Group Design Activity: Modeling a Real-Time Notification System for a Community Event App

GROUP ASSIGNMENT

GROUP MEMEBERS:

Bristol Shalowa 2421003

Sibongile Tembo 2410291

Gabriel Mwila 2410234

GROUP DISCUSSIONS AND CONTRIBUTIONS:

Our group collaborated effectively throughout the 2-hour activity, dividing tasks based on individual strengths and maintaining clear communication using online and in-person discussion.

Person A (Research Lead):
Investigated suitable design patterns and concurrency control methods. Found that the **Observer Pattern** best fit the real-time update requirement, while **Singleton** and **Factory** could support central control and dynamic notification creation. Also researched **locking and queuing** to prevent simultaneous registration conflicts.

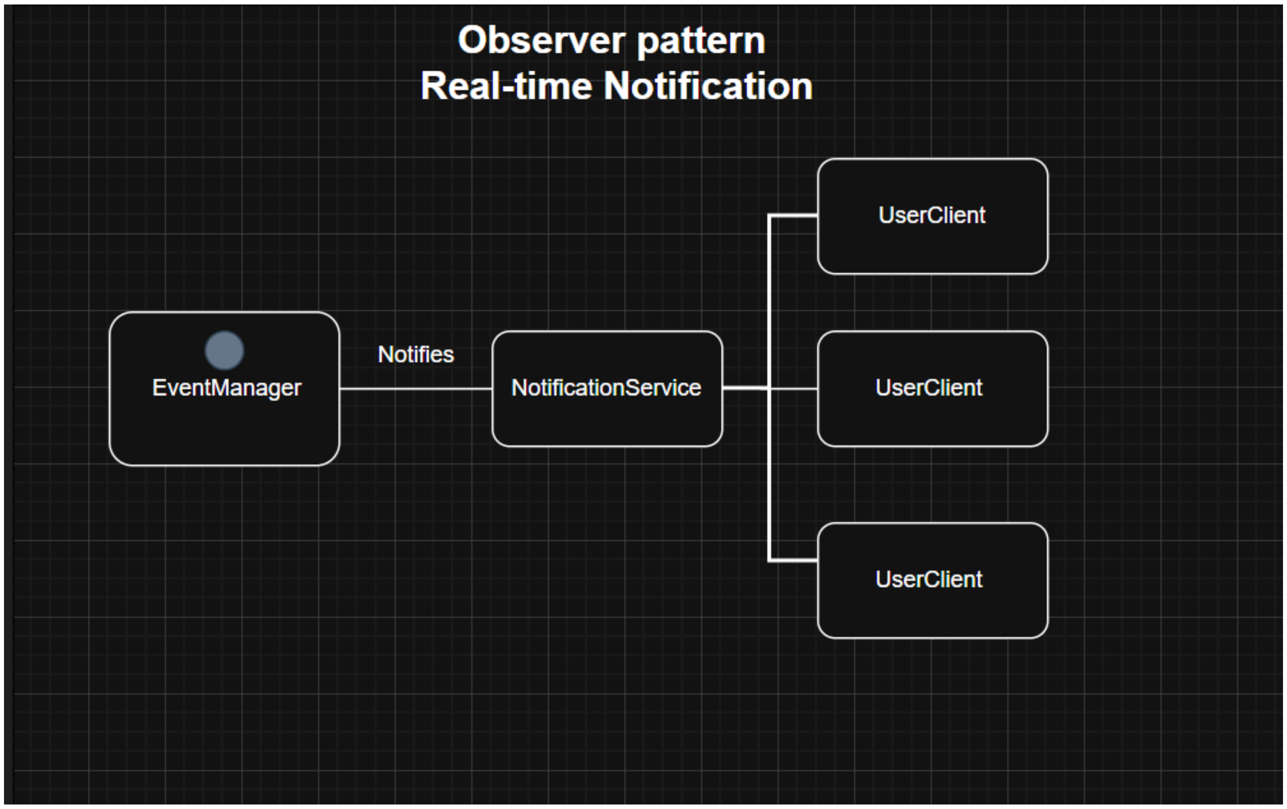
Person B (Modeling Lead):
We used **Draw.io** to model the **class diagram** (static structure). Integrated research findings into the model, showing how users, events, and notifications interact through the Observer pattern. Exported the diagrams as PNG/PDF files for presentation.

Person C (Writer & Presenter):
Compiled all group input into a concise **1-page reflection** and **2-minute presentation summary**. Wrote the explanation for why the Observer pattern was chosen, its tradeoffs, and how concurrency was addressed. Ensured the final document was clear and aligned with the lecturer's expectations.

Use of AI and Human Input

We used **AI tools (ChatGPT)** to brainstorm ideas, structure the report, and clarify design pattern functions. AI suggested using the **Observer pattern** for real-time updates and inspired the concurrency strategy using queues and locks. However, **human input remained central** — we discussed, adapted, and finalized all ideas manually to fit our understanding and the system requirements.

