EVENTHUB

Software Requirements Specification (SRS)

CS383
Group Project

ARYAM Alzamil -
JANA Alrayes -
REEMA Alghufily -
NOURA Alomair -
WASAYIF Alsalammah -

TABLE OF CONTENT

1. INTRODUCTION

EventHub is a comprehensive event management platform designed to digitalize and streamline the entire event lifecycle. The system addresses key challenges in event organization by providing integrated solutions for planning, registration, ticketing, and feedback collection.

This document specifies the complete requirements for developing EventHub, ensuring it meets the needs of all stakeholders including event organizers, attendees, speakers, and administrators while maintaining high standards of security and usability.

1.1 Purpose

This Software Requirements Specification (SRS) document describes all requirements for the *EventHub system* - an integrated event management platform. It serves as a complete description of the system's behavior and acts as a formal agreement between developers, clients, and stakeholders.

1.2 Product Scope

EventHub is a web-based platform that simplifies event management for organizers while providing smooth registration and ticketing for attendees. The system supports the entire event lifecycle from creation to feedback collection, serving multiple user roles including administrators, organizers, speakers, and attendees.

1.4 Structure

This document contains:

- Introduction & System Overview
- Requirements Engineering Process
- Functional & Non-Functional Requirements
- External Interfaces & Use Cases

1.3 References

- IEEE Std. 830-1998 - Software Requirements Specifications
- WCAG 2.1 Accessibility Guidelines
- OWASP Security Standards
- EventHub Project Charter v1.0

2. SYSTEM OVERVIEW

2.1 Product Perspective

EventHub is a new, self-contained web-based software system. It is designed as a comprehensive, integrated solution for event and conference management, intended to replace traditional manual processes (such as email registration or spreadsheets).
The system will interact with external services such as Payment Gateways and Email Services to perform its core functions.

2.2 Product Features

The main features EventHub will provide to its users are:

- User Management: Allowing users of all types to securely register, log in, and manage their profiles.
- Event Management: Enabling Organizers to create, publish, edit, and cancel events with details including date, venue, and description.
- Registration & Booking: Allowing Attendees to search for and register for events, secure payment processing, and receive digital tickets.
- Session & Speaker Management: Enabling Organizers to schedule sessions and to review, accept, or reject abstracts submitted by Speakers.
- Feedback System: Allowing Attendees to submit feedback and rate sessions or events after completion.
- Reporting Management: Enabling Admins to generate reports on system activity.

2.3 User Roles and Characteristics

- Admin: A high-privilege user responsible for system oversight, approving new Organizer accounts, and monitoring reports.
- Organizer: The user who creates and manages an event (sets pricing, schedules sessions, reviews abstracts, manages tickets).
- Attendee: The primary user who searches for events, registers, pays fees, attends, and provides ratings.
- Speaker: A user who wishes to present at an event, submits an abstract, and tracks their submission status.

2. SYSTEM OVERVIEW

2.4 Operating Environment

- EventHub will operate as a centralized Web Application.
- The system will be hosted on a cloud server to ensure high availability and scalability (as implied by the microservice architecture).
- Users (all types) will require a device (PC, smartphone, etc.), a modern web browser (e.g., Chrome, Safari), and an internet connection to access the system.
- The system will connect externally to Payment Gateway APIs and Email Service APIs via the internet.

2.5 Design and Implementation Constraints

- The system must be a web application (no native mobile app is required at this stage).
- All communication must use HTTPS (SSL) to ensure security .
- Critical Constraint: The system must not store any user credit card information directly in its database. All payments must be processed exclusively through an external, compliant Payment Gateway.
- A relational database (e.g., MySQL) must be used for data persistence.

2.6 Assumptions and Dependencies

- Assumption: It is assumed that all users have a stable internet connection and a compatible web browser.
- Dependency: The system's core functionality is critically dependent on the availability of third-party APIs (Payment Gateway, Email Service). If these services fail, system functions will be impaired.
- Dependency: The system assumes an active Admin is available to approve new Organizers before they can create events.

3. REQUIREMENTS ENGINEERING

The requirements engineering process for the EventHub – Event & Conference Management System was conducted using systematic and collaborative methods to gather, analyze, and validate all system requirements. The goal was to ensure that the final system meets the needs of all stakeholders, including Admins, Event Organizers, Attendees, and Speakers.

———

3.1 Requirements Elicitation

Requirements elicitation involved collecting information about what the users expect from the EventHub system. Our group used several techniques:

1. Brainstorming Sessions

The team met to discuss potential features and identify the key problems the system should solve.
Through these sessions, the idea of EventHub was shaped around event creation, registration, payment, abstract submission, and session management.

2. Comparative Analysis

We reviewed existing event management platforms to identify common features, weaknesses, and improvements for our system.
This helped refine realistic, useful requirements.

3. Identifying User Types

Based on the system’s nature, we determined four main user groups:

- Admin
- Event Organizer
- Attendee
- Speaker

Understanding each user’s needs guided the functional and non-functional requirements.

4. Group Interviews / Discussions

Each team member represented a stakeholder type (organizer, attendee, speaker, admin) and contributed expectations based on real use cases.
This helped us elicit clear requirements from multiple perspectives.

3. REQUIREMENTS ENGINEERING

3.2 Requirements Analysis

Requirements analysis was performed to:
- Remove inconsistencies
- Group related requirements
- Ensure feasibility
- Identify system boundaries

1. Classification of Requirements

We categorized all requirements into:
- Functional Requirements

(e.g., event creation, registration, login, payment, abstract submission)
- Non-Functional Requirements

(performance, security, scalability, usability)
- External Interface Requirements

2. Refinement and Prioritization

Each requirement was reviewed to ensure:
- Clarity
- No ambiguity
- No redundancy
- Consistency across the entire SRS

Requirements were then prioritized as:
- Critical (must-have) → login, event creation, payment
- Important (should-have) → speaker abstract submission
- Optional (nice-to-have) → calendar sync

3. Modeling and Representation

We used several diagrams to support analysis:
- Use Case Diagram – identified system interactions
- Activity Diagrams – visualized workflows
- Sequence Diagrams – detailed interaction sequences
- Class Diagram – defined objects and relationships

These diagrams helped ensure the requirements matched the expected system behavior.

3. REQUIREMENTS ENGINEERING

3.3 Requirements Validation

Requirements validation ensures that the documented requirements are complete, correct, and aligned with stakeholder expectations.

Validation Techniques Used

1. Team Review Sessions

The group reviewed the SRS document together to confirm:

- Requirements are testable
- Requirements reflect real user needs
- Requirements match system goals
- No requirement is missing or contradictory

These reviews happened during Meetings 3, 4, and 6.

2. Cross-Checking with Use Cases

We compared the functional requirements with the use case diagram to ensure every use case had a supporting requirement.

3. Consistency Checking

The UML diagrams were checked against the SRS to ensure the design aligns with the requirements:

- Sequence diagrams matched functional requirements
- State diagrams matched system behavior rules
- Class diagrams aligned with entities described in requirements

4. Feasibility and Scope Review

Each requirement was validated for:

- Technical feasibility
- Time feasibility
- Alignment with project scope
- Compatibility with external APIs

Requirements outside scope were removed or revised.

4.FUNCTIONAL REQUIREMENTS

1. User Management:
The system must allow users to register, log in, and manage their profiles securely.

2. Event Management:
Organizers must be able to create, update, publish, and cancel events with details including date, time, venue, and description.

3. Registration & Booking:
Participants must be able to register for events, select sessions, and receive email/SMS confirmation notifications.

4. Ticketing & Payment:
The system must support digital ticket generation, secure payment processing, and refund management.

5. Sessions & Speakers Management:
Organizers must manage sessions, schedules, and comprehensive speaker profiles efficiently.

