## **System Design Document (SDD) for Eventify**

## **1. Introduction**

## This section introduces Eventify, a web-based event management and ticketing platform designed to allow users to discover events, purchase tickets, and manage event-related activities. It outlines the purpose, scope, and intended audience of this System Design Document (SDD), ensuring alignment with the system’s functional and non-functional requirements.

**1.1 Purpose**

The primary purpose of this System Design Document (SDD) is to describe **how Eventify will be designed and implemented**. It details the system architecture, major software components, data structures, interfaces, and key technical decisions.

This document serves as a **technical blueprint** for developers and stakeholders, ensuring that Eventify is built according to the defined requirements while providing a scalable, secure, and user-friendly event management solution.

**1.2 Scope**

Eventify is a **web-based platform** that enables users to browse events, register accounts, purchase tickets, and manage event-related activities. The system supports different user roles, including Guests, Registered Users (Customers), and Administrators.

**In Scope (What the system will do)**

* **Event Browsing & Discovery**
  + Allow guests and registered users to browse upcoming events.
  + Support searching and filtering events by keywords or categories.
  + Display detailed event information such as date, location, price, and availability.
* **User Management**
  + User registration and authentication.
  + Role-Based Access Control (RBAC) for Guests, Registered Users, and Administrators.
  + Secure login and session management.
* **Ticket Purchasing**
  + Allow registered users to select events and purchase tickets.
  + Validate payment details and confirm ticket purchases.
  + Display purchased ticket information in the user dashboard.
* **Administrative Management**
  + Enable administrators to add, edit, and delete events.
  + Allow administrators to manage users (create, update, or remove accounts).
  + Maintain system logs for monitoring and auditing purposes.
* **System Logging & Error Handling**
  + Log critical system actions such as event management and user operations.
  + Handle validation, authorization, and database errors gracefully.

**Out of Scope (What the system will not do)**

* Event organization by external organizers (handled exclusively by Admin users).
* Physical ticket printing or offline ticket sales.
* Native mobile applications (system is web-based and responsive only).
* Advanced analytics or AI-based event recommendations.
* Integration with external payment gateways beyond basic card validation (if not implemented).
* Real-time chat or messaging between users.

**1.3 Intended Audience**

This document is intended for the following stakeholders:

* **Development Team**  
  To guide system implementation, architecture decisions, and component interactions.
* **Project Managers**  
  To track development progress and ensure alignment between requirements and implementation.
* **Quality Assurance (QA) Team**  
  To derive test cases, validation scenarios, and testing strategies based on the system design.
* **System Administrators**  
  To understand system deployment, configuration, and operational responsibilities.
* **Academic Evaluators / Instructors**  
  To assess the completeness, clarity, and correctness of the system design.

**1.4 Definitions and Acronyms**

* **Eventify**A web-based event management and ticketing system that enables users to browse events, register accounts, and purchase tickets online.
* **Guest (Unregistered User)**A visitor who does not have an account and is allowed to browse and view event details but must register or log in to complete a ticket purchase.
* **Registered User (Customer)**An authenticated user who can browse events, purchase tickets, and view purchased ticket information through a personal dashboard.
* **Administrator (Admin)**A privileged system user responsible for managing events, users, and system logs, as well as overseeing overall platform operations.
* **Event**A scheduled activity published on the Eventify platform that includes details such as title, date, location, ticket price, and availability.
* **Ticket**A digital record generated after a successful purchase that grants a registered user access to a specific event.
* **Checkout**The process through which a registered user confirms selected tickets and completes the purchase.
* **Dashboard**A personalized interface that displays relevant information based on the user role, such as purchased tickets for customers or management tools for administrators.
* **Authentication**The process of verifying a user’s identity using valid credentials such as email and password.
* **Authorization**The process of determining whether an authenticated user has permission to access specific system features or perform certain actions**.**
* **RBAC (Role-Based Access Control)**A security mechanism that restricts system access and functionality based on predefined user roles, such as Guest, User, or Admin.
* **Validation**The process of checking user input or system data to ensure correctness, completeness, and compliance with defined rules before processing.
* **System Log**A record of significant system actions and events used for auditing, monitoring, and troubleshooting purposes.

**2. System Overview**

Eventify is a **web-based event management and ticketing system** designed to streamline the discovery, booking, and administration of events.

The system follows a **role-based architecture**, supporting three main user roles: **Guests**, **Registered Users (Customers)**, and **Administrators**. Each role interacts with the system through controlled workflows to ensure security, usability, and data integrity.

At its core, Eventify revolves around **Event Management**, **User Management**, and **Ticket Purchasing**, supported by logging and validation mechanisms.

**Major System Components**

1. **User Management Module**  
   Handles user registration, authentication, session management, and role-based access control (RBAC). It ensures secure access to system features based on user roles.
2. **Event Management Module**  
   Responsible for creating, updating, deleting, and displaying events. This module ensures event availability, validation of event data, and controlled access for administrators.
3. **Ticketing & Checkout Module**  
   Manages ticket selection, payment validation, purchase confirmation, and ticket visibility in the user dashboard.
4. **Event Browsing & Search Module**  
   Enables guests and users to browse events, perform searches, and view detailed event information.
5. **Logging & Monitoring Module**  
   Records system actions such as administrative changes, access attempts, and critical errors for auditing and system reliability.
6. **Error Handling & Validation Module**  
   Ensures that invalid inputs, authorization failures, and system errors are detected and communicated clearly to users.

## **3.Data Design**

**Database Name:** eventify

* **Entity-Relationship (ER) Model**

This section describes how the data entities interact.

**Users ↔ Tickets (1:N):** A single user can purchase multiple tickets (for the same event or different events).

* + Constraint: If a user is deleted, their tickets are also deleted (ON DELETE CASCADE).
* **Events ↔ Tickets (1:N):** A single event can have many tickets sold for it, but a specific ticket is for only one event.
  + Constraint: If an event is deleted, its associated tickets are also deleted (ON DELETE CASCADE).
* **Users ↔ Logs (1:N):** A single user can generate multiple system logs (login, updates).
  + Constraint: If a user is deleted, the log remains but the user\_id is set to NULL (ON DELETE SET NULL) to preserve audit history.

#### Table 1: users

Stores account information for administrators and standard users.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Type** | **Constraints** | **Description** |
| id | INT | **PK**, Auto Increment | Unique identifier for the user. |
| name | VARCHAR(100) | Not Null | Full name of the user. |
| email | VARCHAR(150) | **Unique**, Not Null | User's email address (login credential). |
| password | VARCHAR(255) | Not Null | Hashed password. |
| role | VARCHAR(50) | Not Null | Permission level (e.g., 'admin', 'user'). |
| status | VARCHAR(50) | Not Null | Account status (e.g., 'active', 'not active', 'banned'). |
| created\_at | TIMESTAMP | Default Current Time | Account creation timestamp. |

**Table 2: events**

Stores details about the events available for booking.

|  |  |  |  |
| --- | --- | --- | --- |
| Column | Type | Constraints | Description |
| id | INT | **PK**, Auto Increment | Unique identifier for the event. |
| title | VARCHAR(150) | Not Null | Name of the event. |
| venue | VARCHAR(150) | Not Null | Location of the event. |
| date | DATE | Not Null | Date of the event. |
| time | TIME | Not Null | Start time of the event. |
| category | VARCHAR(100) | Not Null | Event type ('Theatre', 'Sports'). |
| capacity | INT | Not Null | Maximum number of attendees. |
| price | DECIMAL(10,2) | Not Null | Cost per ticket. |
| status | ENUM | Default 'active' | States: 'active', 'cancelled', 'sold\_out'. |
| has\_seats | tinyint(1) | Default 'Yes' | Has seats = 1 No seats = 0 |
| created-at | TIMESTAMP | Default Current Time | Account creation timestamp. |

**Table 3: tickets**

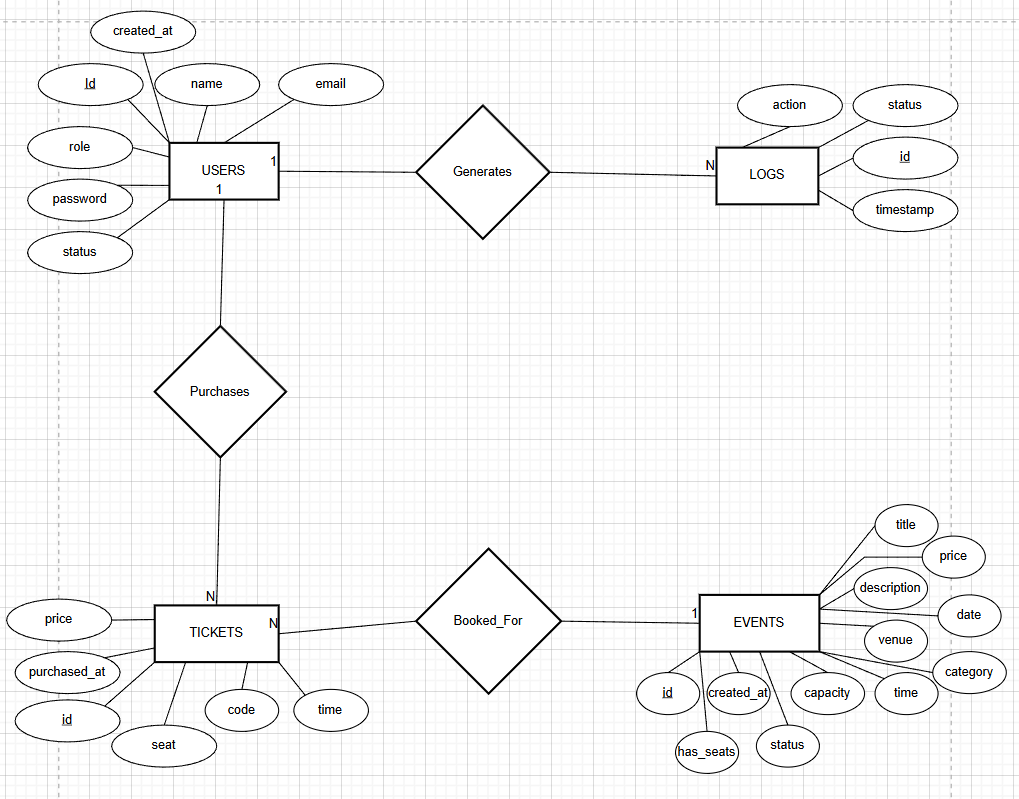
The transaction table links Users and Events.

|  |  |  |  |
| --- | --- | --- | --- |
| Column | Type | Constraints | Description |
| id | INT | **PK**, Auto Increment | Unique ticket identifier. |
| user\_id | INT | **FK** (Ref users.id) | The user who bought the ticket. |
| event\_id | INT | **FK** (Ref events.id) | The event the ticket is for. |
| code | VARCHAR(100) | **Unique** | Unique booking reference code. |
| seat | VARCHAR(50) | Nullable | Specific seat number if applicable. |
| price | DECIMAL(10,2) | Not Null | The price paid of purchase. |
| purchased\_at | DATETIME | Default Current Time | Time of transaction. |
| time | TIME | Not Null | Start time of the event. |

**Table 4: logs**

Maintains an audit trail of system activities.

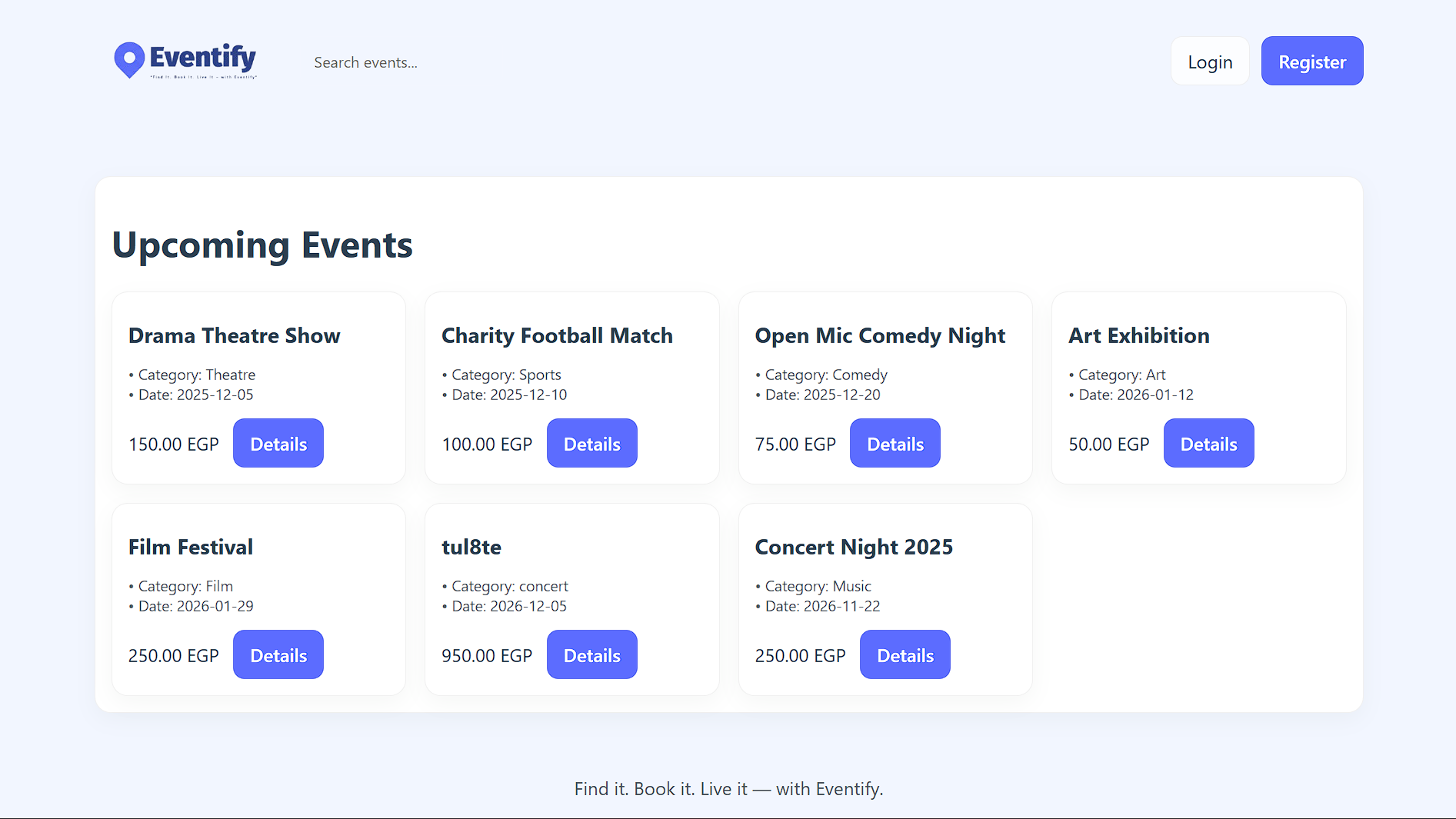
|  |  |  |  |
| --- | --- | --- | --- |
| Column | Type | Constraints | Description |
| id | INT | **PK**, Auto Increment | Unique log identifier. |
| user\_id | INT | **FK** (Ref users.id) | The user who performed the action (nullable). |
| action | VARCHAR(255) | Not Null | Description of activity (e.g., 'User logged in'). |
| status | ENUM | Default 'success' | Outcome: 'success', 'warning', 'error'. |
| timestamp | TIMESTAMP | Default Current Time | When the action occurred. |

