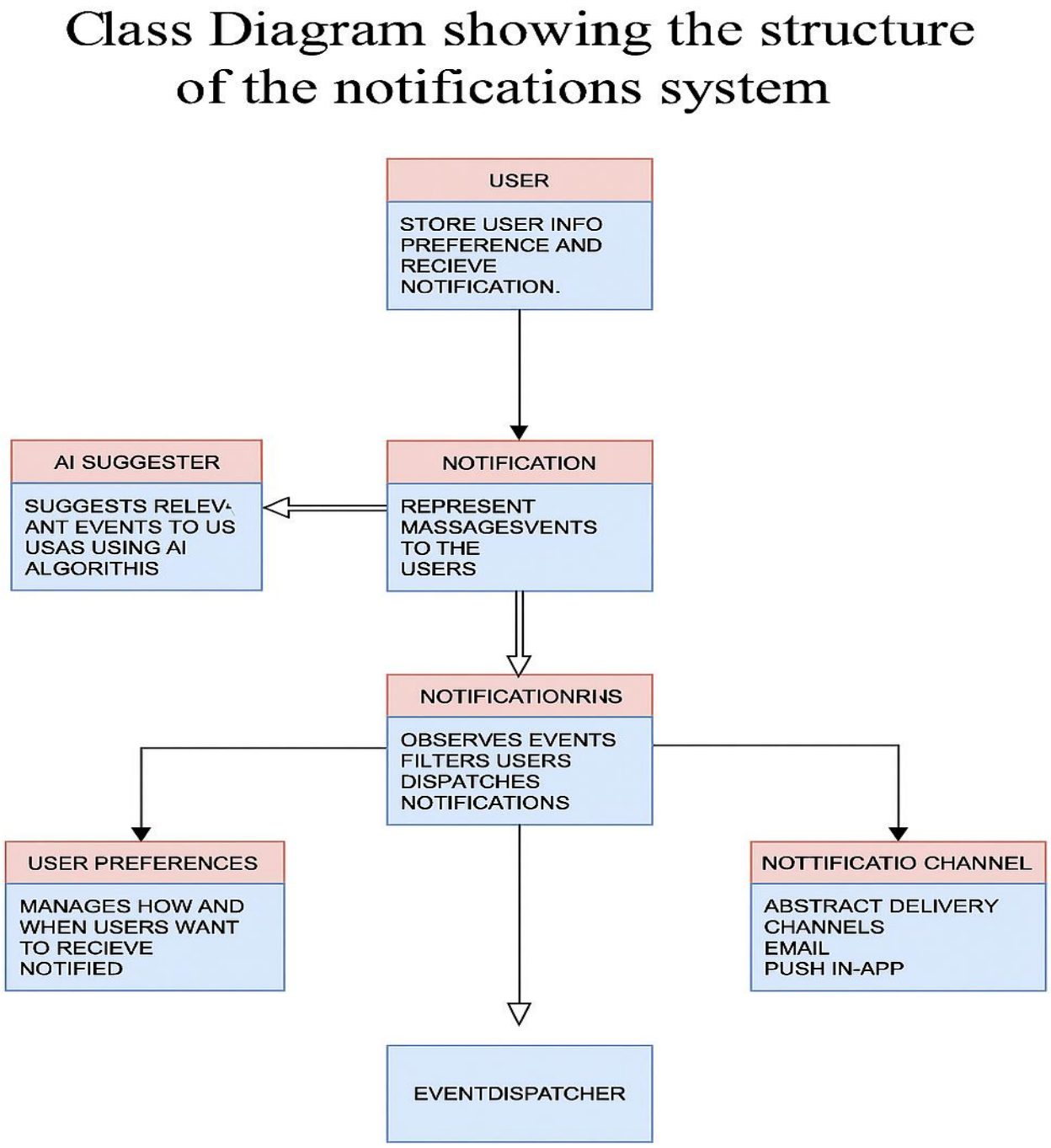
Member A (Mike); Documentation and Research

Member B (Robert); Class diagram design

Member C (Israel); Sequence diagram design

Class Diagram showing the structure of the notifications system

++_



USER

- Represents the individual using the system.
- Stores personal data and **notification preferences**.
- Is the final recipient of notifications.

AI SUGGESTER

- Uses machine learning to **analyze user behavior** and suggest relevant events.
- Helps tailor notifications to what the user might care about most

NOTIFICATION

- The actual message or alert sent to the user.
- Could be anything from a reminder, update, or promotional message.

NOTIFICATION SERVICE

- The **central brain** of the system.
- Watches for events (via the EventDispatcher).
- Filters users based on relevance and preferences.
- Sends out notifications through appropriate channels

USER PREFERENCES

- Defines **how**, **when**, and **what** the user wants to be notified about.
- Ensures notifications are **non-intrusive** and **personalized**.

