Event Management System

SE-327

Submitted by:

Manahil Aamir (22011598-018)
Aeman Nisar (22011598-044)
Zoraiz Saad (22011598-003)

Submitted To:

Mam Hafsa Dar

Contents

Event Management System

Problem Definition:

People often find it difficult to manage events, which causes scheduling conflicts, missed details, and disorganization. This project solves this problem by providing a system to help users create and manage events efficiently.

Importance of Solving the Problem:

Solving this problem saves time, reduces errors, and helps users avoid scheduling conflicts. It will make organizing events easier, ensuring no one misses important meetings or deadlines.

Impact on Intended Users:

- This system will help users create, edit, and organize events quickly and accurately.
- It will also detect conflicts and provide reminders, helping users manage their time better.

Requirements Prerequisites:

Functional Requirements:

1. Users can create, edit, and delete events.
2. The system must handle event scheduling (start and end times).
3. The system must notify users if there are any conflicts in event schedules.
4. A calendar view must display all events.
5. Events must be categorized (e.g., work, personal).

Non-functional Requirements:

1. **Performance**: The system must run smoothly without delay, even with multiple users.
2. **Security**: User data must be protected with secure logins and data storage.
3. **Usability**: The system must be simple to use and easy to navigate
4. **Scalability**: It should work well as the number of users and events grows.

Architectural Prerequisites:

Components and Subsystem:

User Interface (UI):

- **Event Creation/Editing Subsystem:** Allows users to create or edit events using forms.
- **Calendar View Subsystem:** Displays events in a calendar format for easy scheduling.

- **Notification Subsystem:** Provides reminders or alerts for upcoming or conflicting events.

Business Logic Layer:

- **Event Management Subsystem:** Handles the creation, updating, and deletion of events.
- **Conflict Detection Subsystem:** Checks for overlapping or conflicting event schedules.
- **Event Categorization Subsystem:** Organizes events into categories like work or personal.
- **Reminder Management Subsystem:** Sends out notifications for upcoming events.

Data Access Layer:

- **Event Data Subsystem:** Manages saving, updating, and retrieving event details from the database.
- **User Data Subsystem:** Handles user authentication, profiles, and access permissions.
- **Notification Data Subsystem:** Stores and retrieves user notification preferences and schedules reminders.

Class Diagram:

Admin-User:

Responsible for managing the overall system and overseeing Event-Managers.

Event-Manager:

Manages multiple events, including creating, updating, deleting, and checking for conflicts among them.

Event:

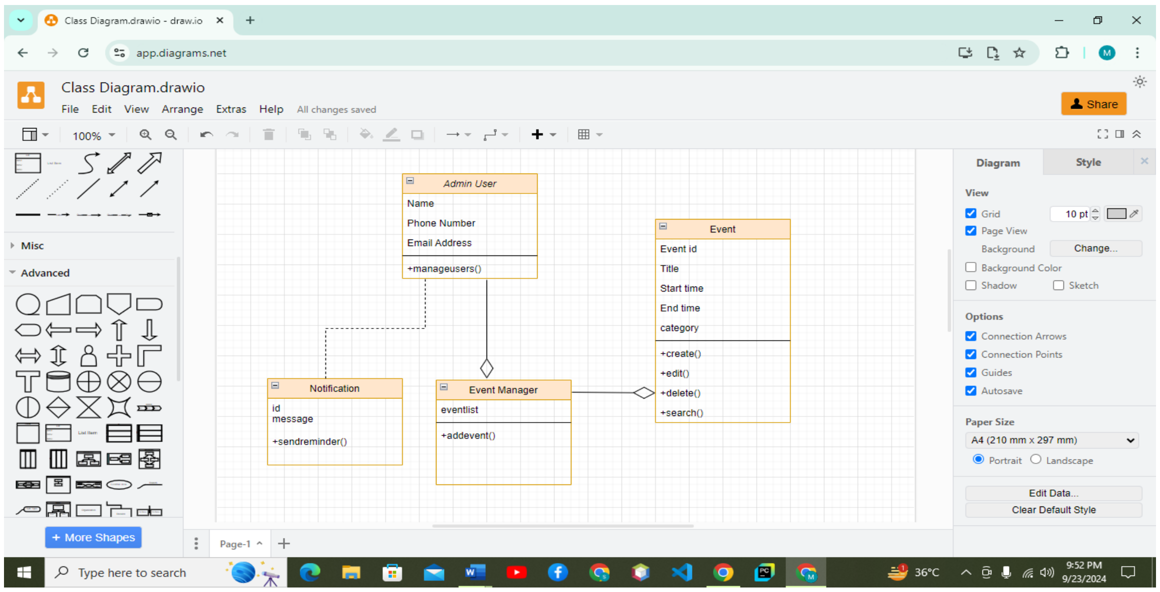
Represents the details of an event, including title, date, time, category, and other relevant information.