* **Entity-Relationship (ER) diagram**

## **4.User Interface Design**

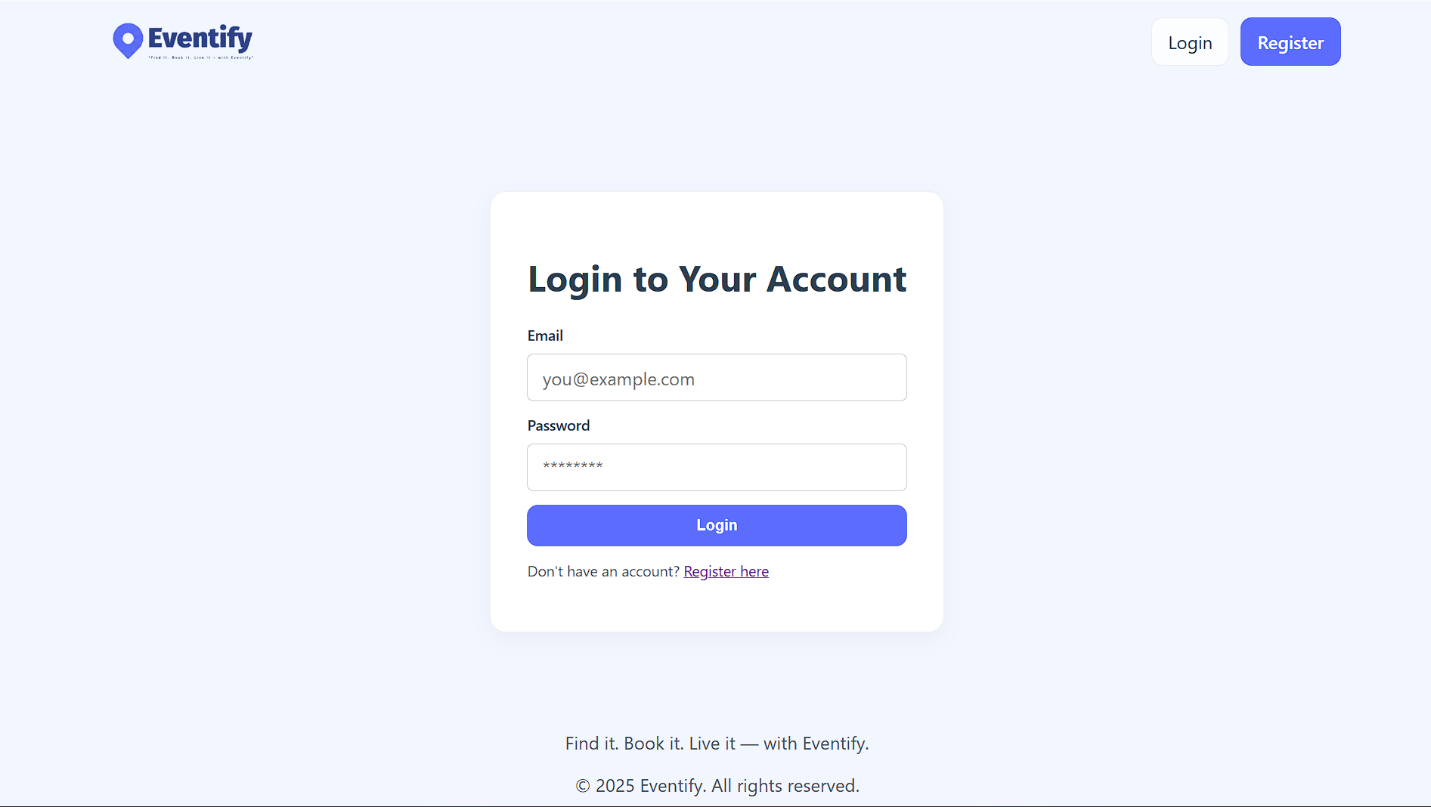
* **Guest**

**Home page**

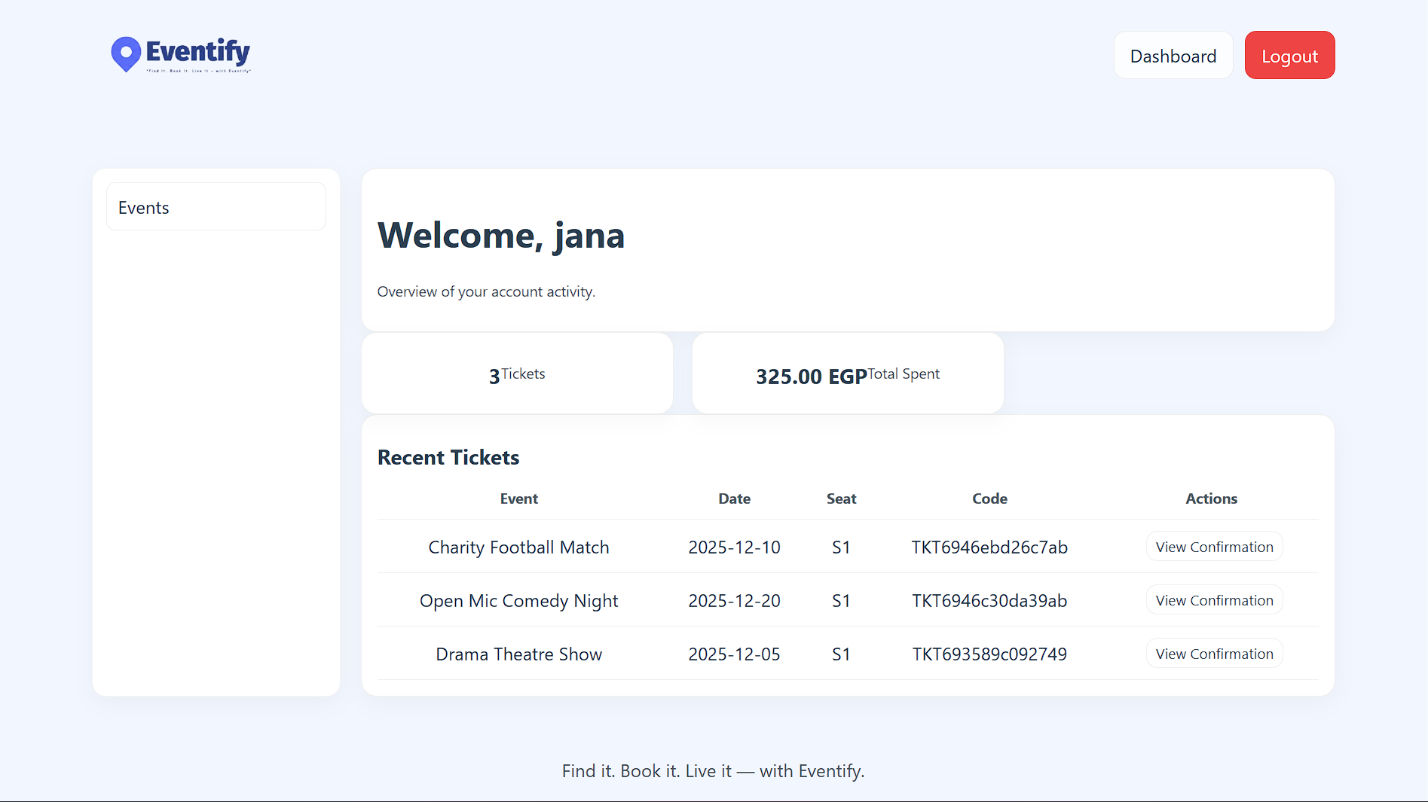
****

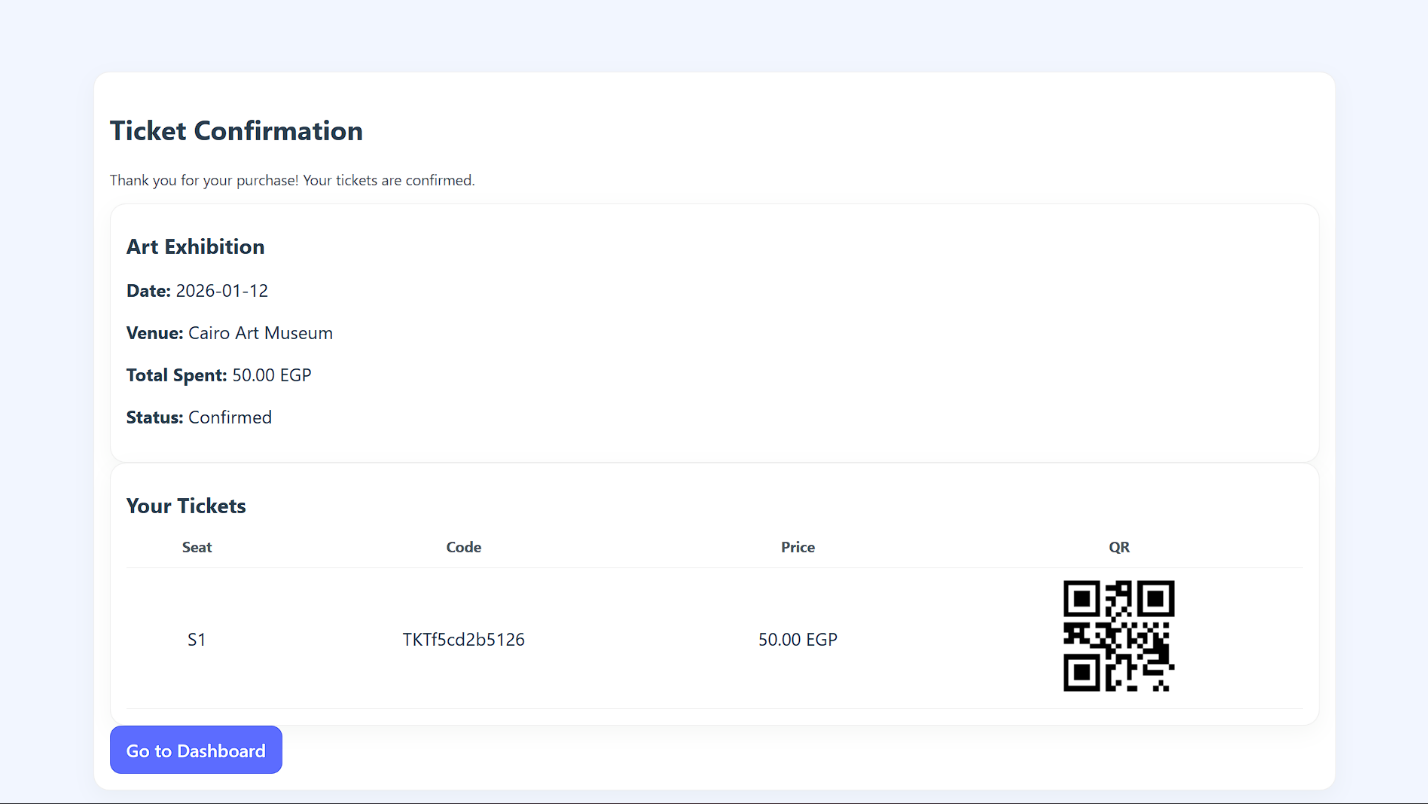
**Details of event if not registered/ logged**

**Registration**

**Log in**

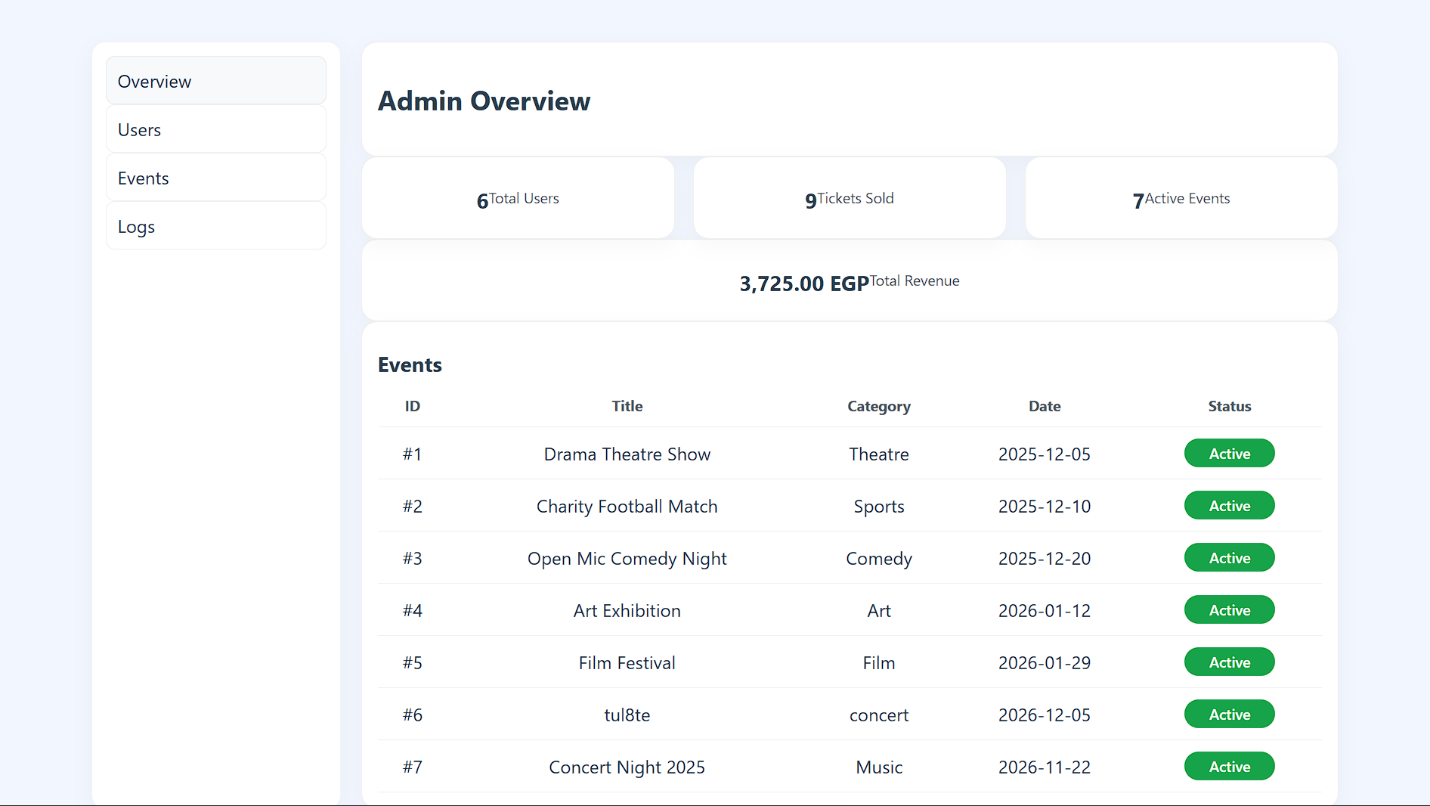
* **User**

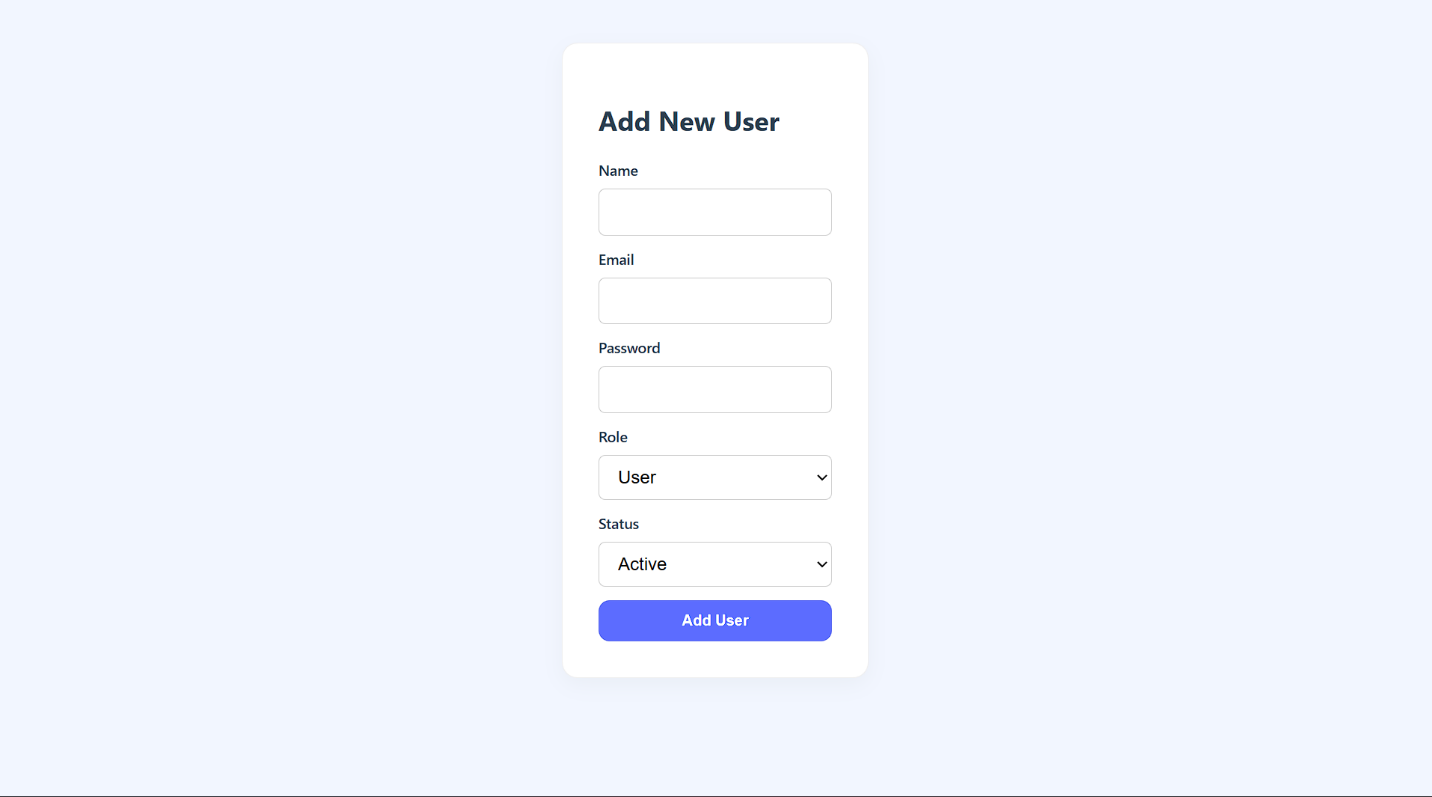
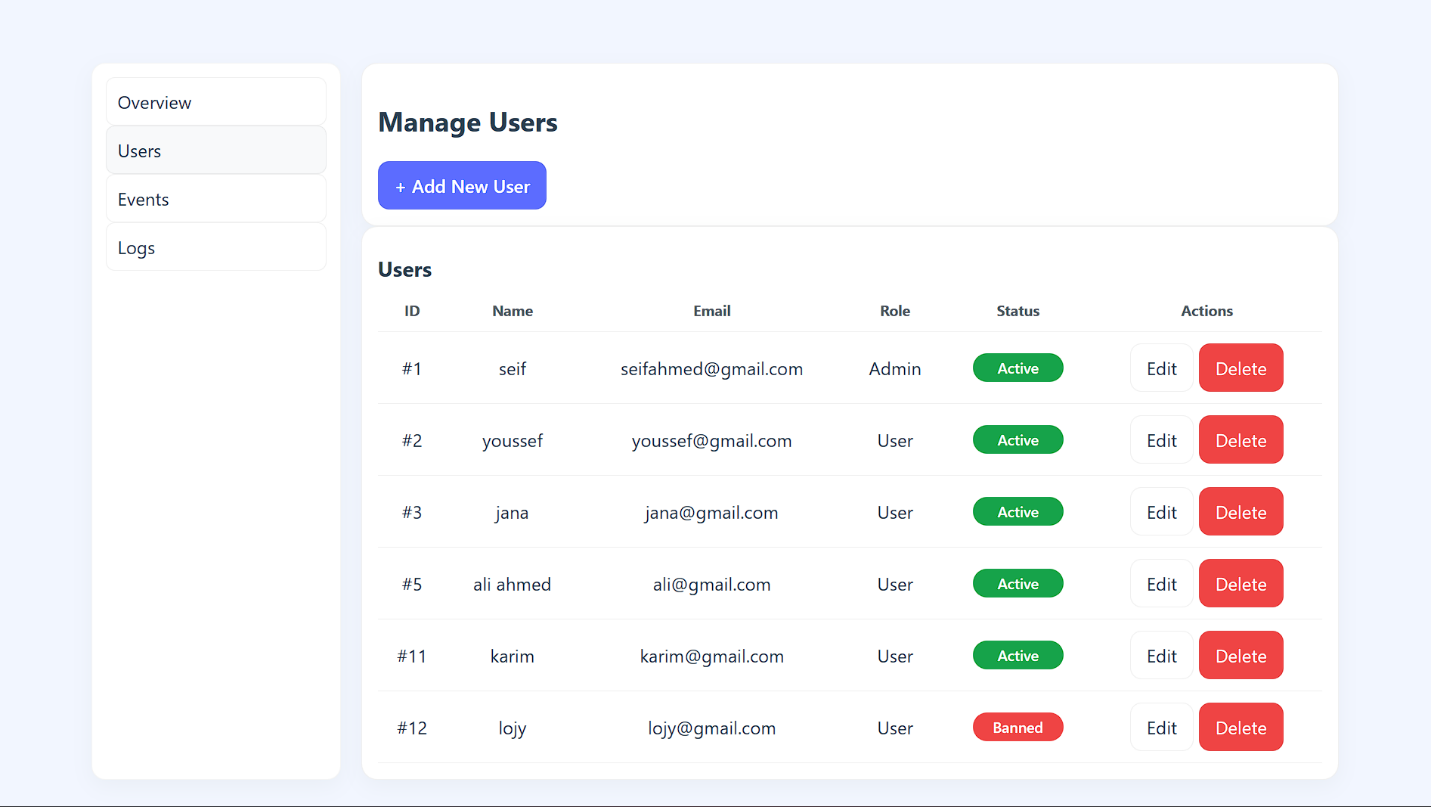
**Logged in as user**

