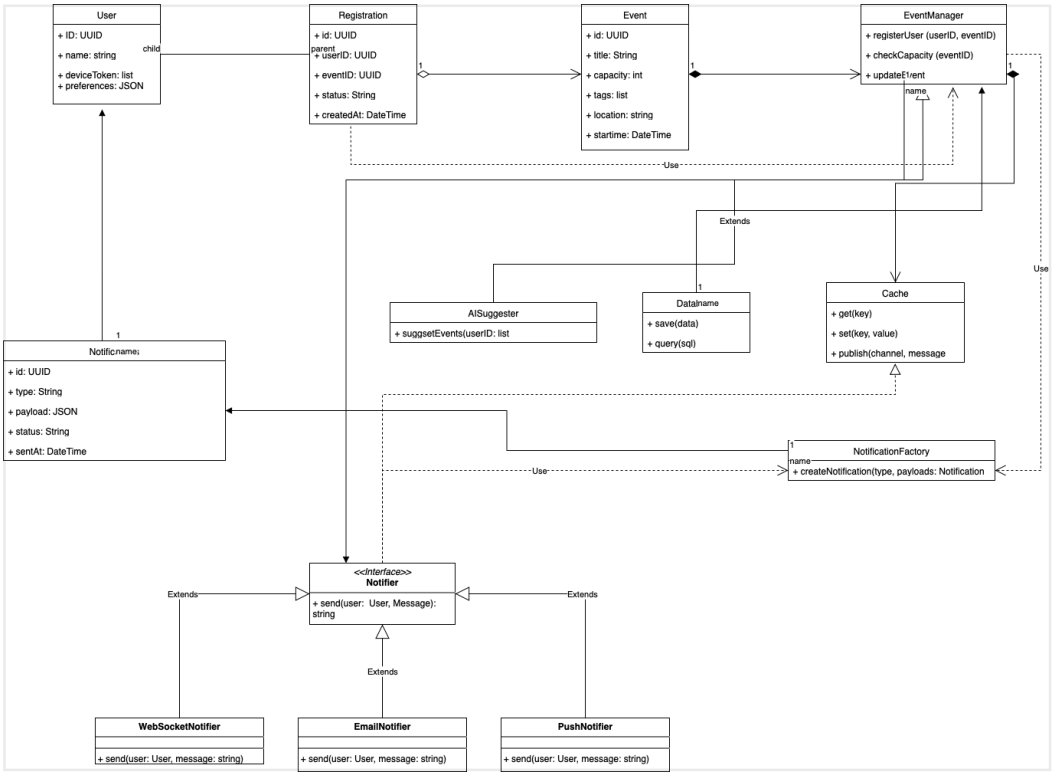
Presentation



Realtime Event Notification System — Short Presentation Script

Introduction

Our design is a real-time event registration and notification system. It allows users to register for events, receive instant notifications, and get intelligent event recommendations.

Core Entities

Main classes are **User**, **Event**, and **Registration**.
A User can register for an Event through the Registration class, which holds the user, event, status, and timestamp.

EventManager

Acts as the brain of the system.
It handles registration logic, checks capacity, prevents double booking, logs actions, and interacts with the **NotificationFactory** and **Database**.

Notification System

Uses **Factory** and **Strategy** patterns.

- **NotificationFactory** creates notifications.
- The **Notifier** interface has subclasses:
  - **WebSocketNotifier**
  - **EmailNotifier**
  - **PushNotifier**

These allow flexible delivery methods.