6. Feedback System:
Participants should be able to submit feedback and rate events or sessions using a 5-star system.

7. Organizers & Staff Management:
The system should allow assigning roles, tasks, and permissions to event staff with different access levels.

8. Accounts & Invoicing:
The platform must automatically generate invoices, track payment status, and provide downloadable receipts.

5.NON-FUNCTIONAL REQUIREMENTS

Non-Functional Requirements

1. Performance Requirements

- **The system must load the main events page within 3 seconds under normal network conditions.**
- **The system should handle up to 500 concurrent users registering for an event without performance degradation.**
- **Searches for events must return results within 2 seconds.**

---

2. Safety Requirements

- **The system must ensure that all user data is backed up automatically at least once per day to prevent data loss.**
- **In the event of a failure (e.g., power outage, server crash), the system should recover to the last known stable state without losing user registrations or payment information.**
- **The system must prevent accidental deletion of events by requiring confirmation prompts for critical operations.**

---

3. Security Requirements

- **All communication between the user and the system must be encrypted using SSL/TLS.**
- **Users must authenticate using a secure login system (hashed & salted passwords).**
- **Sensitive data such as payment information must never be stored directly on the server.**
- **Only authorized users (Admin, Organizer) should access restricted system functions.**

---

4. Software Quality Attributes

4.1 Usability

- **The interface must be easy to use, allowing attendees to complete the registration and payment process in no more than 4 steps.**
- **The system should support both desktop and mobile screens.**

4.2 Reliability

- **The system should maintain 99% uptime during the event registration period.**
- **System errors must be logged automatically for debugging.**

4.3 Maintainability

- **The system must follow a modular architecture to allow easy updates and improvements.**
- **Code should follow standard naming conventions and be documented for future developers.**

4.4 Scalability

- **The system should be capable of expanding to support multiple organizations, more events, and higher user load without architectural changes.**

6.EXTERNAL INTERFACE REQUIREMENTS

1. Payment Gateway Interface

The system must connect securely with external payment providers (e.g., PayPal, Stripe, or Mada).
All payment requests and responses must use HTTPS with TLS 1.2 or higher.
The interface must send transaction data (amount, user ID, event ID) in JSON format and receive confirmation or failure responses from the payment API.

2. Email & SMS Notification Interface

The system must integrate with an external email/SMS service (e.g., SendGrid, Twilio) to send registration confirmations, ticket details, and event reminders.
The communication must use API keys for authentication and support both HTML and plain-text message formats.

3. Database Interface

The application must connect to an external relational database (e.g., MySQL or PostgreSQL) through a secure connection string.
The interface must support CRUD (Create, Read, Update, Delete) operations for event, user, and ticket data.
All queries must be parameterized to prevent SQL injection.

7.SOFTWARE ENGINEERING ETHICS AND RESPONSIBILITY

Principle 1: PUBLIC:

The project will follow the PUBLIC interest principle by prioritizing its responsibility to all users.This is achieved through two main applications: Privacy, by protecting all user personal data (like emails and phone numbers) and never sharing it without explicit consent; and Security, by protecting users from financial harm through secured payment channels .

Principle 2: CLIENT AND EMPLOYER:

It is applied in two ways: Honesty & Competence, meaning the team must be transparent about the system's true capabilities (like user limits or speed); and Confidentiality, which requires protecting the organizer's private business data (like sales revenue and attendee lists) from competitors.

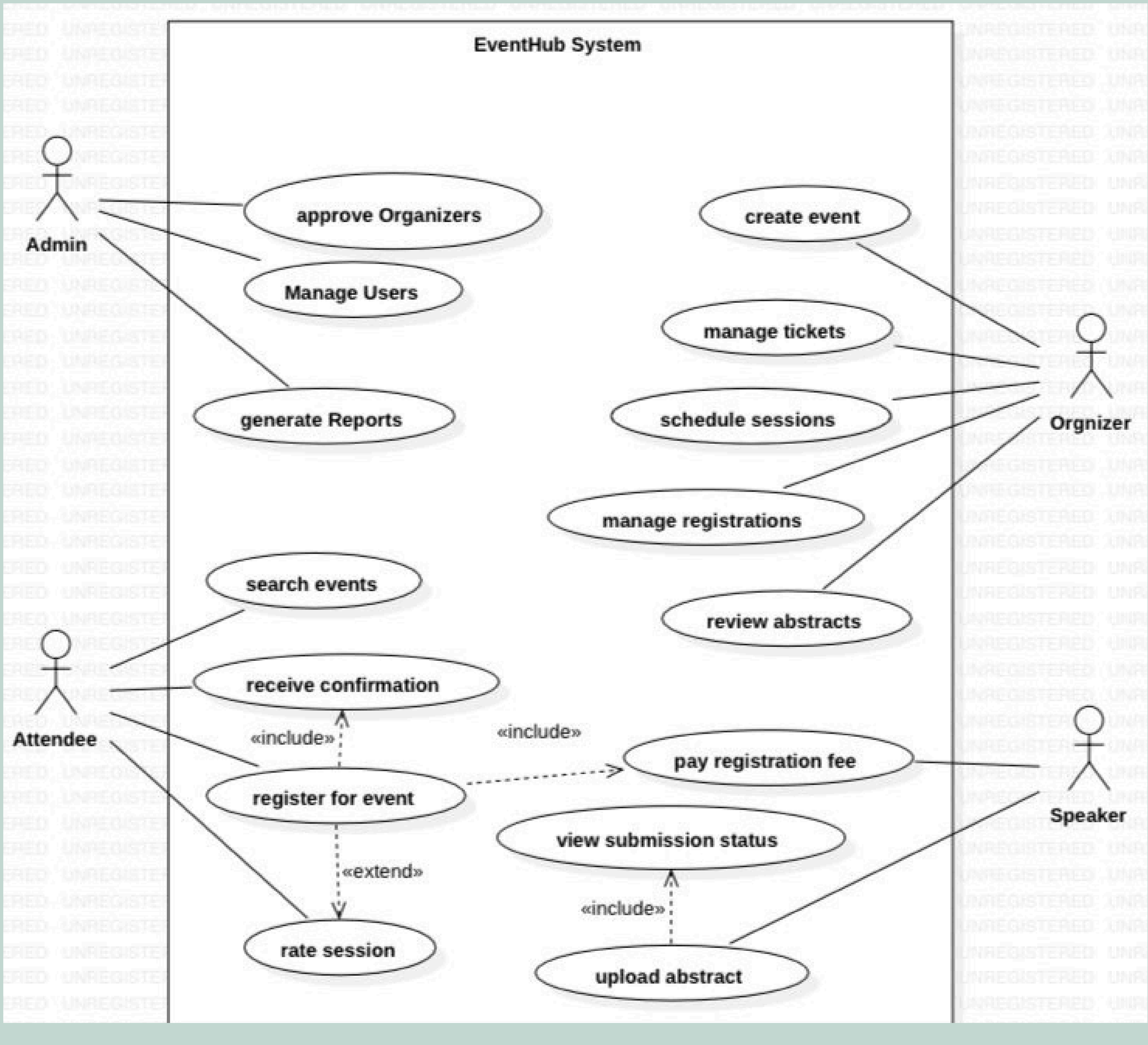
Principle 3: PRODUCT:

Ensures the PRODUCT's quality. It is applied through Reliability, meaning the system must be dependable and not fail during critical operations (like crashing during sales, losing data, or overbooking), and Robustness, meaning it must handle failures gracefully (e.g., ensuring payment transactions either fully complete or fully fail, leaving no uncertainty for the user).

Principle 4: JUDGEMENT:

This principle applies to the JUDGEMENT of the system, ensuring its processes are fair and honest. This is demonstrated through Transparency, by clearly displaying the total cost (including all fees) to users before payment .