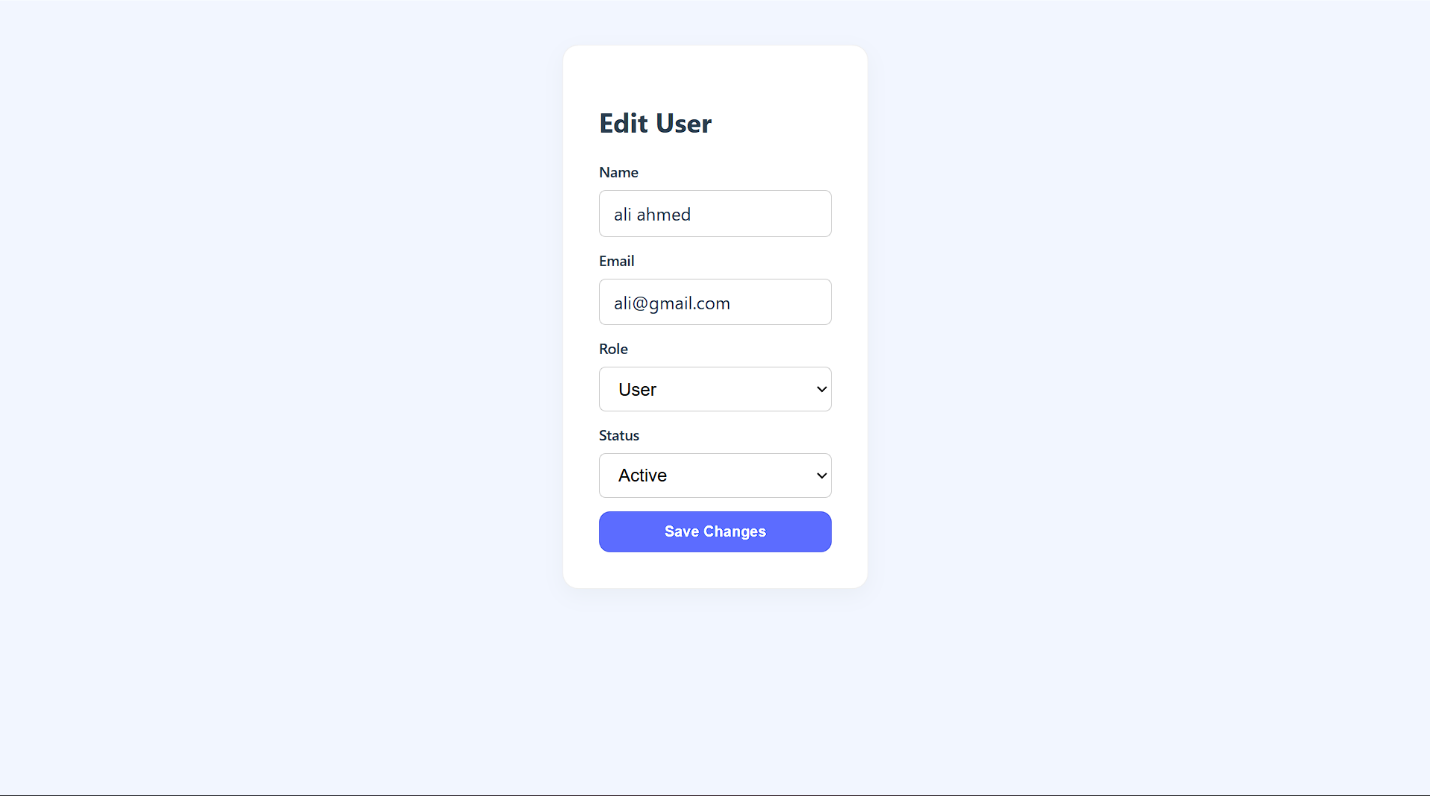
**Events details as logged in user  
  
After clicking on reserve ticket (checkout page)  
After confirming payment (confirmation ticket)**