AISuggester

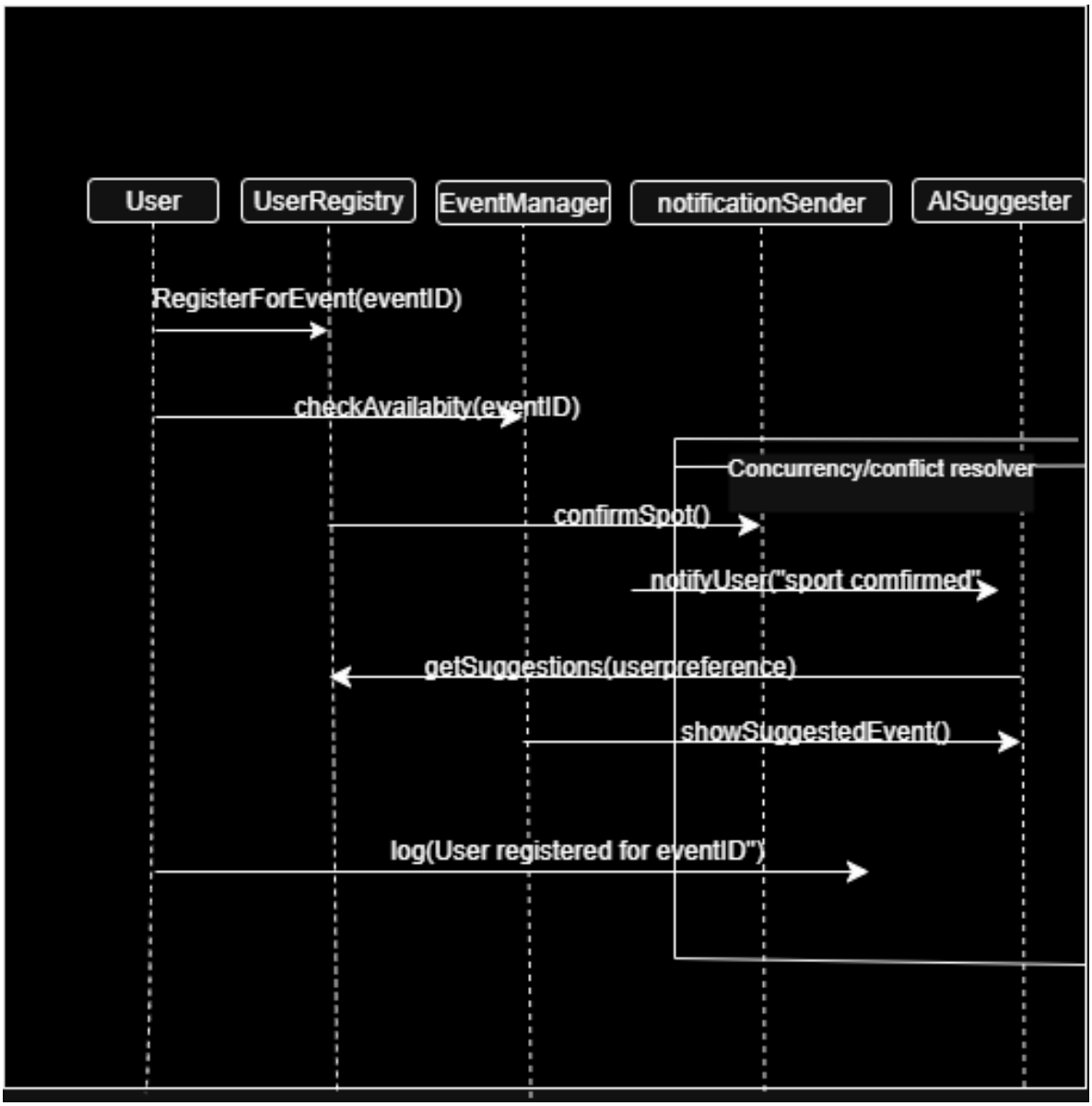
Analyzes user preferences and registration history to recommend similar events.
Works with the **Database** for stored data and **Cache** for faster access.

Concurrency & Logging

EventManager ensures safe, atomic updates to event capacity using the database and cache.
All actions are recorded in an **AuditLog** for transparency and debugging.

Conclusion

Our design combines **design patterns**, **AI recommendations**, and **real-time updates**.
It’s modular, scalable, and reliable ideal for modern event platforms.



Sequence of Operations

1. User → UserRegistry:
• The user starts the process by requesting to register for a specific event.
2. UserRegistry → EventManager:
• The system checks if there's space available for the event.
3. EventManager:
• If available, the EventManager reserves a spot for the user.
4. EventManager → notificationSender:
• A confirmation message is sent to the user via the notificationSender.
5. EventManager → AISuggester:
• Based on the user's preferences, the system requests additional event suggestions.
6. AISuggester → EventManager:
• The AI returns a list of suggested events.
7. UserRegistry:
• The system logs the registration for tracking and auditing purposes.

Concurrency/Conflict Resolver

- This mechanism ensures that if multiple users try to register for the same event simultaneously, conflicts (like overbooking) are resolved gracefully possibly by locking, queuing, or prioritizing requests.

Summary

This diagram models a well-orchestrated flow where:

- The UserRegistry acts as the gatekeeper.
- The EventManager handles core event logic.
- The notificationSender keeps users informed.
- The AISuggester adds personalization.
- And the Concurrency Resolver ensures fairness and consistency.