8.USE CASE


EventHub System
Admin
approve Organizers
Manage Users
generate Reports
search events
receive confirmation
«include»
register for event
«extend»
rate session
Attendee
create event
manage tickets
schedule sessions
manage registrations
review abstracts
Orgnizer
«include»
pay registration fee
view submission status
«include»
upload abstract
Speaker

EVENTHUB

Software Design Document (SDD)

CS383
Group Project

ARYAM Alzamil -
JANA Alrayes -
REEMA Alghufily -
NOURA Alomair -
WASAYIF Alsalammah -

TABLE OF CONTENT

1. INTRODUCTION

This Software Design Document (SDD) provides the technical blueprint for the EventHub event management platform. It translates the requirements from the SRS into detailed architectural and design specifications, serving as a comprehensive guide for the development team during implementation.

The document outlines the system's architecture, component designs, data models, and interface specifications to ensure consistent and efficient development of all EventHub modules including event management, user registration, ticketing, and payment processing.

1.1 Purpose

This Software Design Document (SDD) provides the comprehensive architectural and design details for the EventHub event management system. It serves as a technical blueprint for developers, outlining the system structure, component interactions, data models, and interface specifications to guide the implementation phase.

1.2 Product Scope

This document covers the complete system design of EventHub, including:
- Architectural overview using layered architecture pattern
- Detailed component design with UML diagrams (class, sequence, activity, state diagrams)
- Database schema and data flow design
- User interface specifications and wireframes
- Integration with external services (payment gateways, email/SMS providers)

1.3 References

- IEEE Std. 1016-2009 - Software Design Documentation
- EventHub Software Requirements Specification (SRS) v1.0
- UML 2.5 Specification - Object Management Group
- WCAG 2.1 Accessibility Guidelines

1. INTRODUCTION

1.4 Structure

This document is organized as follows:
- Section 1: Introduction and overview
- Section 2: System architecture and decomposition
- Section 3: Data design and database schema
- Section 4: Component design with UML diagrams
- Section 5: User interface design specifications

1.5 Process Model

The development of the ***EventHub*** system follows an Agile software process model. Agile was selected because the project requirements involve multiple user roles (Admin, Organizer, Attendee, Speaker), external integrations (Payment Gateway, Email Service, Calendar API), and several features that may need refinement over time, such as search and filtering, ticket pricing, and feedback collection. An Agile approach supports incremental delivery of these features, continuous feedback from stakeholders, and flexible handling of requirement changes during the semester.

The team works in short, iterative cycles where each iteration focuses on implementing a small, coherent set of functionalities, such as event creation, ticket management, or session scheduling. Each iteration includes the main software engineering activities: planning, analysis, design with UML diagrams, implementation, and testing. At the end of each iteration, the implemented features are reviewed and adjusted if needed. This process model is suitable for an academic group project like ***EventHub***, as it encourages collaboration, regular progress tracking, and gradual improvement of both functional and non-functional requirements.

2. SYSTEM OVERVIEW

The **EventHub** system is designed as a distributed system based on a Microservice Architecture. This design decision was made to ensure high scalability, maintainability, and a clear separation of concerns.

The system is decomposed into several independent services, each responsible for a specific business domain, communicating with each other via lightweight APIs (e.g., REST).

The primary services in the system are:

• **User Service:** Responsible for all user authentication, profile management, and permissions.

• **Event Service:** Responsible for the core business logic, including event creation, session management, registration processing, and capacity validation.

• **Payment Service:** Responsible for processing payments and securely integrating with the external payment gateway.

• **Notification Service:** Responsible for sending all notifications (such as registration confirmations) via email or SMS.

The frontend will be a Single Web Application that all users (Attendee, Organizer, Admin) interact with. This application will, in turn, communicate with the various backend microservices to execute requests.

3. ARCHITECTURE DESIGN

3.1 Architecture Description

EventHub utilizes a Microservice Architecture pattern. the system is separated into a collection of independent, deployable services.

Communication Flow:

1. Entry Point: The user (Admin, Organizer, Attendee) interacts with the Web App Interface (a client-side application).
2. API Gateway: All requests from the client are channeled through a single API Gateway. This gateway is responsible for routing each request to the correct microservice and handling cross-cutting concerns like authentication.
3. Services: The request is processed by the designated service (e.g., User Service, Event Service).
4. Databases: Each microservice owns its private database (e.g., User DB, Event DB) and cannot access the databases of other services directly. This ensures loose coupling and the "database-per-service" pattern.
5. Service-to-Service Communication: When one service needs data from another (e.g., Event Service needing user data), it communicates via a Service Registry, which manages the locations and health of all services.
6. External Services: Specific services (like the Payment Service and Notification Service) interface with the required External APIs.

3.2 Decomposition Description

The system has been decomposed into the following key microservices based on business capability:

- User Service: Responsible for all user-related concerns: account creation, authentication, authorization, and profile management.
- Event Service: The core domain. Manages event creation, session scheduling, capacity validation, and the business logic for registrations.
- Payment Service: An isolated, secure service responsible only for processing payments and interacting with the external payment gateway.
- Notification Service: Manages all user-facing communications (registration confirmations, event reminders) via email/SMS.
- Reporting Service: Gathers and aggregates data for admins, running independently so as not to impact the performance of the core services.

3. ARCHITECTURE DESIGN

3.3 Design Rationale

The Microservice Architecture was chosen over a traditional Monolithic architecture for the following key reasons:

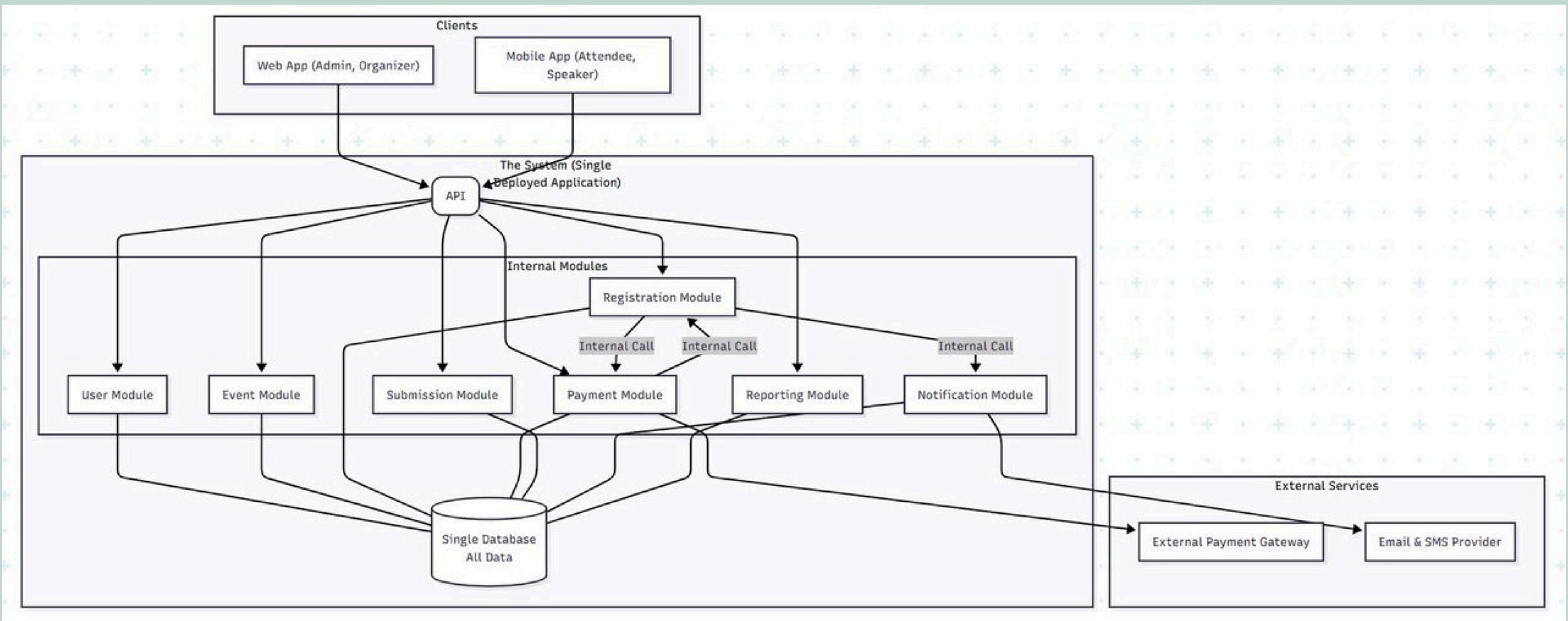
1. Scalability: Services that experience high load (like the Event Service during registration peaks) can be scaled independently of the rest of the system, optimizing resource usage.