**Returning to the dashboard of the user with the updated overview**

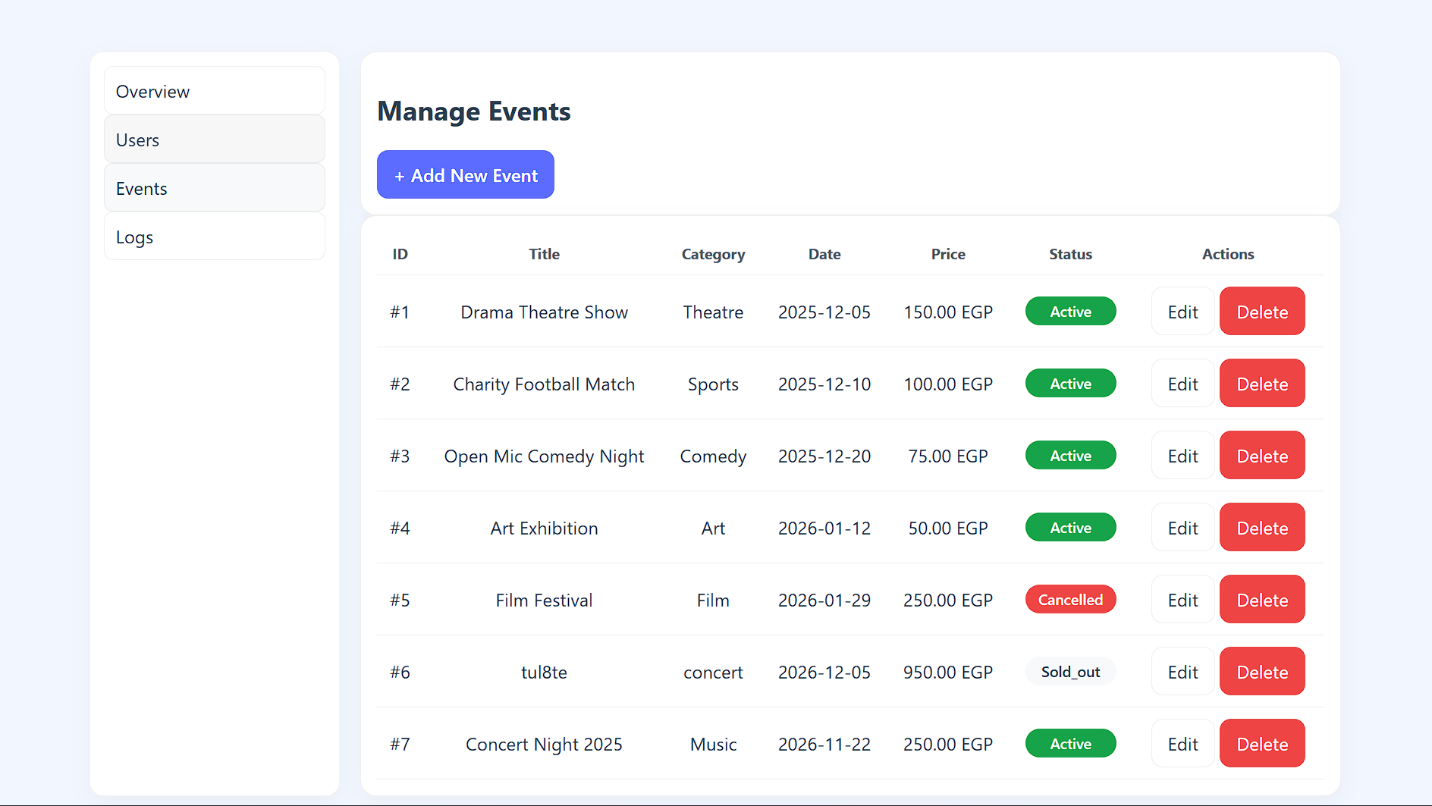
* **Admin**

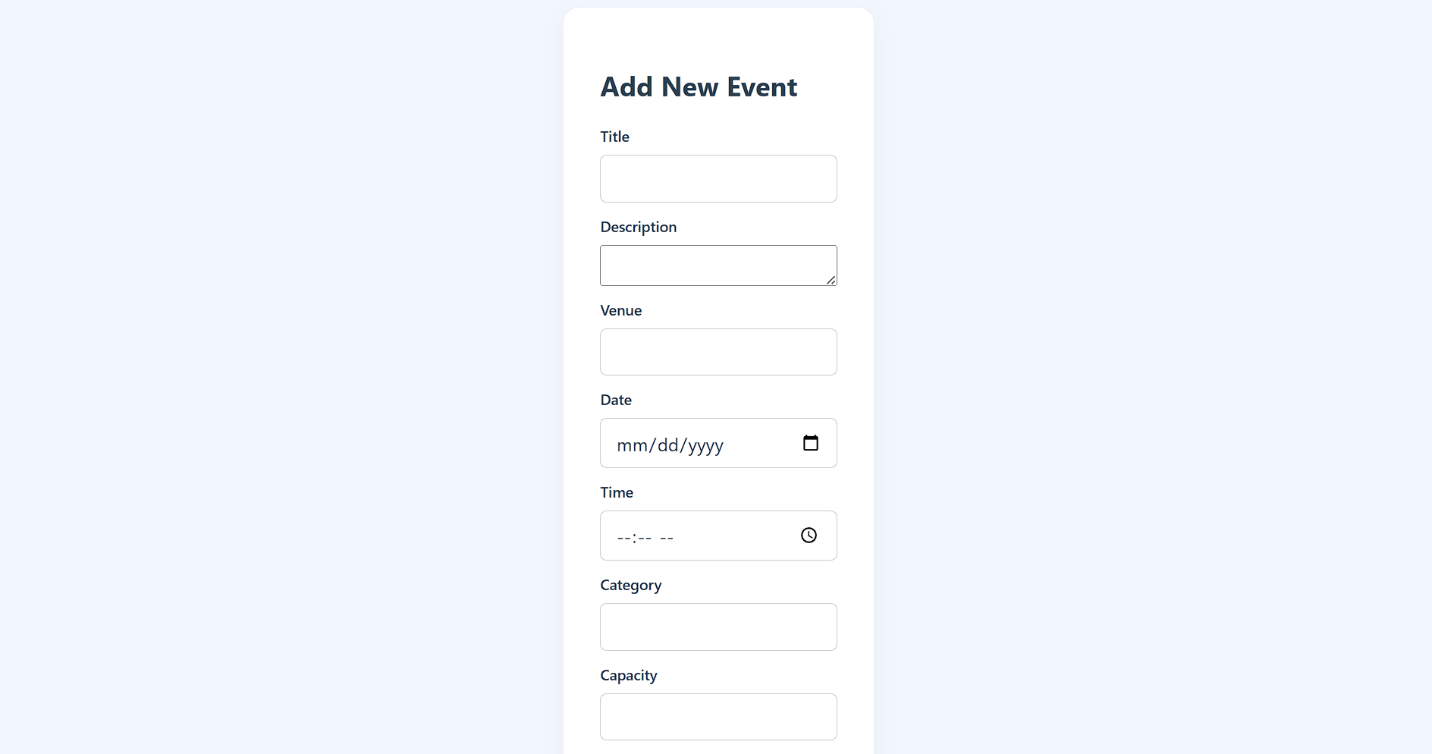
**Admin dashboard**

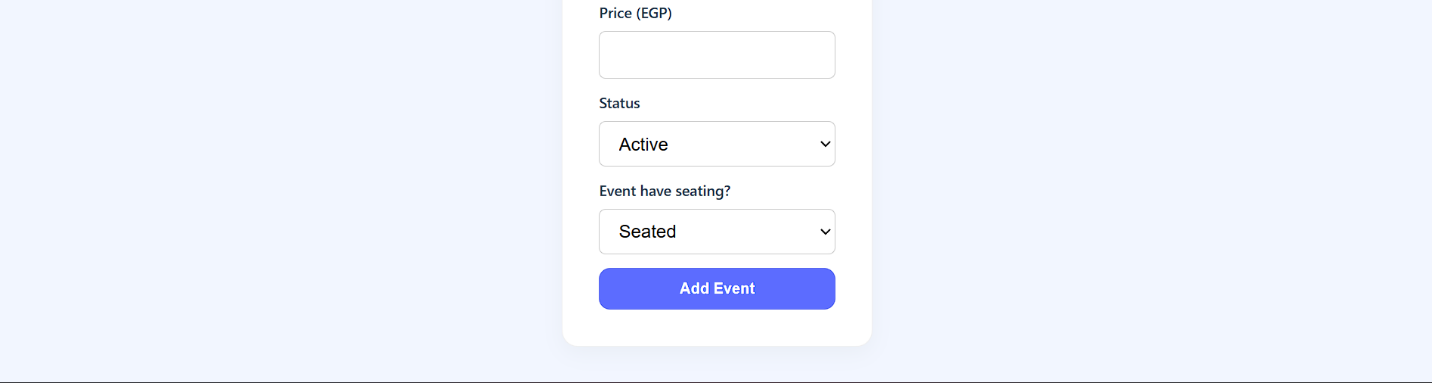
**Admin users**   
**Add new users**

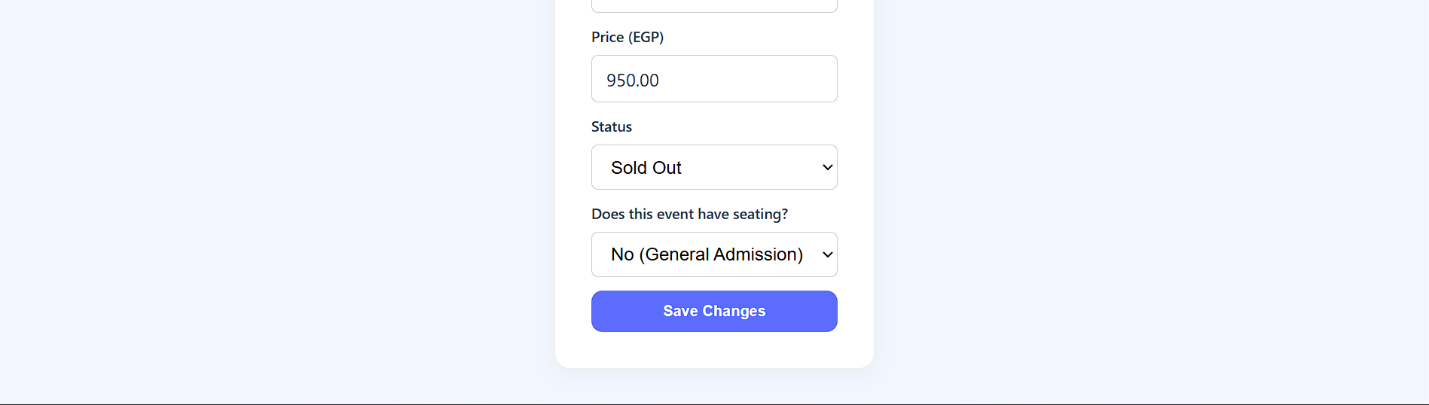
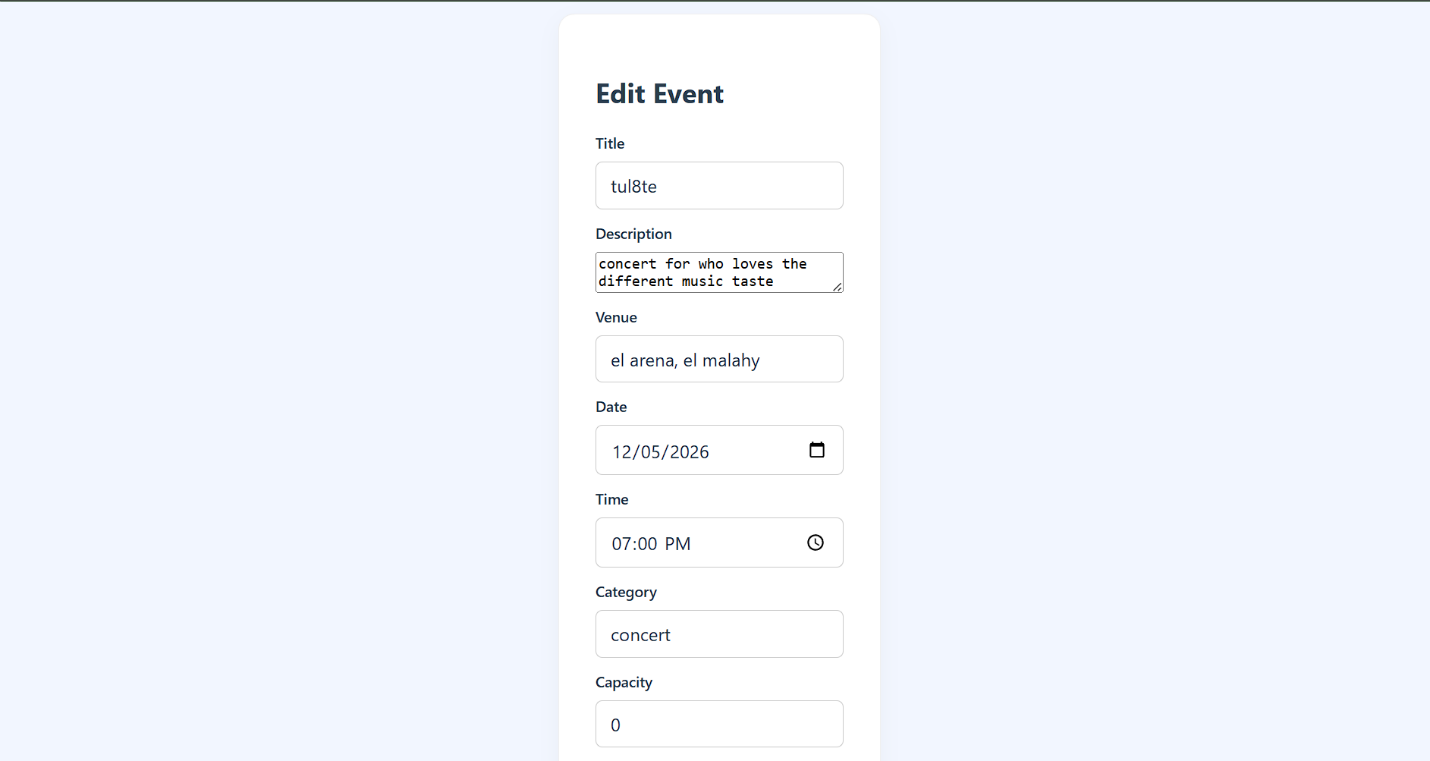
**Edit user**

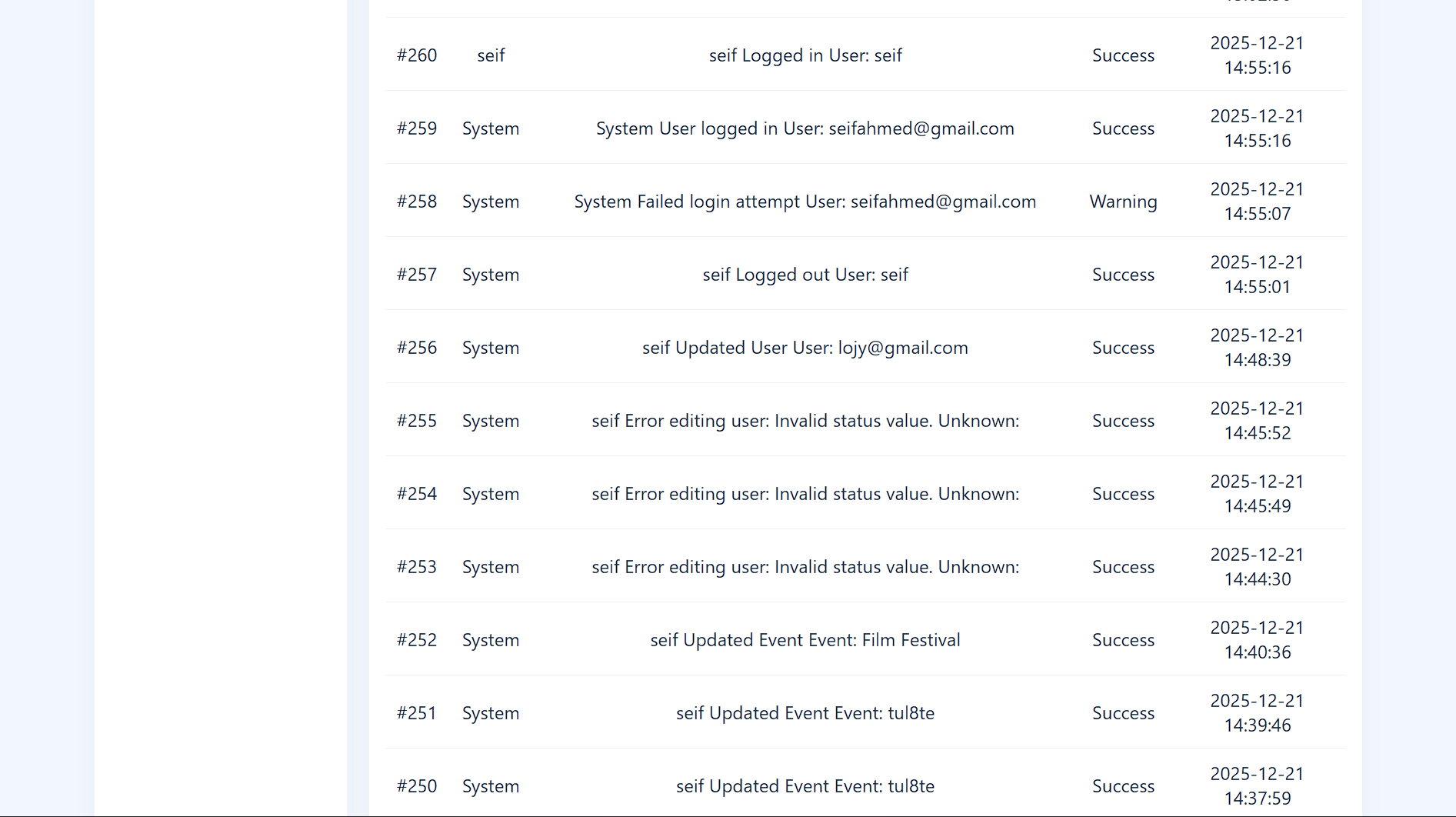
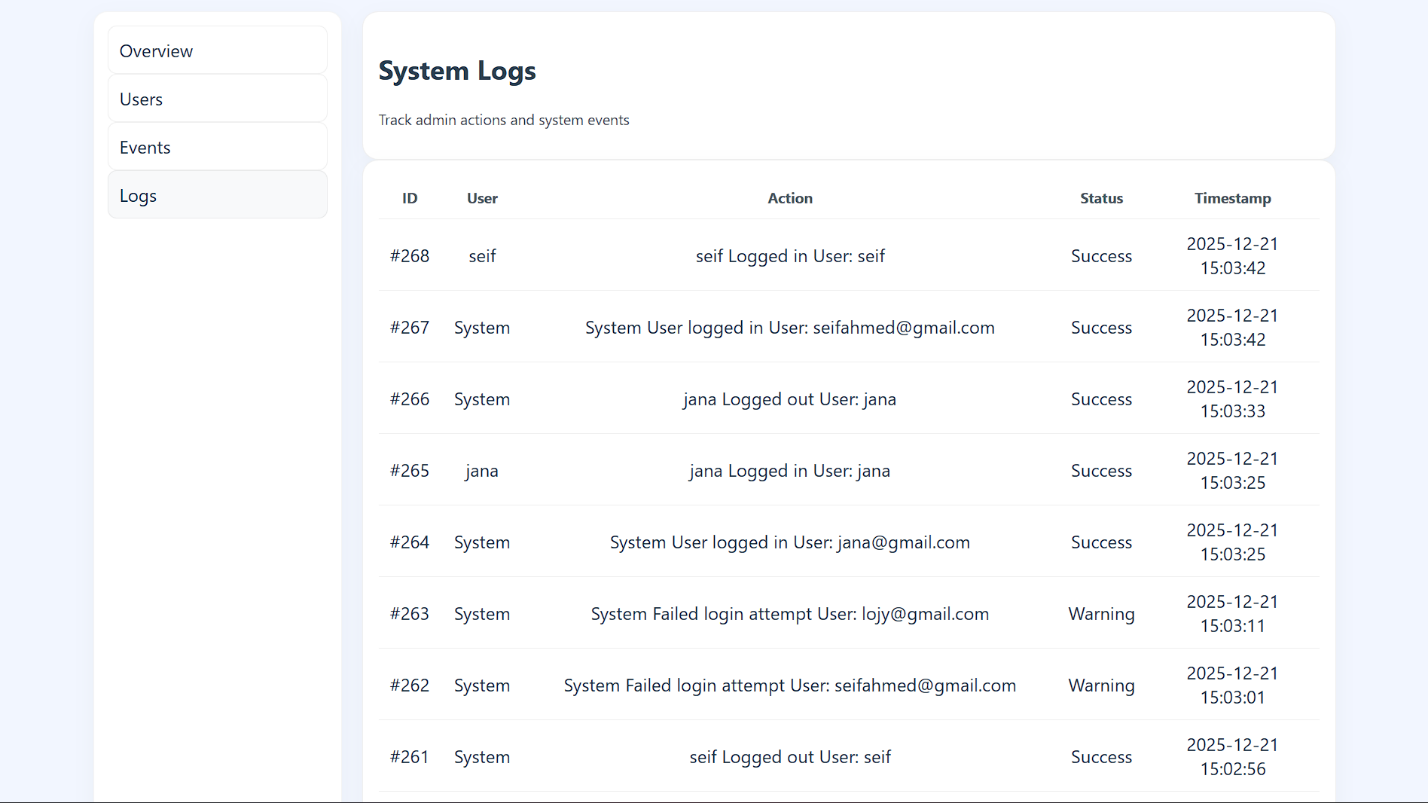
**Admin events**



**Add new event**



**Edit event**

**Admin logs**

## **5. Use case scenarios**

**Unregistered User (Guest)**

1. **Scenario 1.1: Browse Events (Home Page)**

* **Preconditions:** None.
* **Main flow:**  
  **Open Home → view upcoming events grid → use search bar to find a specific event → open event details → to proceed to checkout you must register or log in.**
* **Alternatives & error handling:**
  + **No matching events** → show "No matching events found."
  + **Event details fail to load (network/DB failure)** → show "Event details could not be loaded. Please try again."
* **Postconditions:** Guest is informed and may proceed to Register or Login before checkout.

1. **Scenario 1.2: Registration from Guest (Register Page)**

* **Preconditions:** Guest is on the Register page.
* **Main flow:**  
  **Click Register → enter email, password, and name → validate input → submit → email confirmation sent → redirect to Dashboard.**
* **Alternatives & error handling:**
  + **Email already exists** → show "Account already exists. Please login."
  + **Weak password** → show "Password must be at least 8 characters, include a capital letter and a number."
  + **Database errors** → show "Registration failed. Please try again later."
* **Postconditions:** Account is created and verified, or guest is instructed to resolve issues.

**Registered User (Customer)**

1. **Scenario 2.1: Browse & View Events (Login / Events Page)**

* **Preconditions:** User is authenticated via the Login page.
* **Main flow:**  
  **Login** **→ dashboard → open Events page → browse or search → open event details.**
* **Alternatives & error handling:**
  + **Wrong password** → show "Wrong email or password."
  + **Banned account** → show "This account is banned."
  + **Event no longer available** → show "Event not found ." and redirect to Events list.
* **Postconditions:** User successfully views available events and can procced to check out.

1. **Scenario 2.2: Purchase Tickets**

* **Preconditions:** User is authenticated and tickets are available.
* **Main flow:  
  Select event → choose tickets → proceed to Checkout → confirm payment →receive ticket → some info about ticket appears in Dashboard.**
* **Alternatives & error handling:**
  + **Payment fails** **→** show "Payment failed. Please try again."
  + **Wrong card number →** show " Payment failed. Please enter a valid number."
  + **Wrong expiry date→** show " Payment failed. expiry date must be upcoming."
* **Postconditions:** Ticket is confirmed and visible in dashboard with some info.

**Admin (System Administrator)**

1. **Scenario 3.1: Manage Events**

* **Preconditions:** Admin is authenticated on the Admin Dashboard.
* **Main flow:  
  Go to Events Management → add, edit, or delete events → validate input → save → log action.**
* **Alternatives & error handling:**
  + **Missing fields →** show "Please fill all required fields."
  + **Permission denied →** show "Access denied." and log attempt.
  + **Wrong Price →** show "price must be non-negative."
  + **Wrong date→** show " expiry date must be upcoming."
  + **Database error →** show "Error saving event. Please retry."
* **Postconditions**: Event is updated or added successfully, and the action is recorded in the log.

1. **Scenario 3.2: Manage Users**

* **Preconditions:** Admin authenticated.
* **Main flow:**  
  **Go to users Management → add, edit, or delete users → validate input → save → log action.**
* **Alternatives & error handling:**
  + **Missing fields →** show "Please fill all required fields."
  + **Wrong email format →**show"please enter a valid email"
  + **Database error →** show "Error saving event. Please retry."
* **Postconditions**: user is updated or added successfully, and the action is recorded in the log.