THE OBSERVER PATTERN FOR REAL TIME NOTIFICATION

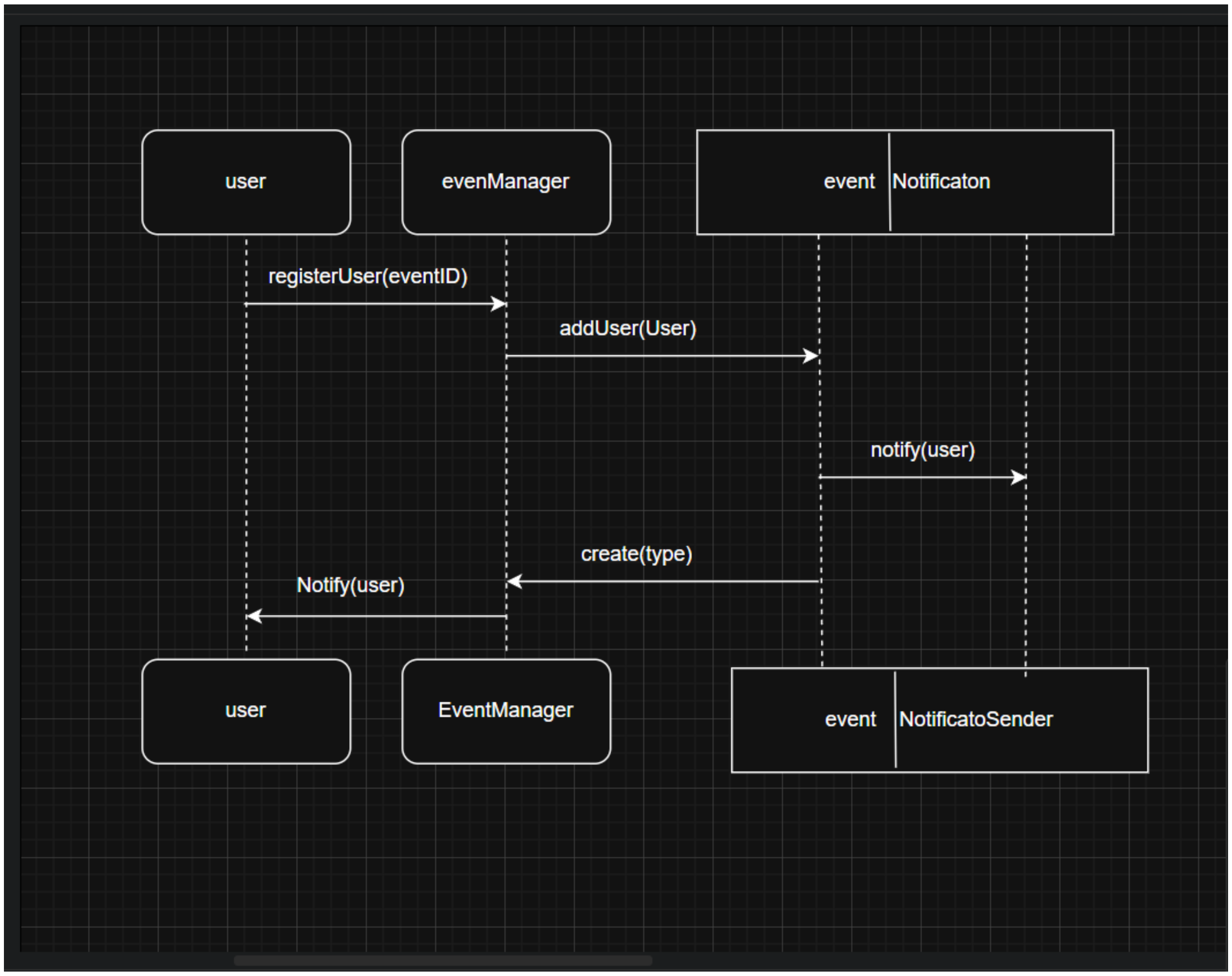


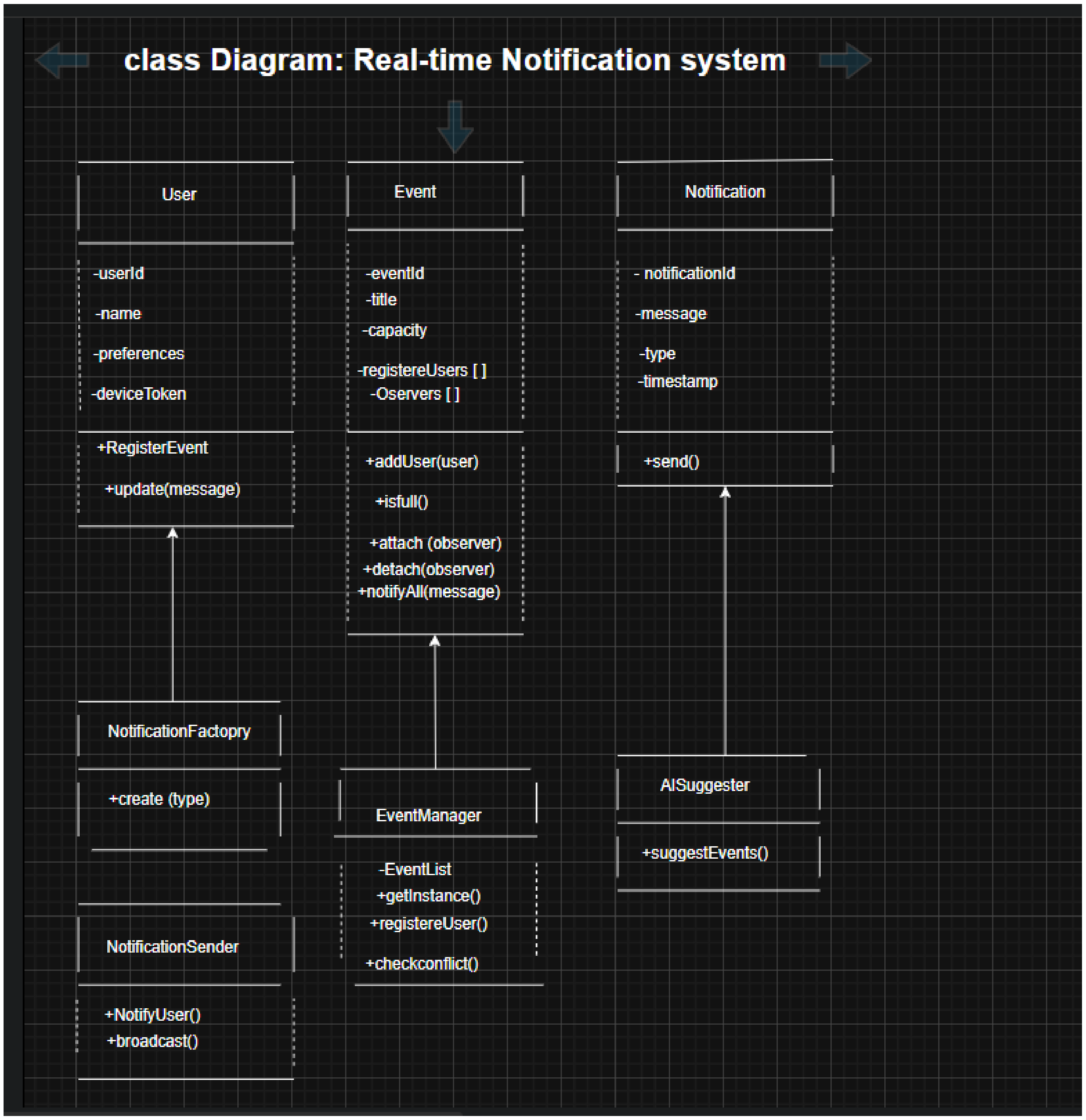
1. EventManager (Subject): Detects changes in events.

2. NotificationService: Sends updates to all subscribed users.

3. UserClients (Observers): Instantly receive notifications like “Event is full!

A SEQUENCE DIAGRAM SHOWING ILLUSTRATING HOW THE PROCESSES ARE SUPPOSED TO OCCUR:





Group Design Activity: Modeling a Real-Time Notification Community Event App

1. System Overview

This project models a simple Real-Time Notification Community Event App for a local community center. The system allows multiple users to register for events, receive real-time updates (like 'Event is full'), and includes a basic AI event suggested. The Observer Pattern was chosen to handle live notifications and ensure users receive instant updates when events change.

2. Explanation and Reflection

We chose the Observer Pattern because it naturally supports real-time notifications — when an event's state changes (like a new registration or a cancellation), all subscribed users are instantly updated without constantly refreshing. This makes it perfect for handling live updates in a community event system.

For concurrency, we applied concepts like locking and queues to avoid double-booking. For example, if two users try to register for the last spot simultaneously, the system will process one request first and place the second in a waiting state. Using queues and synchronized event managers ensures data consistency even with many users online at once.

We also incorporated a simple AI event suggested component, which observes user preferences and recommends similar events, enhancing personalization. Real-time updates are modeled through an event-driven system, where the Event Manager acts as the subject (publisher) and Users, Notifications, and the AI Suggested act as observers (subscribers).

This combination of Observer Pattern, concurrency handling, and event-driven design allows our system to remain responsive, scalable, and reliable, ensuring every user sees the latest event information instantly while maintaining fairness and consistency in registrations.