2. Maintainability: Each service is small and focused, making it easier for the team to understand, develop, test, and update without causing unintended side effects (regression).

3. Technology Flexibility: This pattern allows for the flexibility to use different technology stacks for different services if needed (e.g., using Python for the Reporting Service while the core services use Java or Node.js).

4. Fault Isolation: If one non-critical service fails (e.g., the Notification Service), it does not bring down the entire application. The rest of the system can remain operational.

ARCHITECTURE MODEL



4- DATA DESIGN

4.1 Database Description

The EventHub application uses a relational SQL database to ensure data integrity and clear relationships between entities such as Users, Events, Sessions, and Registrations.
Following the microservice architecture, each module (User, Event, Ticketing) manages its own database to enable independent scaling and maintenance.

4.1.1 User Database

Technology: MySQL / PostgreSQL
Purpose: Stores user accounts, authentication data, and role-based attributes for Admins, Organizers, Attendees, and Speakers.

Advantages:
• Secure credential and role handling
• Clear separation between user types
• Supports scalable authentication

4.1.2 Event Database

Technology: MySQL / PostgreSQL
Purpose: Stores event details, sessions, ticket types, and event status.

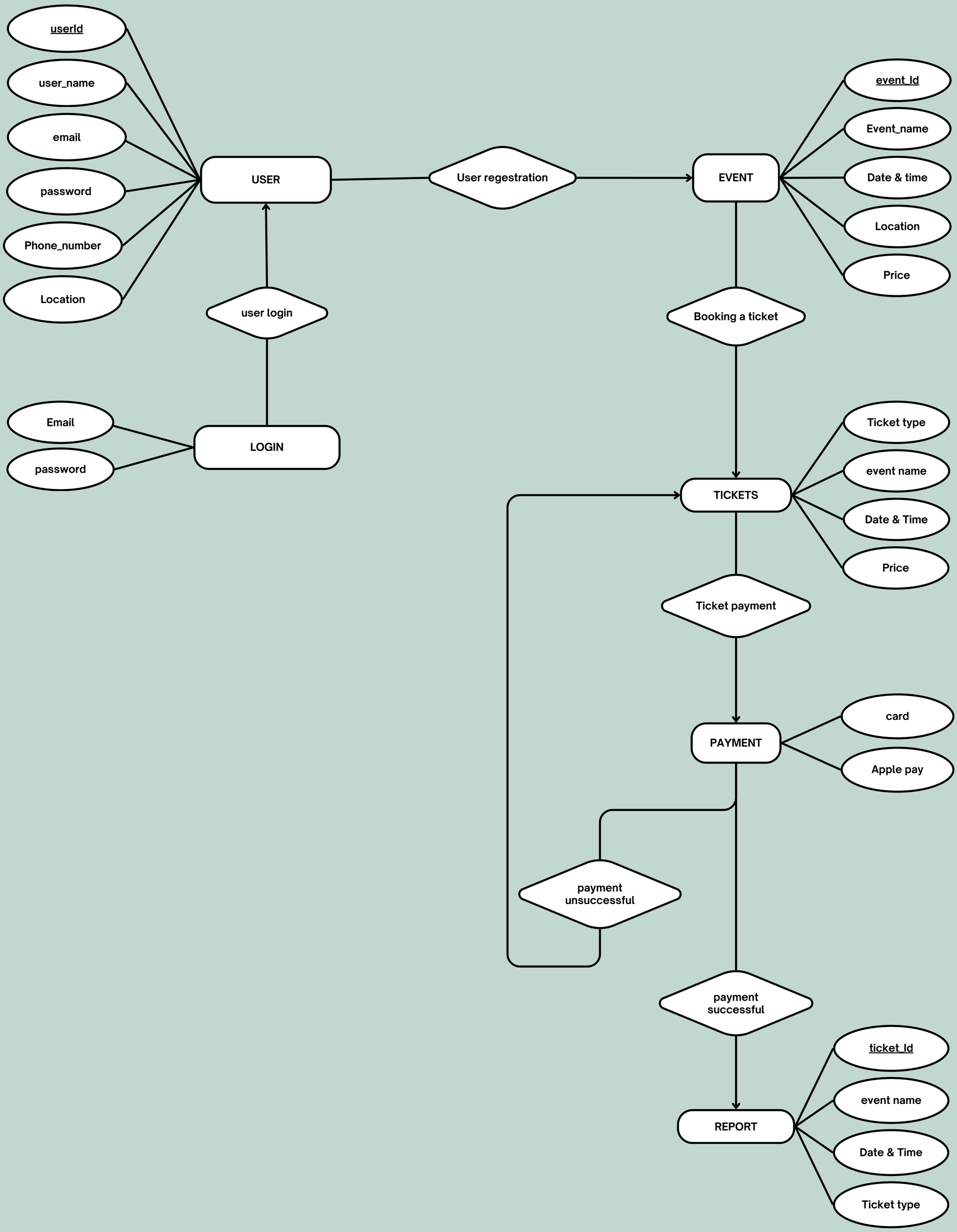
Features:
• One-to-many relations (Event → Sessions / Tickets)
• Tracks event capacity and availability
• Supports event states (Draft, Approved, Ongoing, Completed)

4.1.3 External Data Sources

• Payment Gateway API: Secure online payments (Stripe / HyperPay)
• Email Service API: Sends confirmations and QR-code tickets
• Calendar API: Adds events/sessions to user calendars