1. **Scenario 3.3: View Logs**

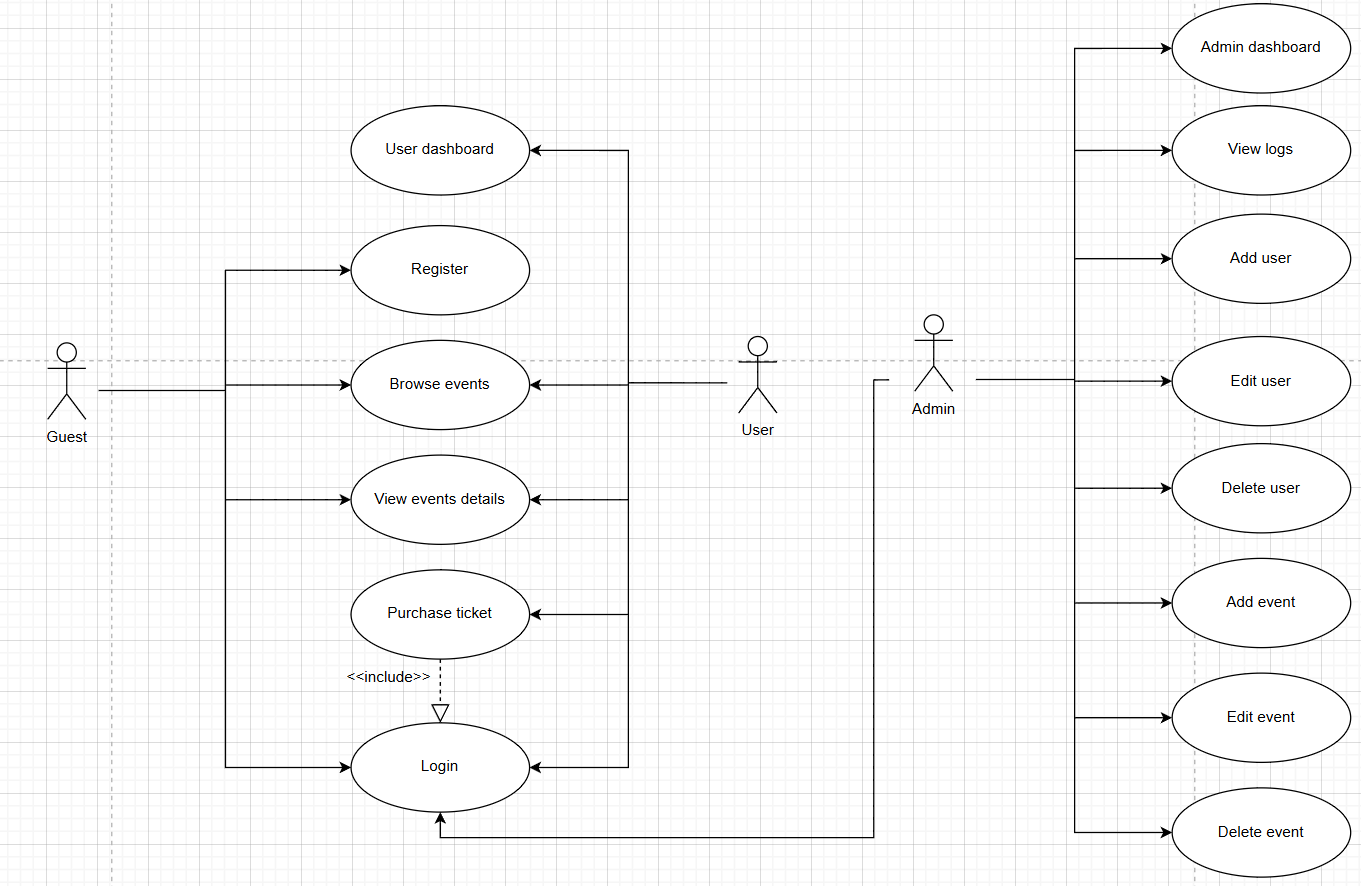
* **Preconditions:** Admin authenticated.
* **Main flow:**  
  **Logs → view → act.**
* **Alternatives & error handling:**
  + **loading** → show "unable to load logs."
* **Postconditions:** logs viewed.

**Cross-Cutting Error Cases**

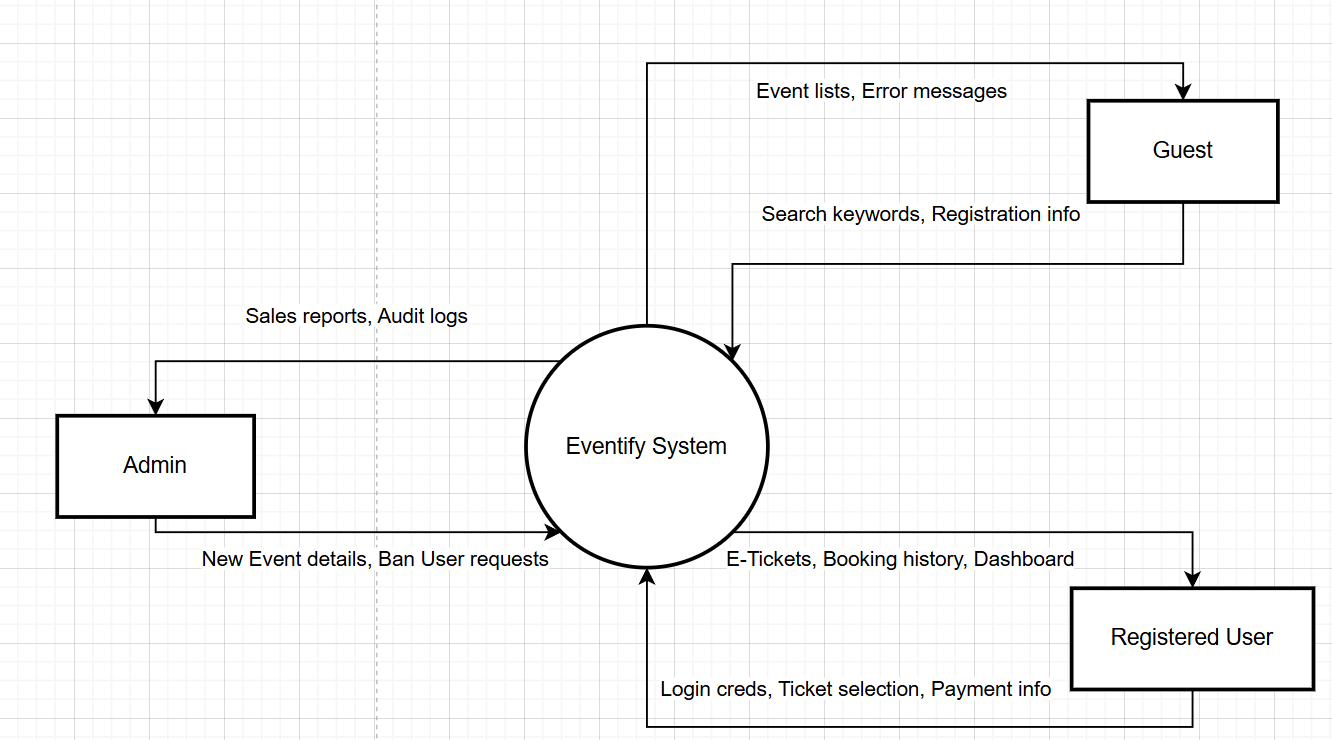
* **Authentication failures** → "Wrong email or password." / "This account is banned."
* **Authorization failures** → "Access denied."
* **Validation errors** → field-level messages (e.g., "Password must be at least 8 characters, contain a capital letter and a number.")
* **Database failures** → "System error. Please try again later."

## **6.Design Viewpoints**

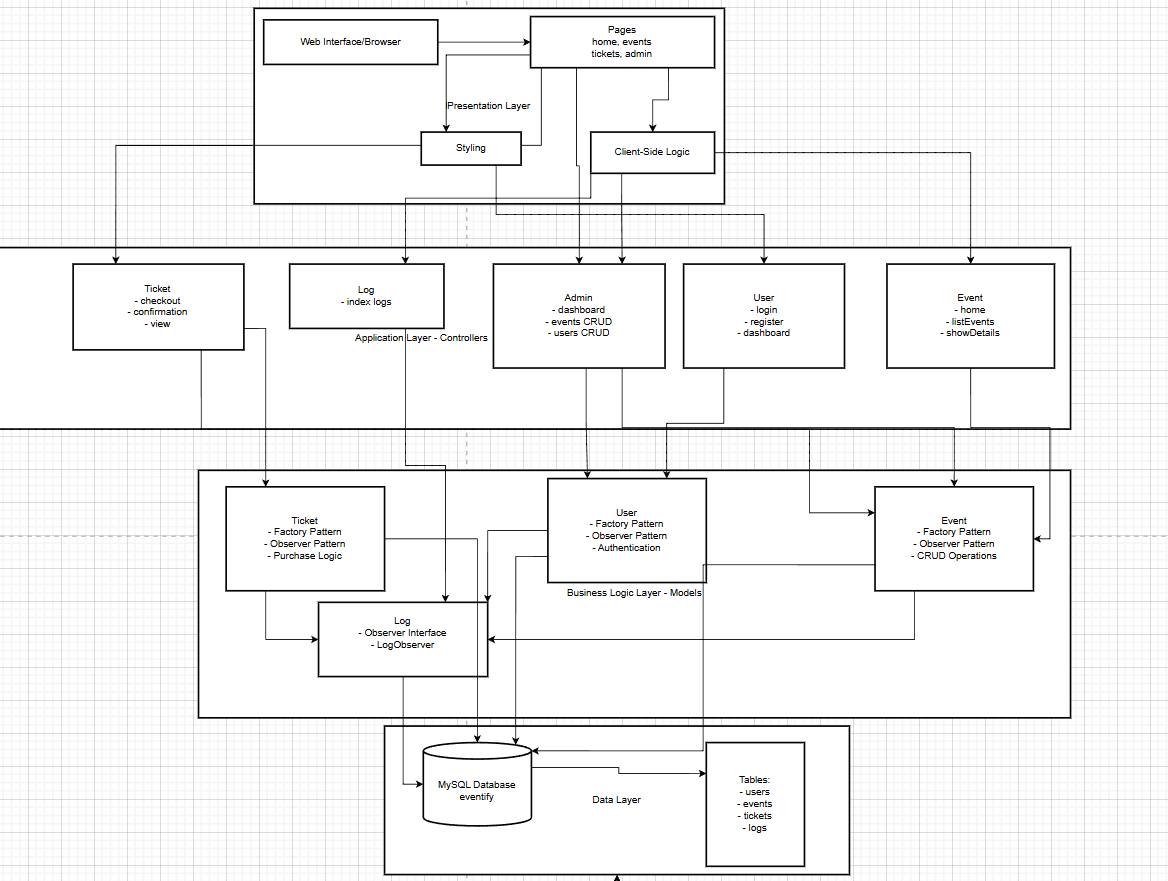
## Use case diagram



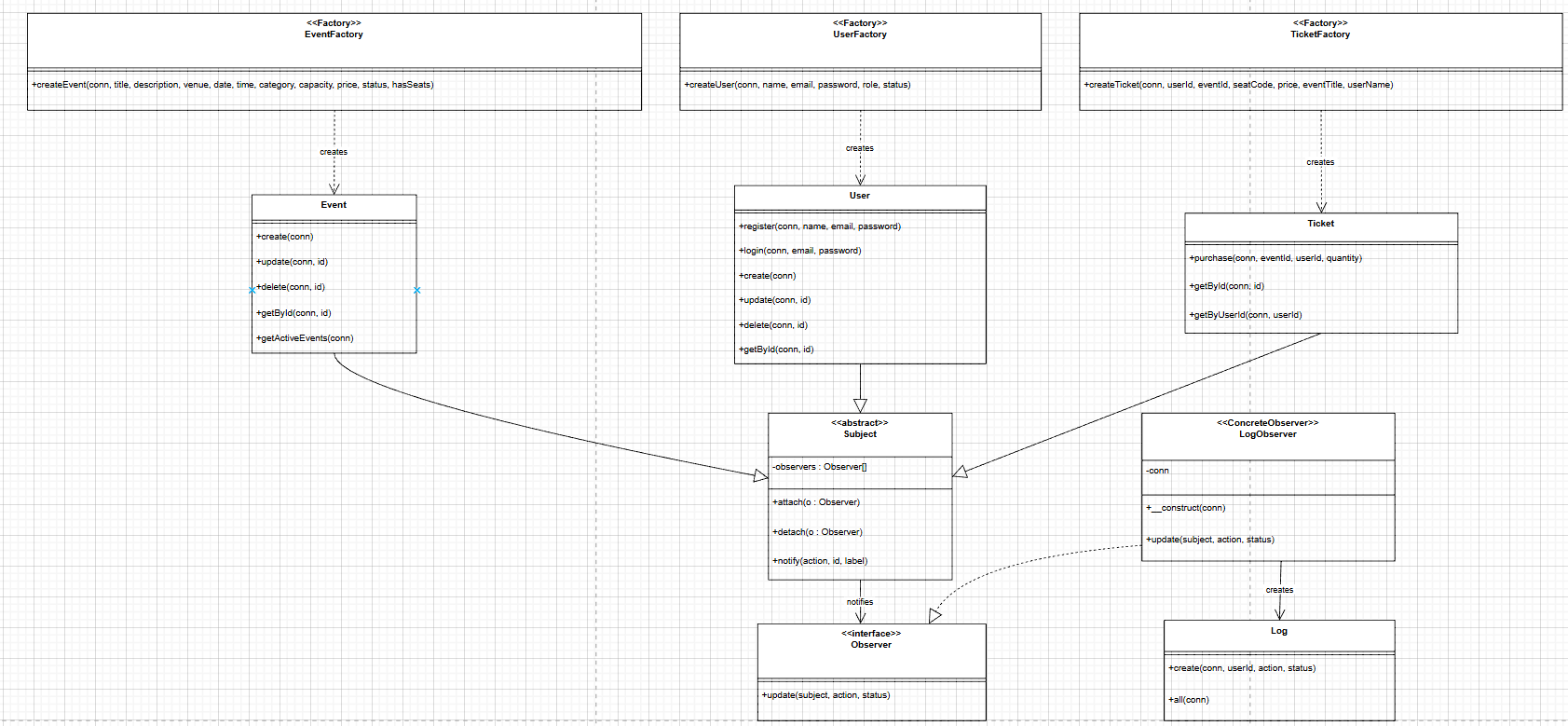
**Context diagram**

****

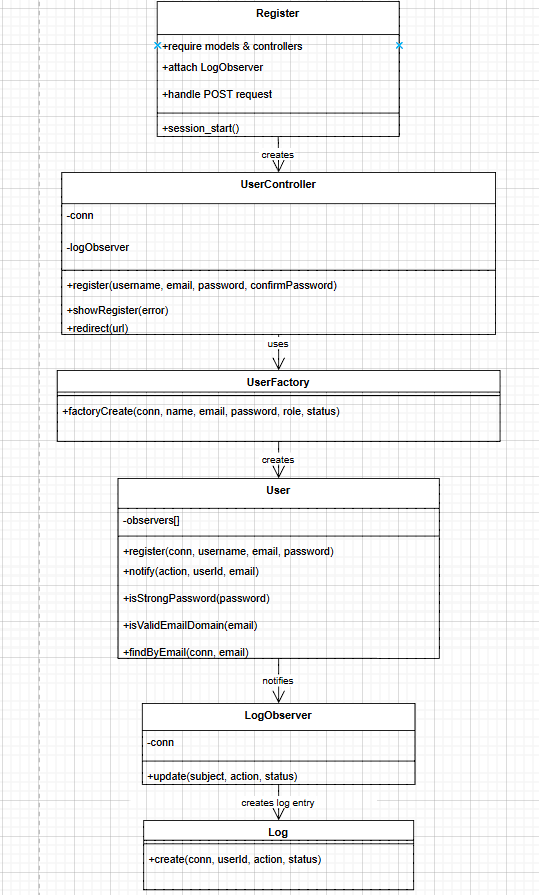
**Architecture diagram**

****

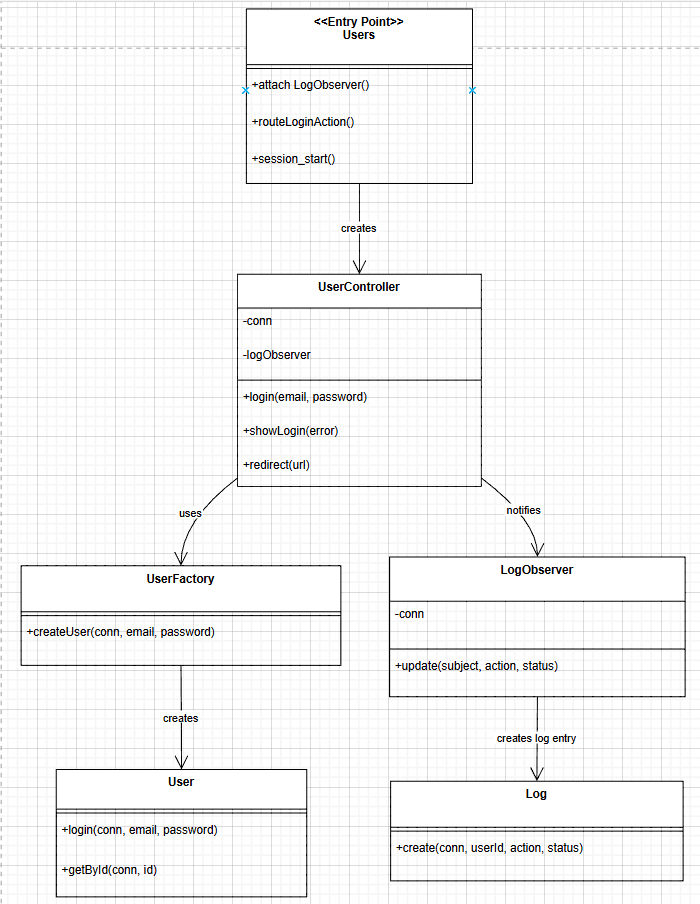
**Class diagram**

****

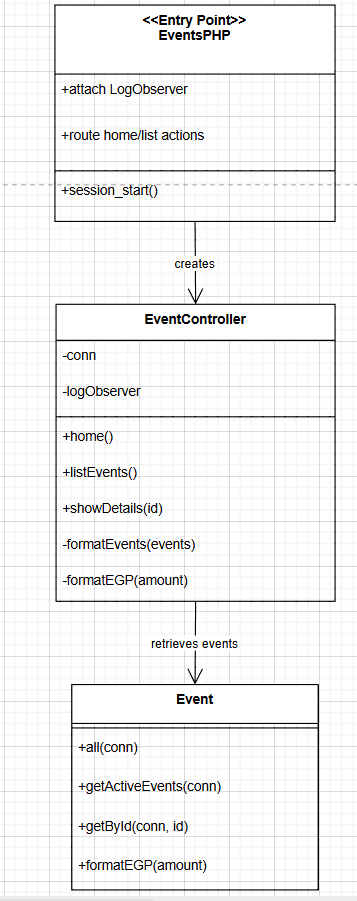
**Class diagram SCENARIO 1: USER REGISTRATION**