User:

Represents individuals who can create, view, and manage their own events and receive notifications.

Notification:

Sends alerts and reminders to users regarding their events, helping them stay organized.



Architecture Prerequisites points:

1. Program Organization:

- The event management module should be organized into separate components, such as:
  - User Interface (UI): Handles user interactions.
  - Business Logic Layer: Contains the core functionality, like managing events.
  - Data Access Layer: Manages interactions with the database.
- This makes system simple and easier to use.

2. Major Classes:

- **User:** Manages user information and authentication.
- **Event:** Represents event details (title, date, time, category).
- **Event Manager:** Responsible for creating, updating, deleting events and checking for conflicts.
- **Notification:** Sends reminders or alerts to users regarding their events.

3. User Interface Design:

- The UI should be intuitive and user-friendly, providing:
  - A calendar view for easy event management.
  - Forms for creating and editing events.
  - Forms for view events by category or date.
  - Forms for deleting events.

4. Resource Management:

- Our system uses efficient resources like memory and processing power, database especially when handling multiple users or events.
- Implement caching strategies for frequently accessed data (e.g., user profiles or event lists).

5. Security:

- This system ensures data protection through:
    - User authentication (e.g., using passwords and username).
    - Data encryption for sensitive information (like user details) are used in the system.
    - Regular security must be handling audits to identify vulnerabilities.

6. Performance:

- This system must respond quickly to user actions (like loading events or saving changes).
- For complex tasks (like checking many events for conflicts), this system must use background processing to avoid slowing down.

7. Scalability:

- The Event Management System is designed to handle growth by:
    - Use a scalable database (like cloud-based solutions) to accommodate more users and events.
    - Distribute the load between multiple servers to avoid performance drops when many users are active.

8. Input/Output:

- The user will provide details like the event name, date, time, location, and any related information.
- The user can modify or update existing event details, such as changing the time or location.
- This system will confirm that the event has been successfully scheduled and display all the details.
- If the event's time clashes with another event, this system will detect the conflict and display a warning.

9. Error Processing:

- This system implements error handling to manage issues gracefully:
    - Use try-catch blocks to capture exceptions.
    - Provide meaningful error messages to users and log errors for developers to review.

10. Fault Tolerance:

- This system is designed to continue operating in case of failure:
    - Regular data backups must be made so that no event data is lost.
    - If the page is not loaded **404 error page** must show to the user.
    - If one part of the system fails, there must be backups or alternative ways to keep it running.

11. Buy vs. Build Decisions:

- Evaluate whether to use third-party libraries or services (e.g., calendar APIs) versus building features in-house.
- In a **buy** scenario, the API already handles event conflict detection, saving time but with limited customization. In a **build** scenario, you'll need to create custom logic for detecting conflicts, giving more control but requiring more development effort.

12. Reuse Decisions:

- The components that can be reused across the project:
    - A notification system is reused from the "Pharmacy Management System" for sending reminders could also be adapted for alerting users to new features or updates.