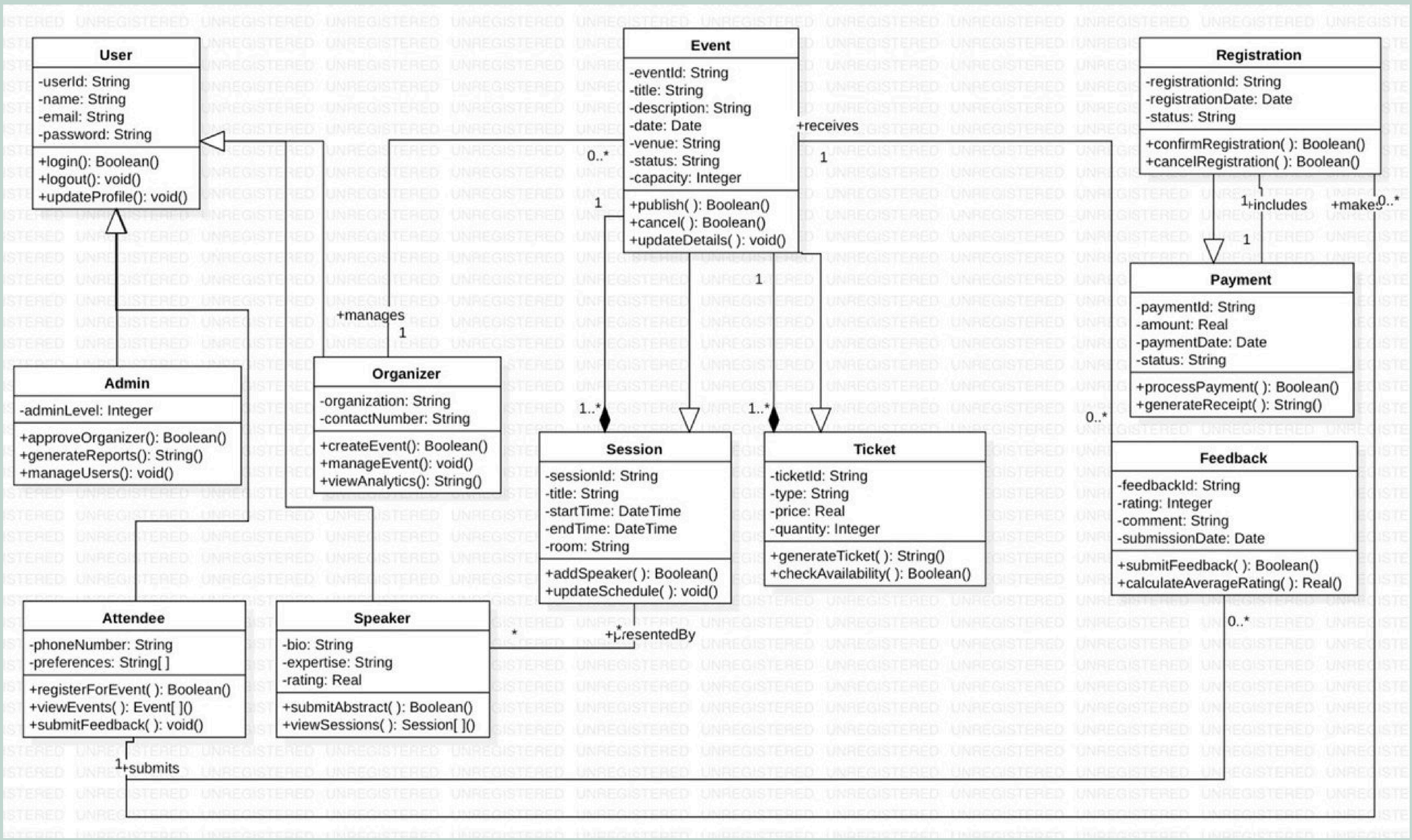
4- DATA DESIGN

4.2 Data structure

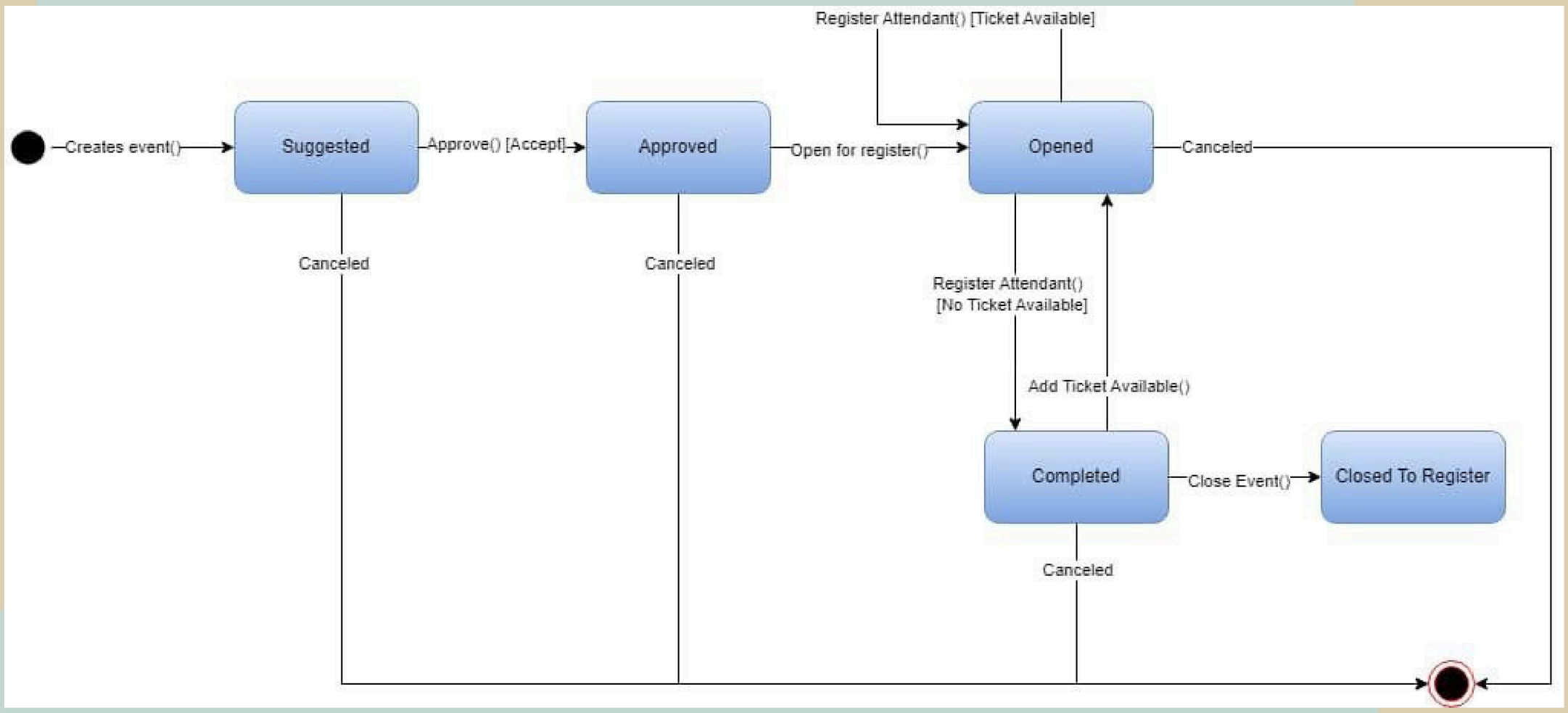


5. COMPONENT DESIGN

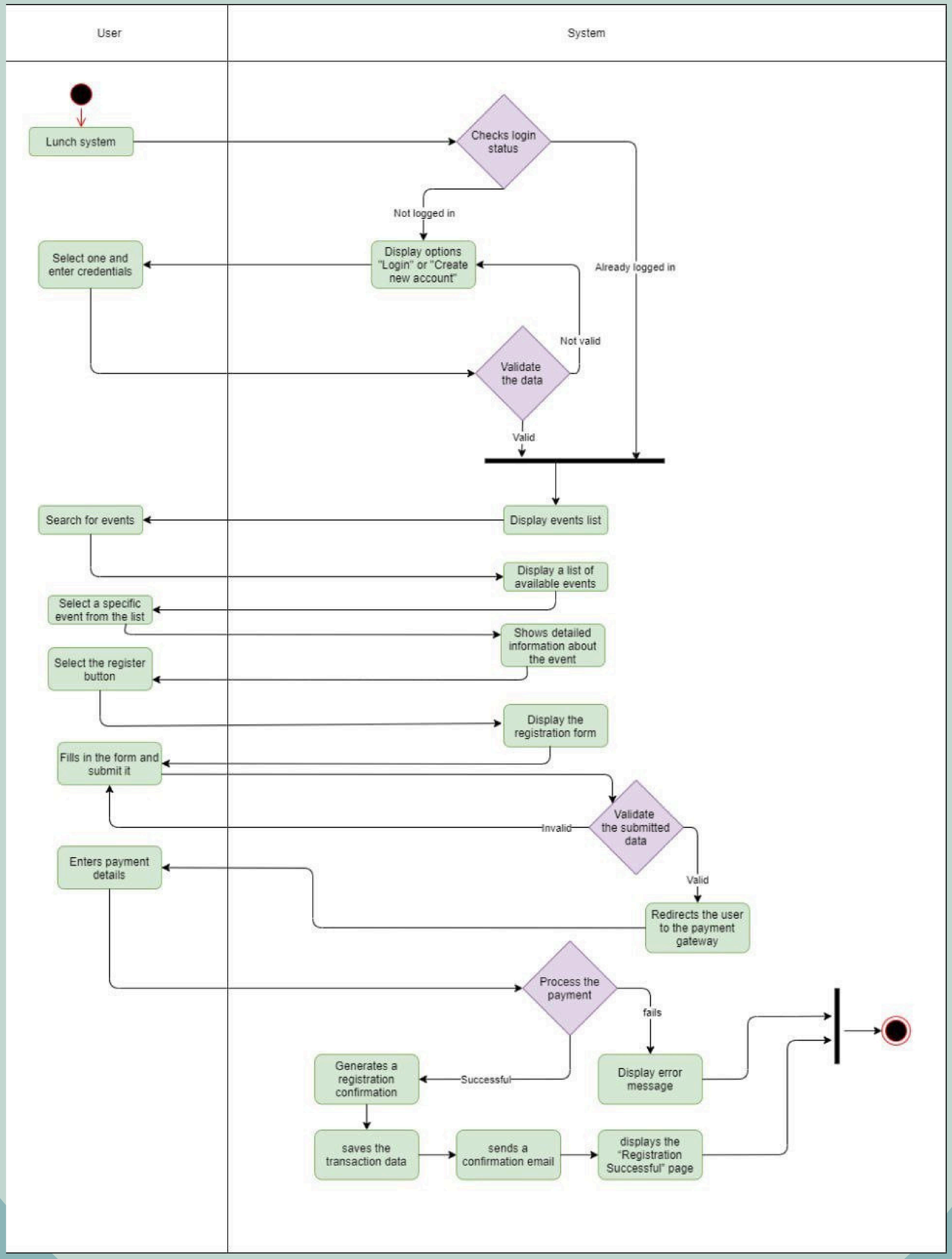
5.1 CLASS DIAGRAM



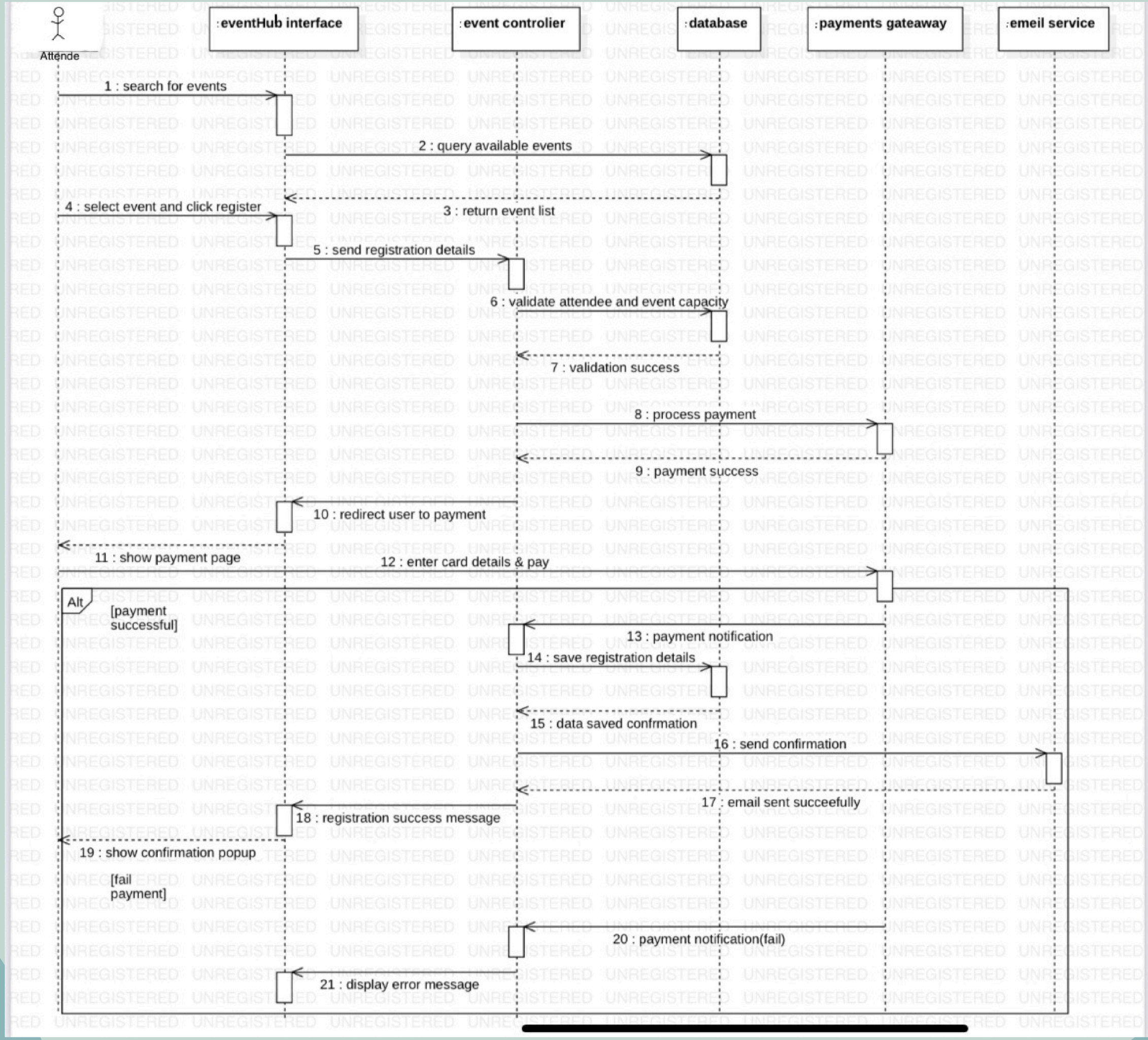
5.2 STATE DIAGRAM



5.3 ACTiViTY DiAGRAM



5.4 SEQUENCE DIAGRAMS



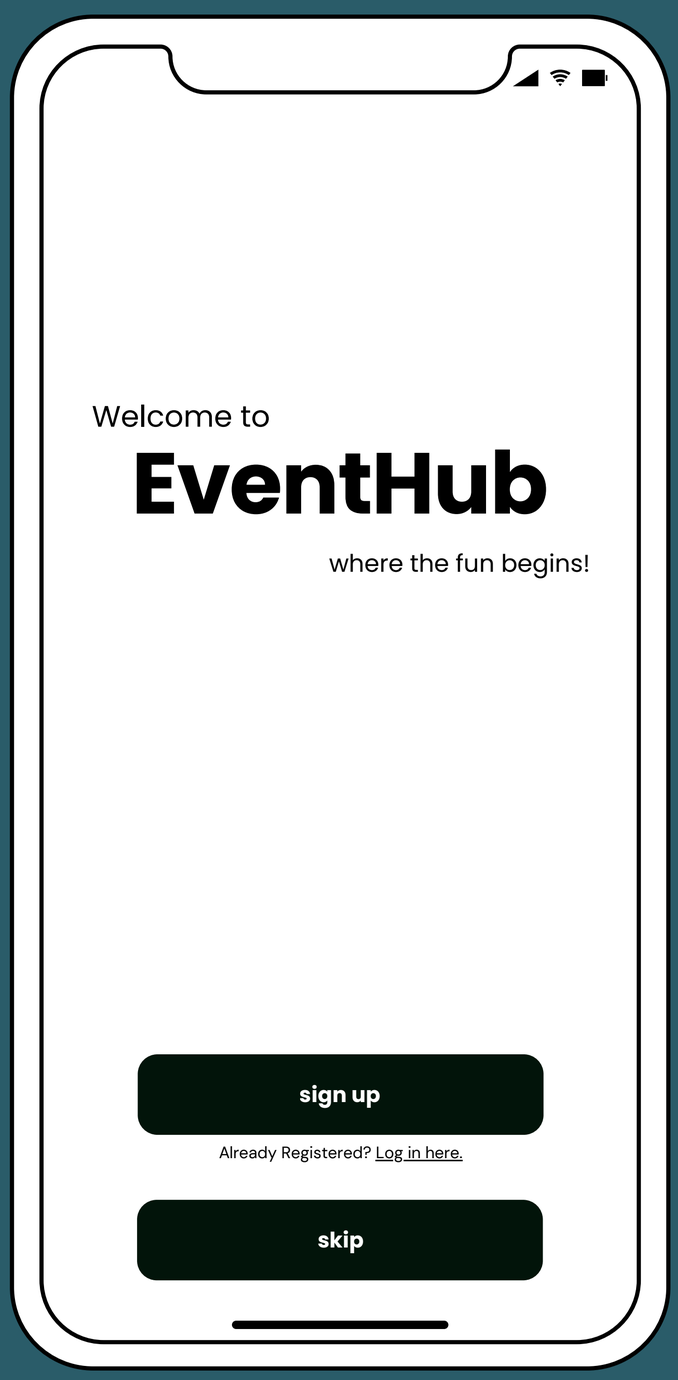
6. HUMAN INTERFACE DESIGN

6.1 Overview of the User Interface