****

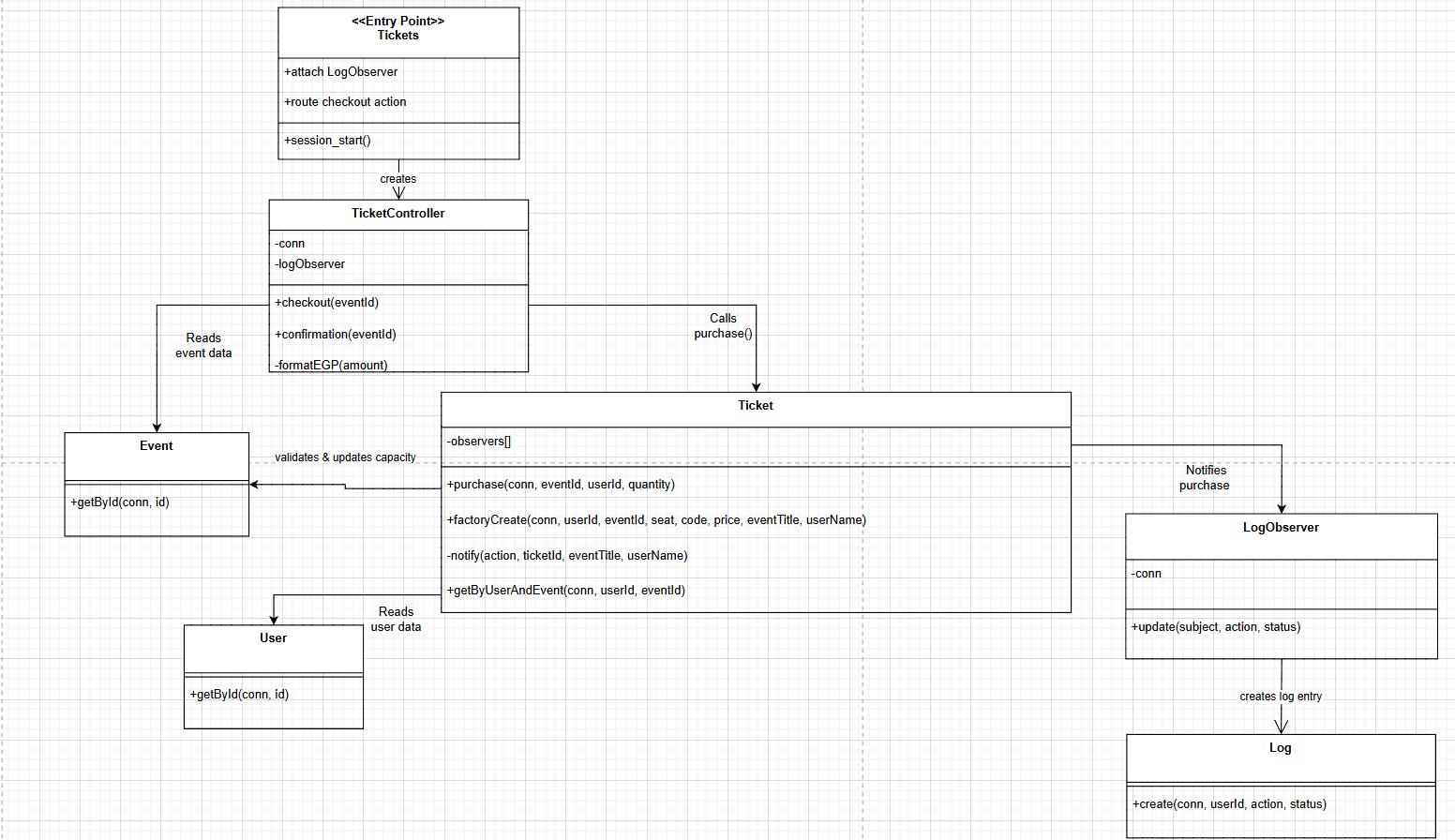
**SCENARIO 2: USER LOGIN**

****

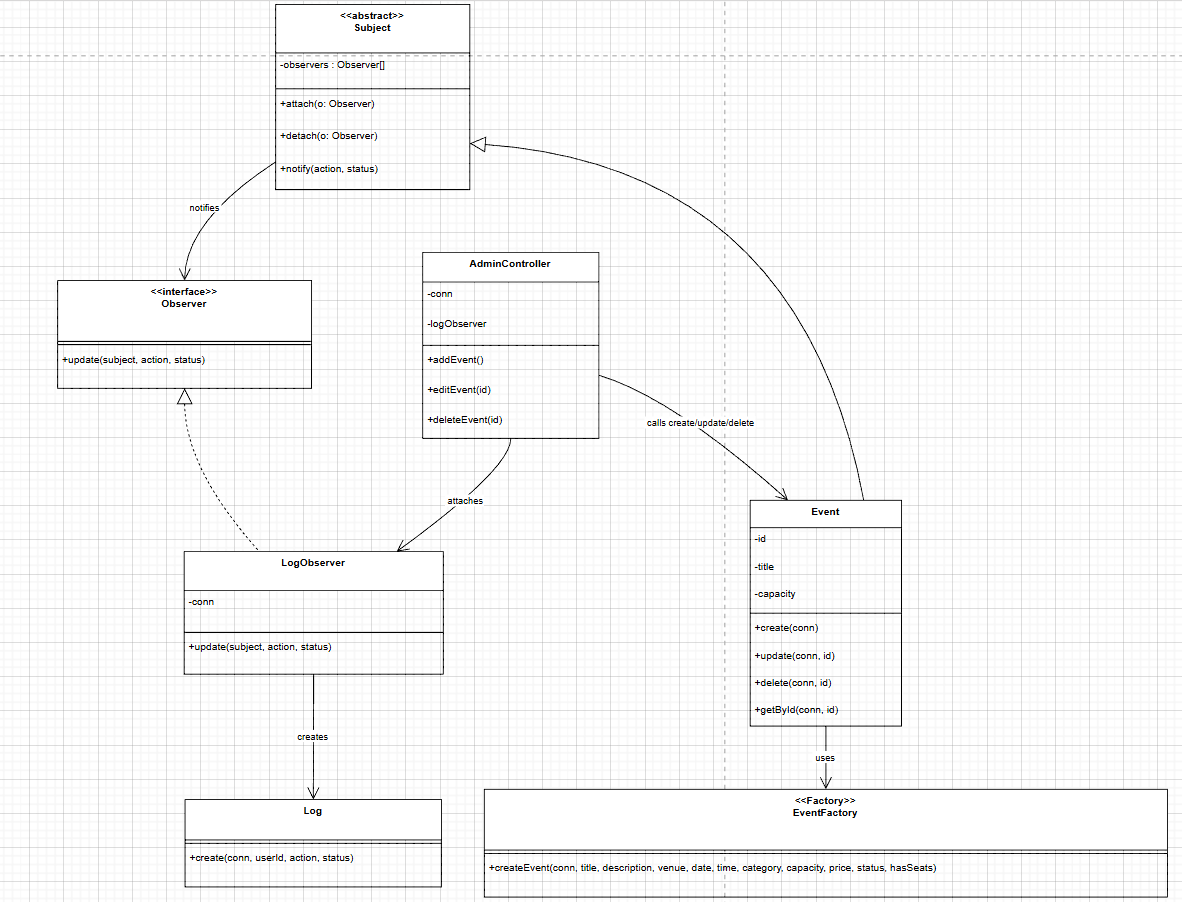
**SCENARIO 3: VIEW EVENTS**

****

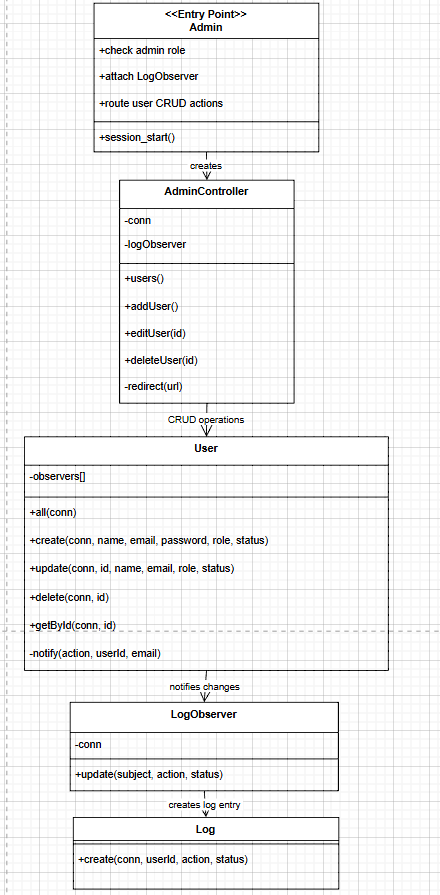
**SCENARIO 4: PURCHASE TICKET**

****

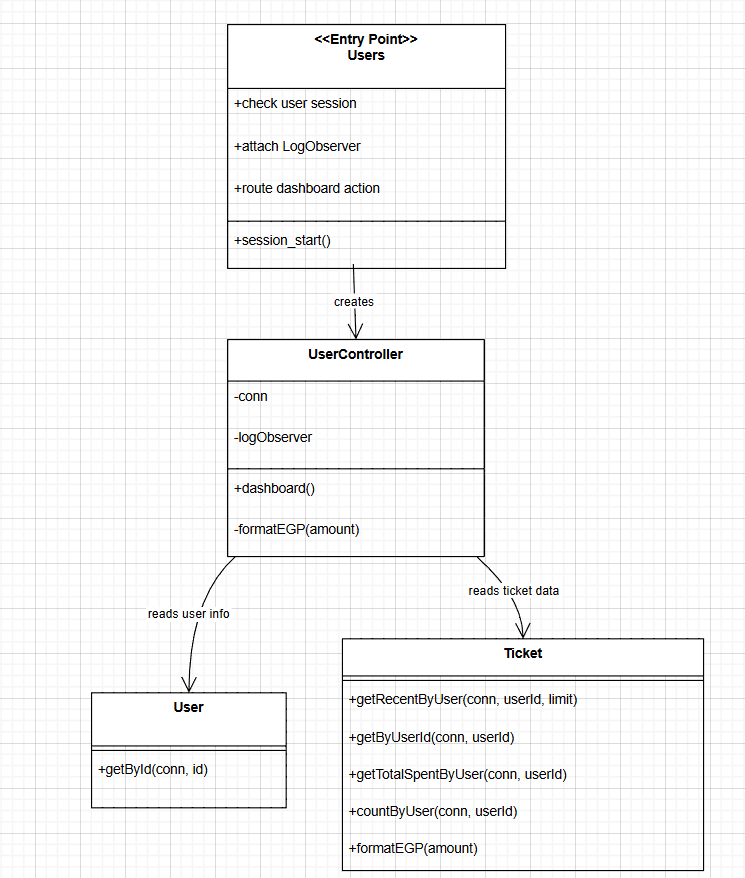
**SCENARIO 5: ADMIN CREATE EVENT**

****

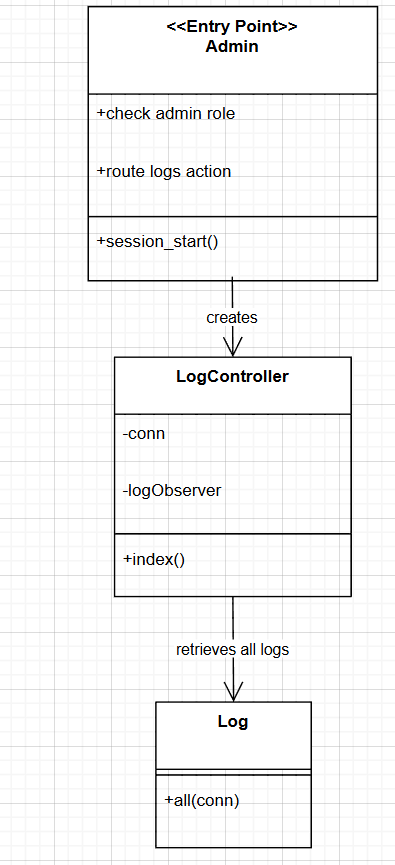
**SCENARIO 6: ADMIN MANAGE USERS**

****

**SCENARIO 7: USER DASHBOARD**

****

**SCENARIO 8: VIEW LOGS (ADMIN)**

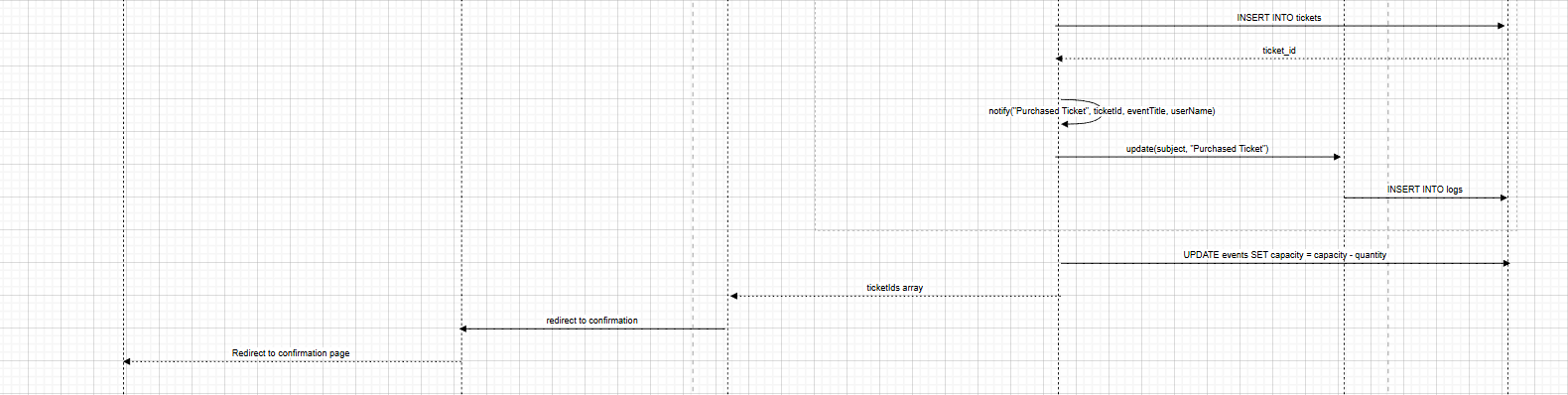
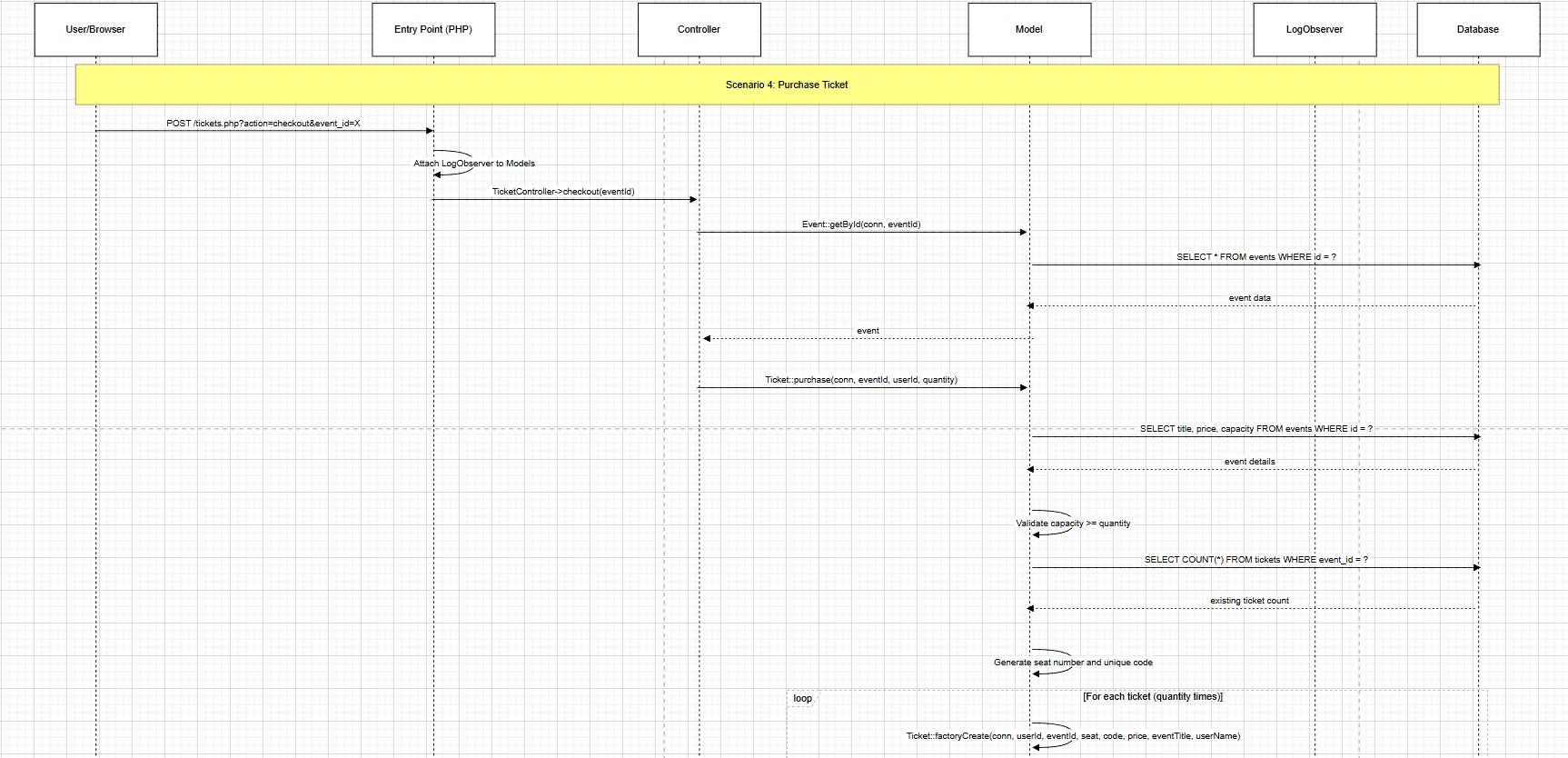
****