NOTIFICATIONS CHANNEL

- Abstract layer for delivery methods (e.g., push notifications, email, in-app alerts).
- Allows flexibility in how messages reach users

EVENTDISPATCHER

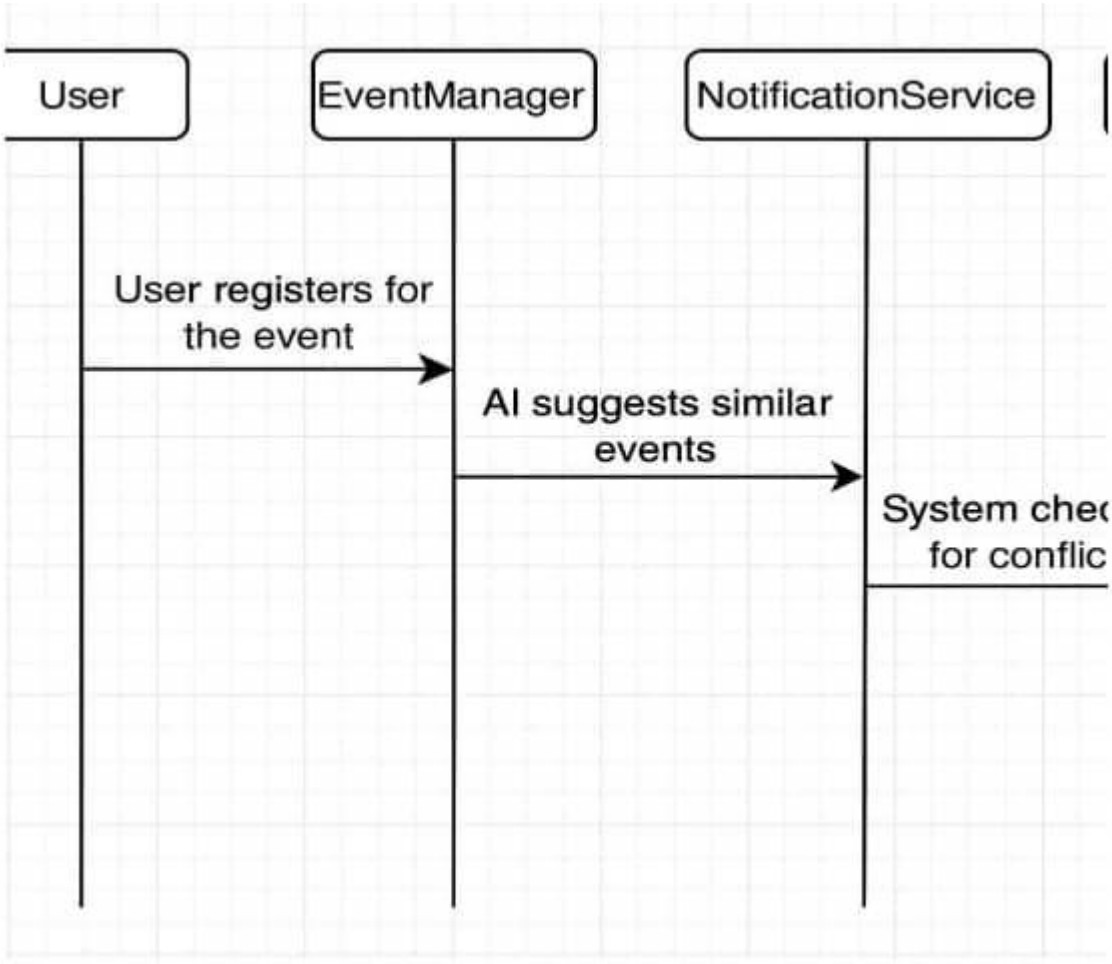
- Triggers the notification process by detecting relevant events.
- Passes these events to the Notification Service for processing.

How It All Connects

1. **EventDispatcher** detects an event.
2. **Notification Service** evaluates the event and checks **User Preferences**.
3. If relevant, it uses the **AI Suggester** to refine targeting.
4. It creates a **Notification** and sends it via the appropriate **Notification Channel**.
5. The **User** receives the message in their preferred format.

This setup ensures that users get **timely, relevant, and personalized notifications** without being overwhelmed. It's a smart, modular design that balances automation with user control.

Sequence Diagram



Flow Summary:

1. User → EventManager: Registers for an event.

2. EventManager → AISuggester: Requests similar event suggestions.

3. AISuggester → User: Displays recommended events.

4. EventManager → NotificationFactory → NotificationService: Sends

confirmation or "event full" notification instantly.

5. EventManager → Logger: Logs the activity and checks for

registration conflicts

Notification System Role

- **AI Suggestions**: After registration, the NotificationService analyzes the user's preferences or past behavior to recommend similar events.
- **Conflict Detection**: It also checks the user's calendar or event history to prevent double-booking or overlapping events.

Reflection on the Notification System Design

This notification system embodies a **modular, user-centric architecture** that balances automation with personalization. The design reflects a thoughtful integration of AI-driven relevance, user control, and scalable delivery mechanisms. Each component is clearly delineated, promoting maintainability and extensibility.

Key reflections:

- **User Empowerment**: By separating User Preferences from User, the system respects autonomy, allowing granular control over notification timing, frequency, and channels.
- **AI Integration**: The AI Suggester introduces intelligent filtering, ensuring users receive meaningful alerts without being overwhelmed—an ethical stance against notification fatigue.
- **Loose Coupling**: The use of abstracted Notification Channel and EventDispatcher components supports flexibility in delivery and event sourcing, making the system adaptable to new technologies or platforms.
- **Responsibility Segregation**: Each class has a focused responsibility, aligning with SOLID principles and making the system easier to test, debug, and evolve.