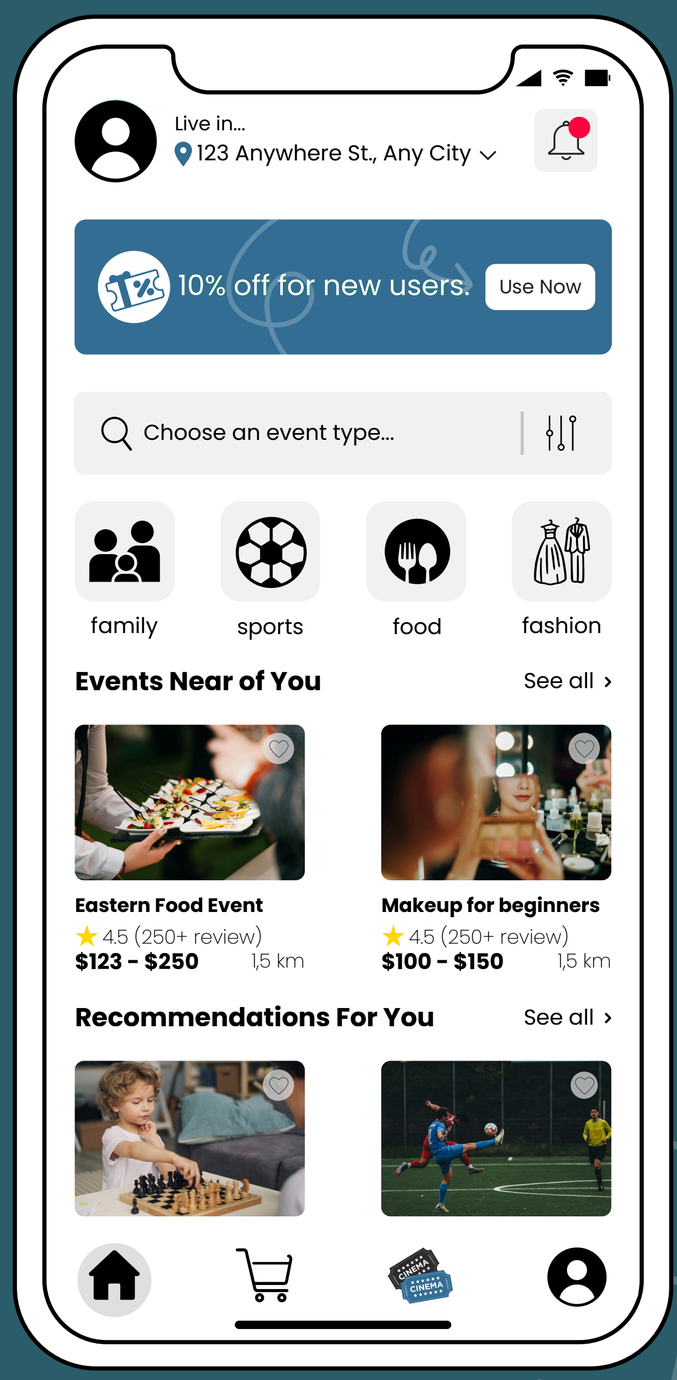
The EventHub interface is designed to be simple and intuitive, allowing users to browse and find events easily using filters such as date, location, and type. When an event is selected, the platform displays key details including the schedule, speakers, venue, and ticket options. The registration process guides users smoothly through ticket selection, personal information entry, and secure payment. Once completed, a confirmation page shows the event details, ticket type, and a QR code, along with a thank-you message.

Users can access a personal dashboard to view registered events, download tickets, receive updates, or cancel bookings. The platform's consistent layout, clear menus, and responsive design ensure a smooth experience for all users.