**Sequence diagrams**

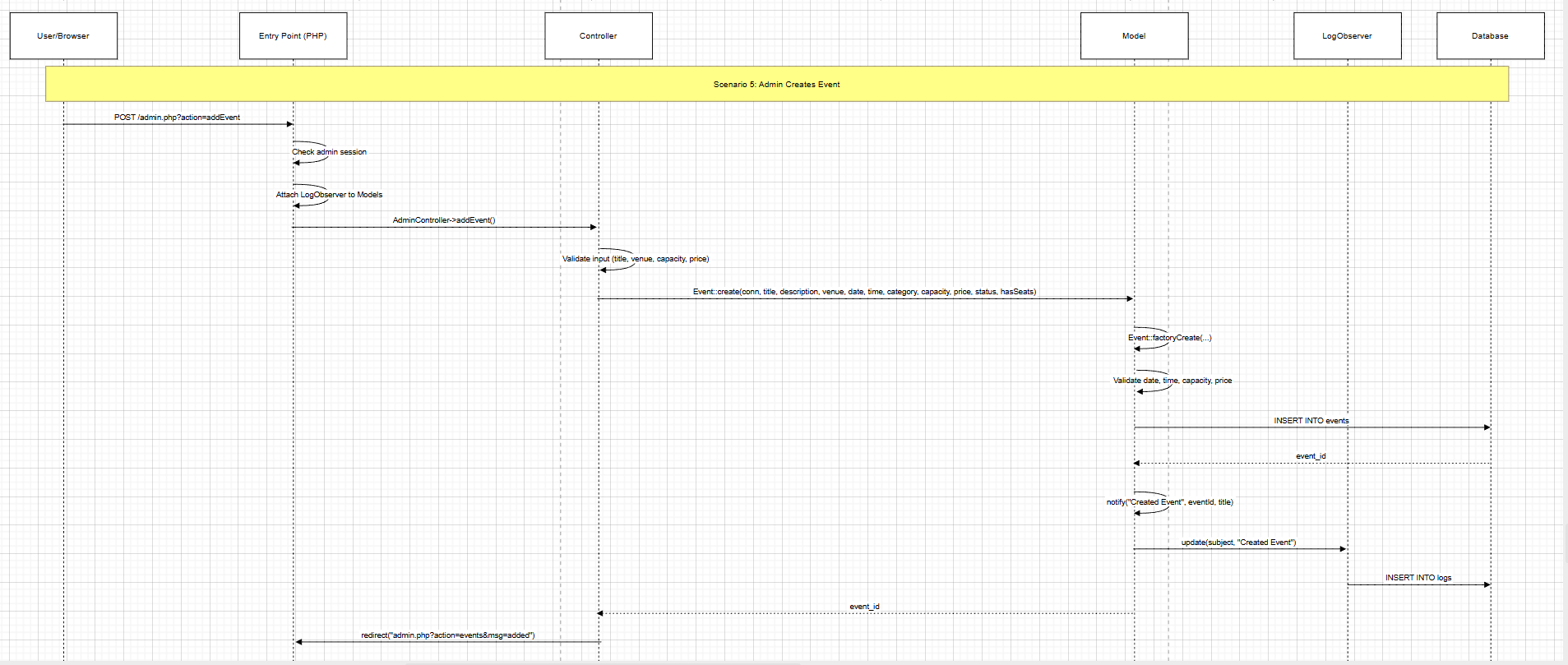
**Scenario 1: User Registration**

**Scenario 2: User Login**

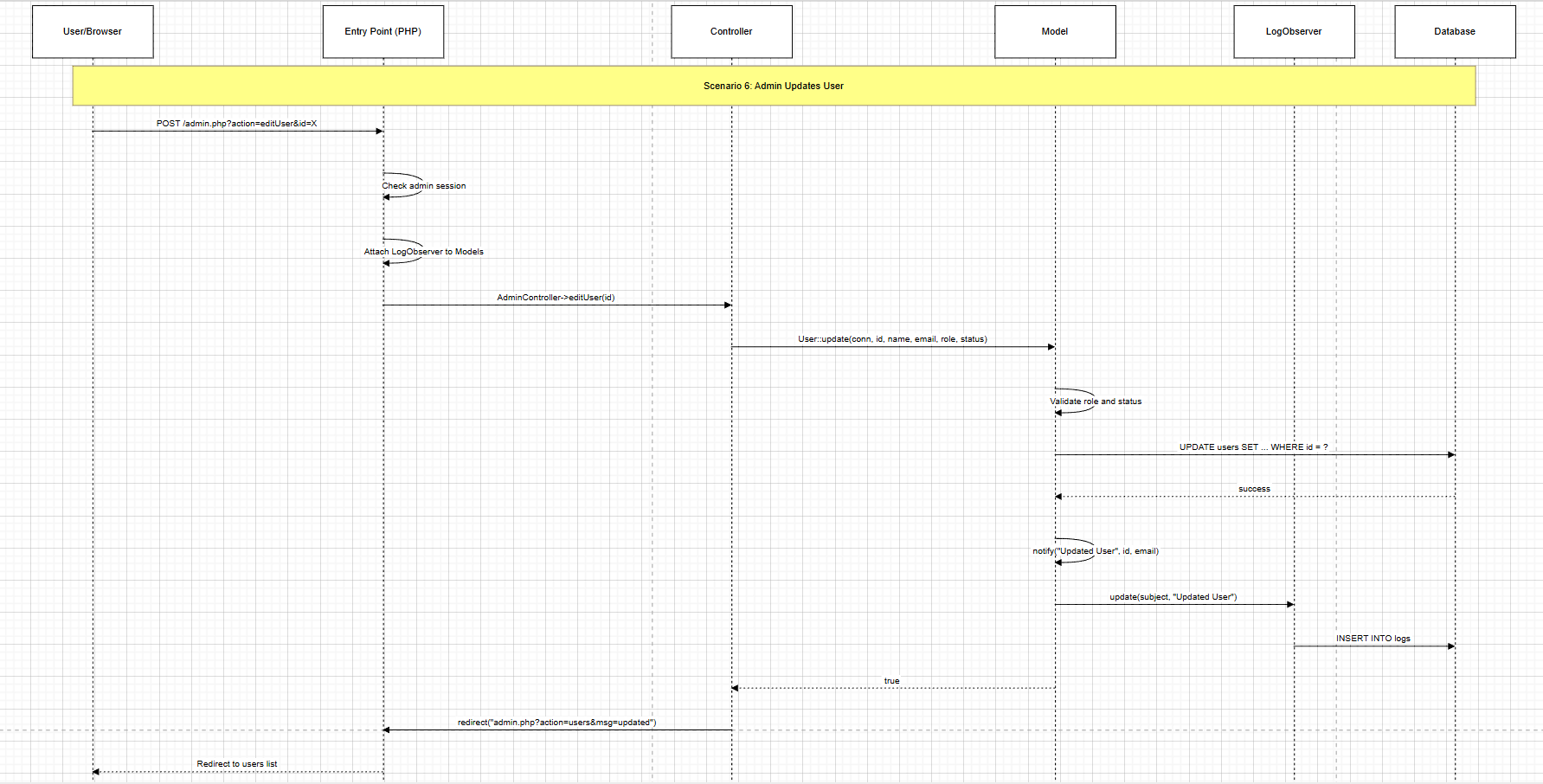
**Scenario 3: View Events (Home)**

**Scenario 4: Purchase Ticket**

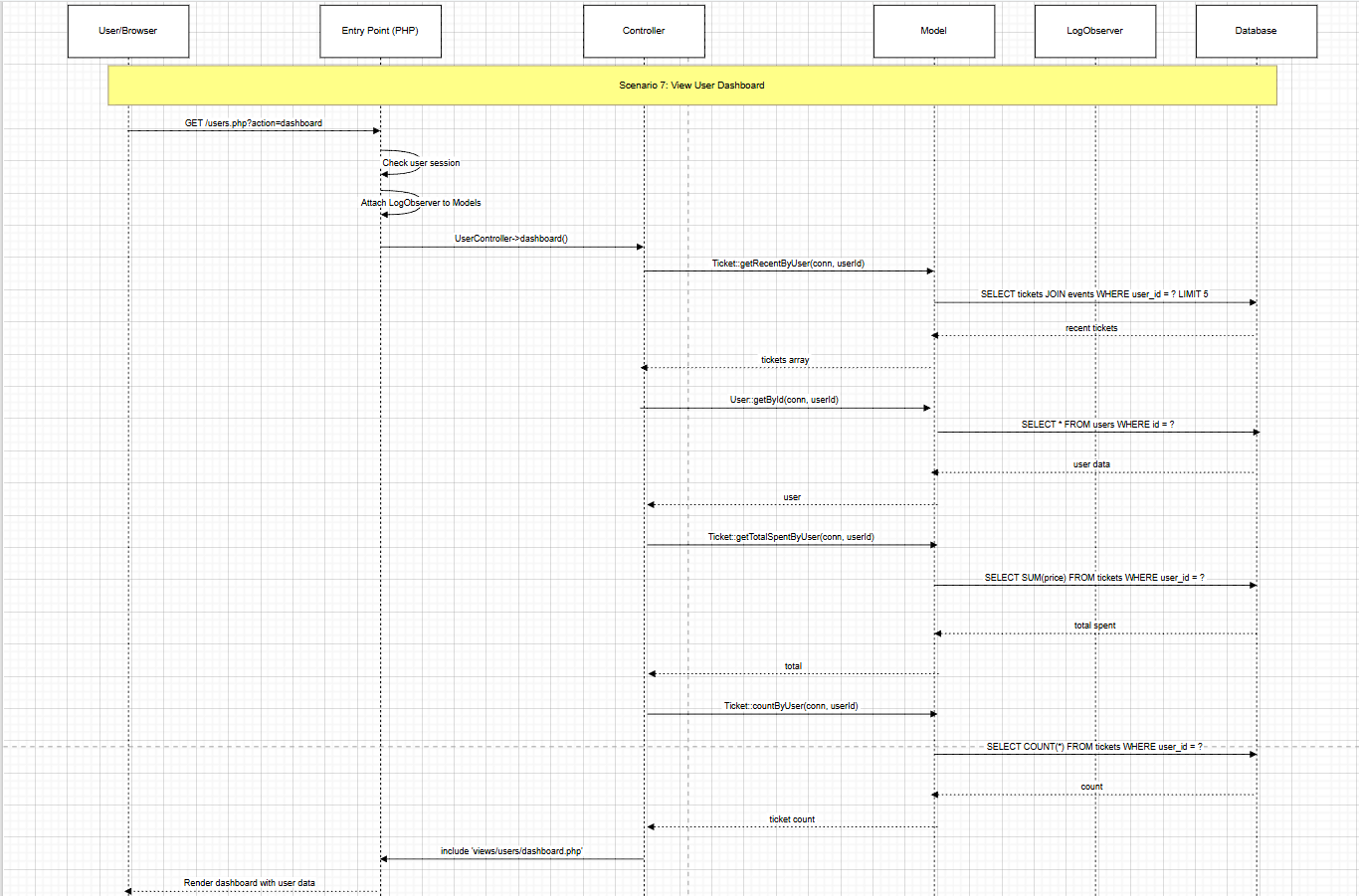
**Scenario 5: Admin Creates Event**

****

Scenario 6: Admin Updates User

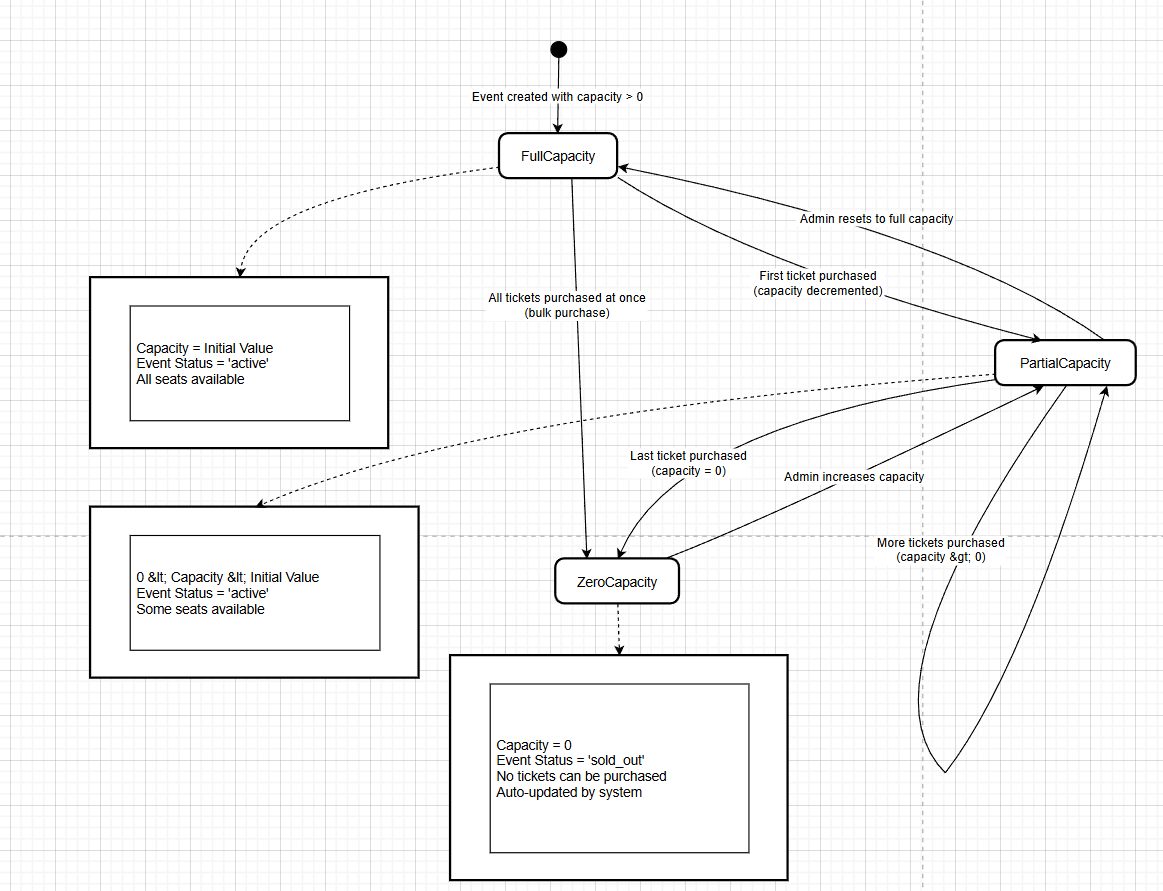