6.2 Detail design of user interface



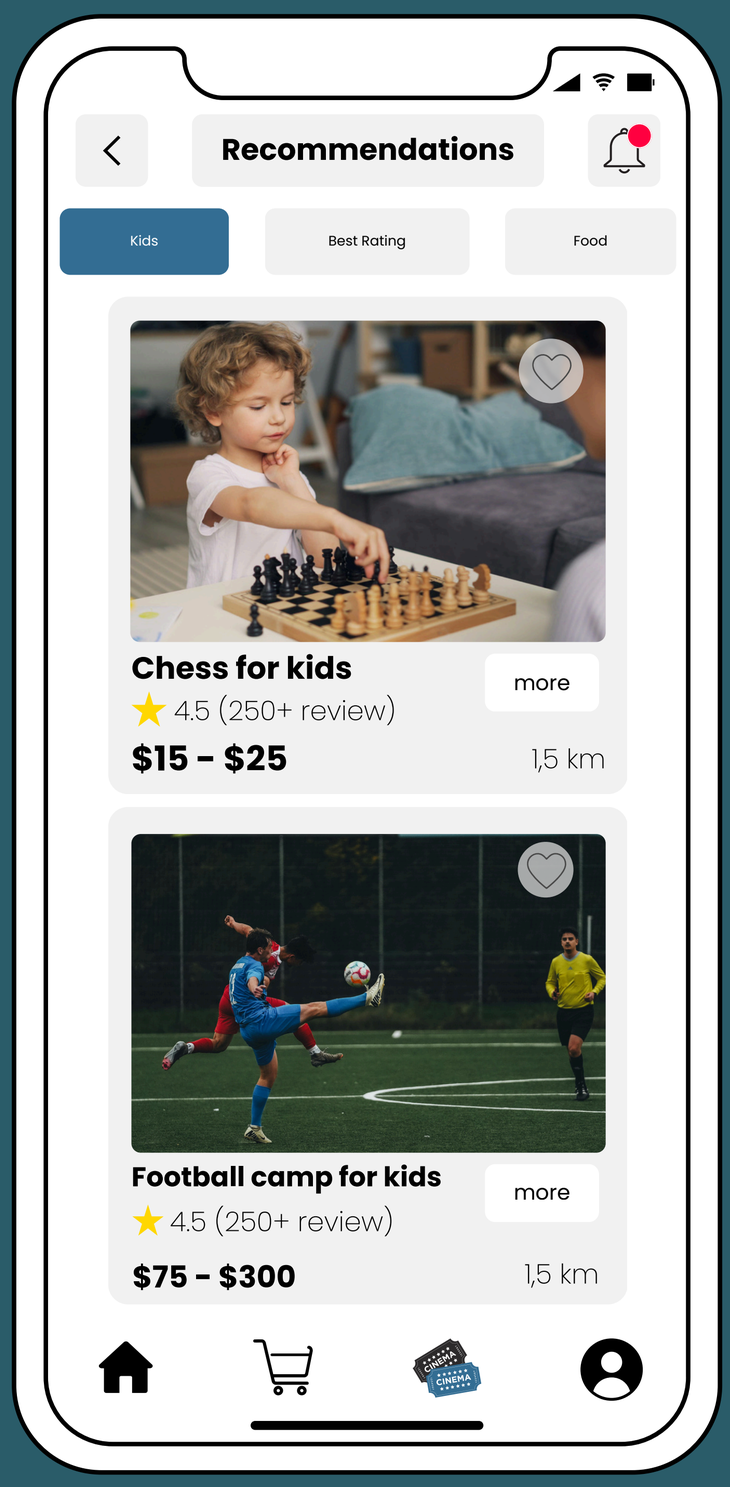
Welcoming Page



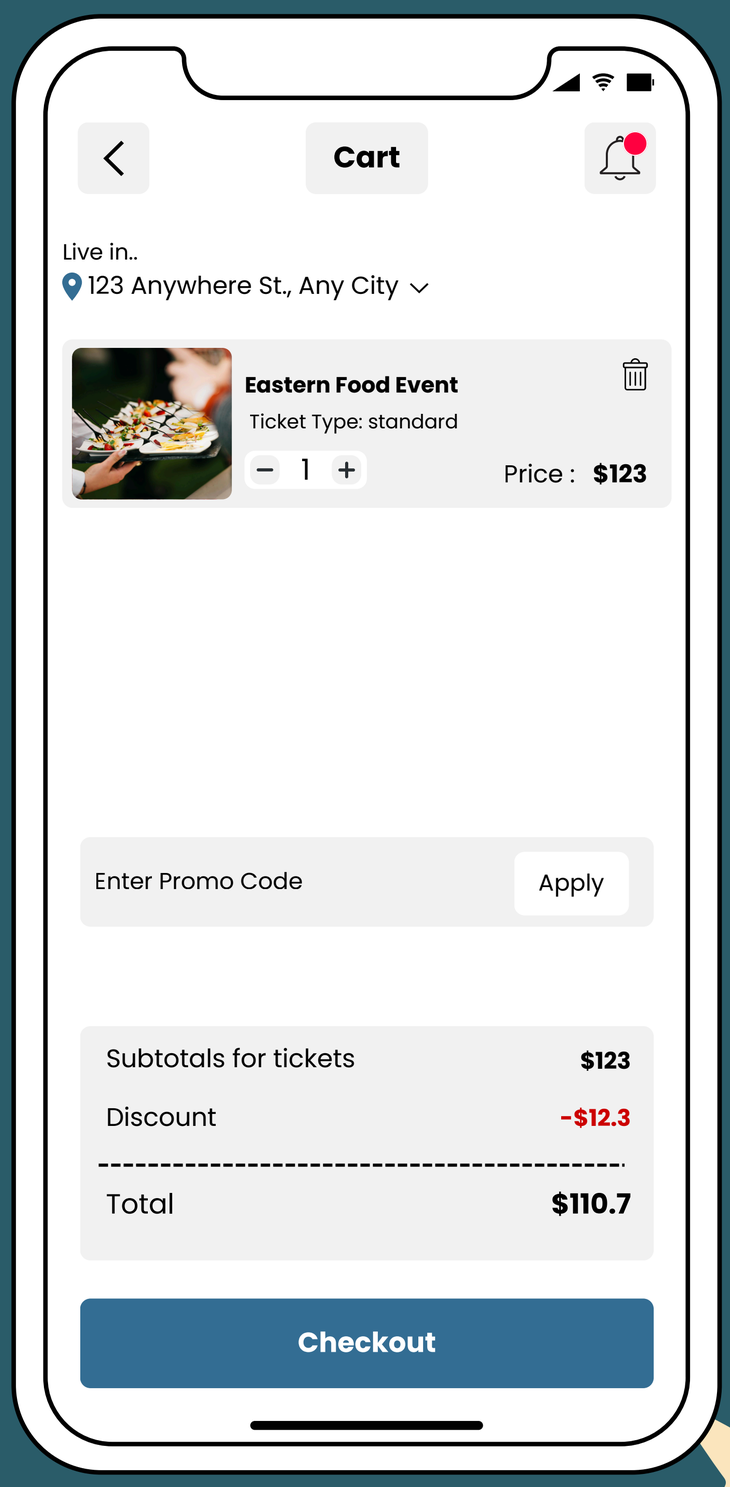
Home Page

6. HUMAN INTERFACE DESIGN CONT.

6.2 Detail design of user interface



Events page



Cart page

The interface design provides an easy and helpful layout that allows users to navigate the app smoothly. It offers clear menus and simple controls, making it effortless for users to scroll through events and find the ones that match their interests. The design keeps everything organized and accessible, ensuring users can browse, register, and manage their bookings without confusion.

7.1 REPOSITORY STRUCTURE

```
├EventHub/
├──── documentation/
│     ├──── SRS/
│     │     ├──── EventHub_SRS_Final.pdf
│     │     ├──── meeting_minutes/
│     │     │     ├──── meeting_1_agenda_minutes.pdf
│     │     │     ├──── meeting_2_agenda_minutes.pdf
│     │     │     └──── meeting_attendance_records.pdf
│     │     └──── requirements_drafts/
│     ├──── SDD/
│     │     ├──── EventHub_SDD_Final.pdf
│     │     └──── UML_diagrams/
│     │           ├──── use_case_diagram.png
│     │           ├──── class_diagram.png
│     │           ├──── sequence_diagram.png
│     │           ├──── state_diagram.png
│     │           └──── activity_diagram.png
│     └──── presentations/
│           └──── EventHub_Final_Presentation.pdf
├──── project_management/
│     ├──── project_plan.md
│     ├──── task_assignments.md
│     └──── contribution_log.md
├──── assets/
│     ├──── images/
│     ├──── wireframes/
│     └──── mockups/
└──── README.md
```

REPOSITORY ORGANIZATION

7.2 Branching Strategy

- main : Production-ready documentation only
- develop : Integration branch for ongoing work
- feature/ : Feature branches (e.g., `feature/srs-development`, `feature/uml-diagrams`)
- docs/: Documentation-specific branches

7.3 Naming Conventions

- Branches : `feature/feature-name`, `docs/document-type`
- Commits: Use descriptive messages (e.g., "Add functional requirements section")
- Files : Use descriptive names (e.g., `srs_introduction.md`, `class_diagram_v2.png`)

7.4 Team Access & Responsibilities

- **Admin : (Team Lead) - Full repository access**
- **Maintain : All team members - Push access to develop and feature branches**
- **Contributors:**
  - **ARYAM -** Ethics, Activity diagram, System overview SRS & SDD, SRS requirements specification, Architecture design
  - **JANA -** External requirements, Architecture model, Interface design, Data structure
  - **REEMA -** Functional requirements, State diagram, Data design, Work section, Development Process Model
  - **NOURA Alomair -**Functional requirements, Class diagram, Introduction SRS & SDD, Repository organization
  - **WASAYIF Alsalammah -** Non-functional requirements, UML use case, Sequence diagram, Meeting and agenda file, Requirements engineering SRS

7.5 File Organization Guidelines

- All final documents in PDF format
- Meeting minutes include agendas and attendance records
- UML diagrams stored in high-resolution PNG format
- Regular backups to cloud storage weekly
- Maintain version control for all documents
- Document all major changes in commit messages