Scenario 7: View User Dashboard

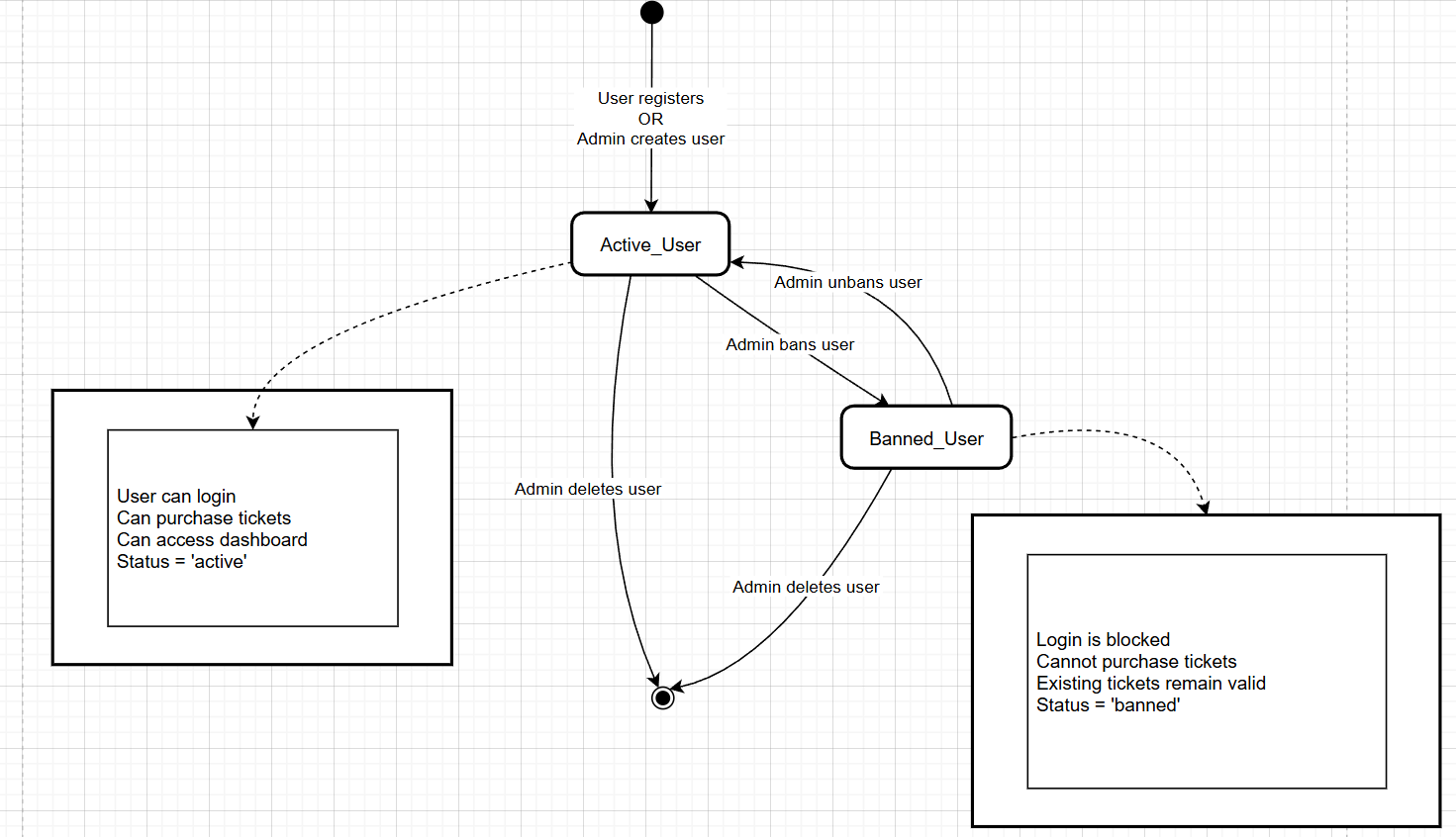


**State diagrams**

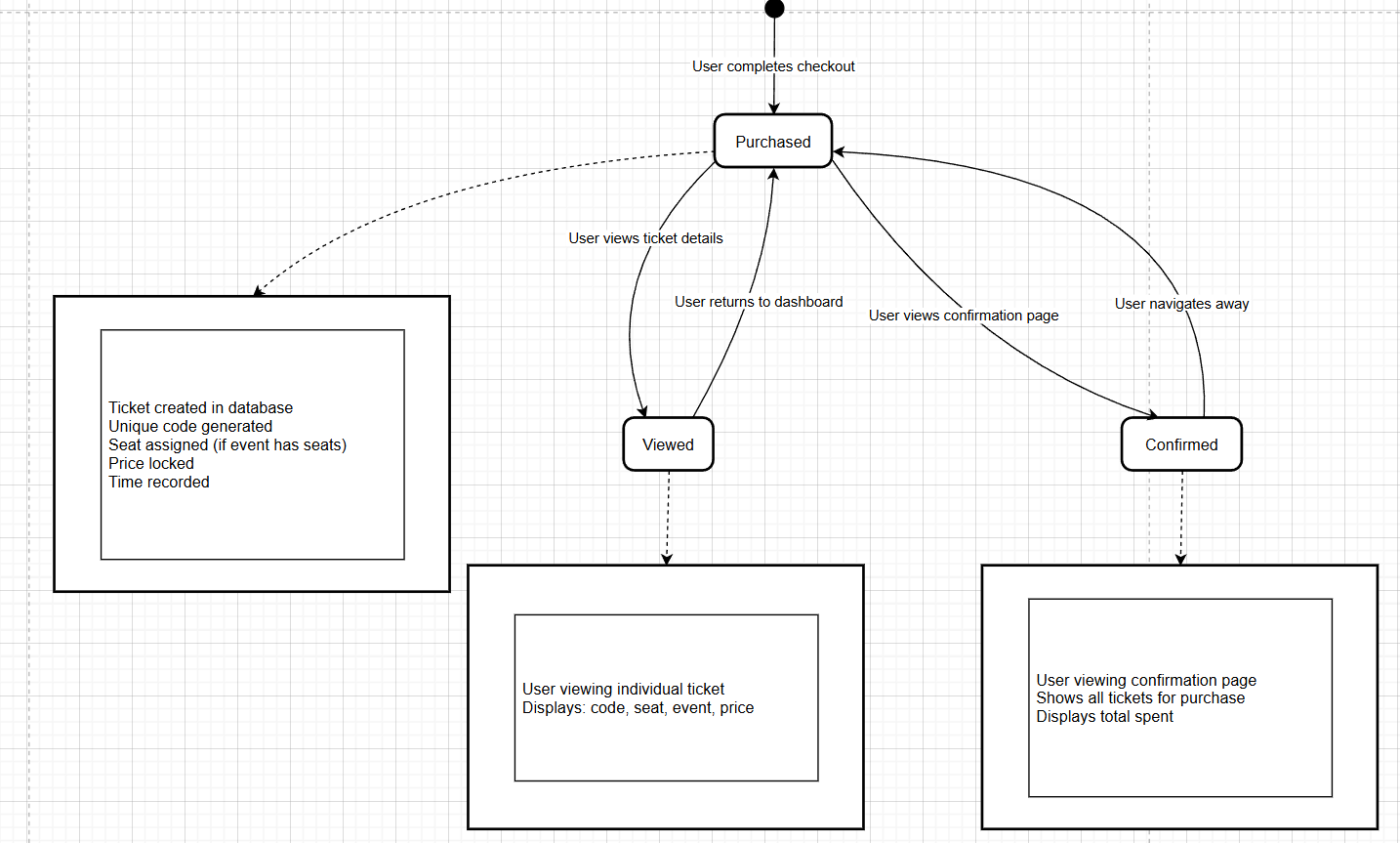
**EVENT CAPACITY TRACKING**

****

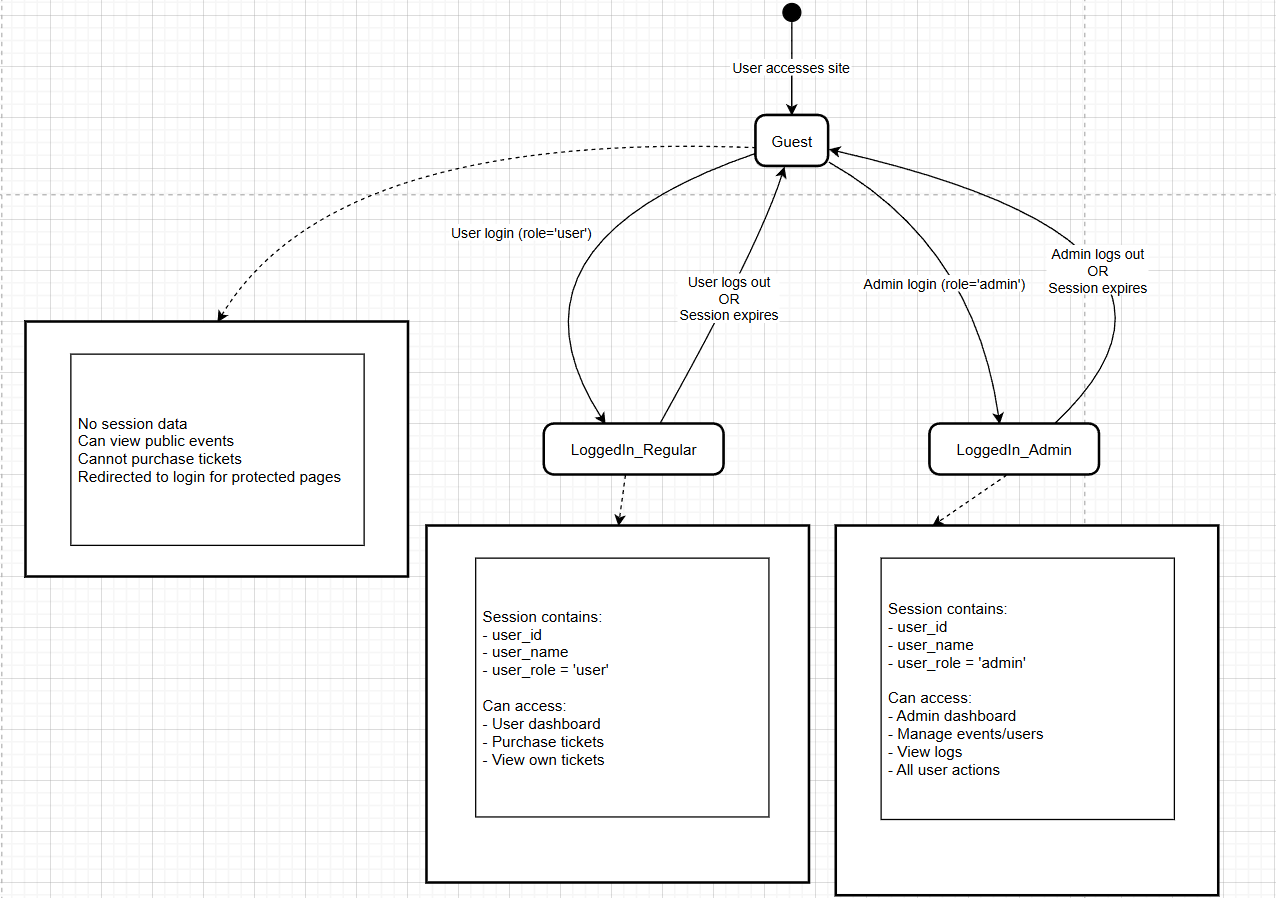
**USER ACCOUNT STATUS**

****

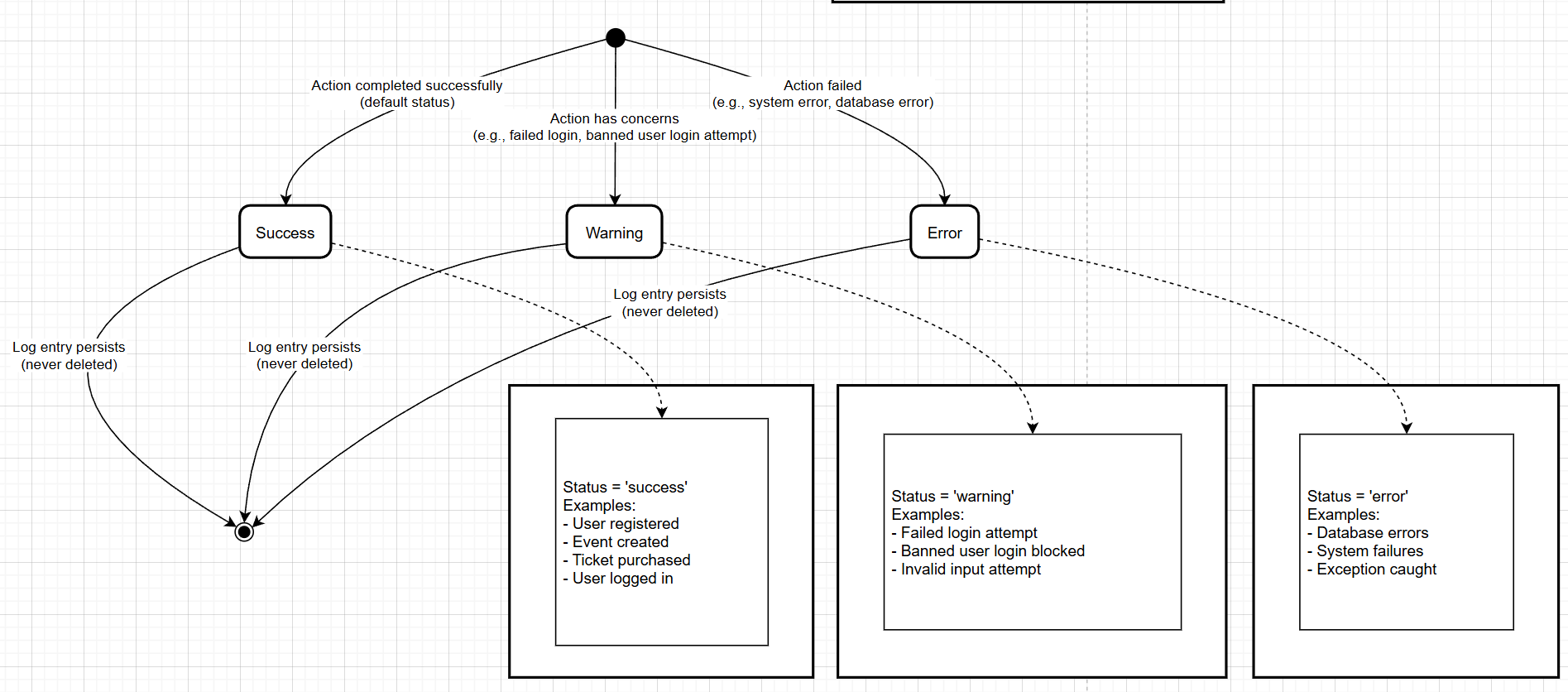
**TICKET LIFECYCLE**

****

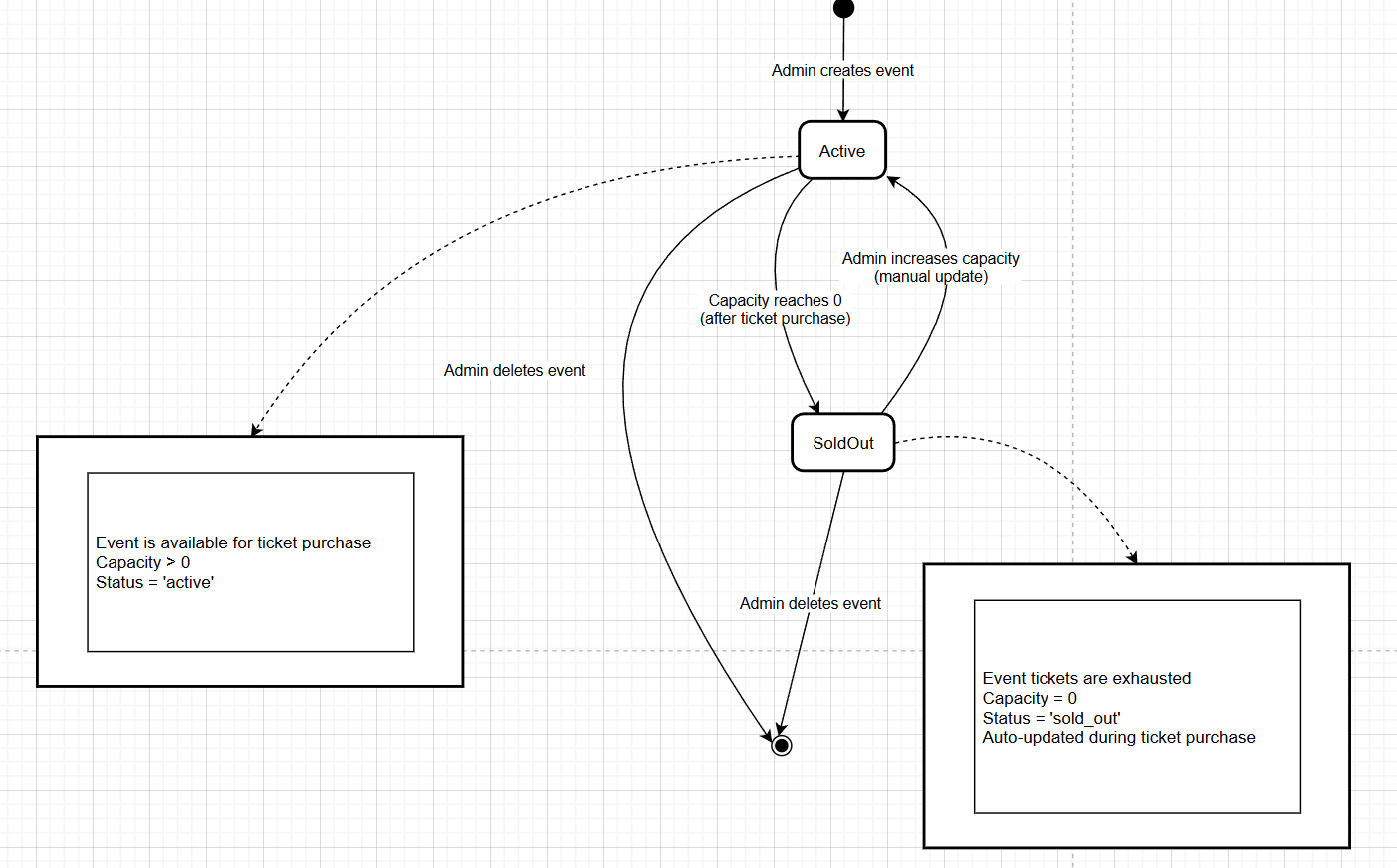
**USER SESSION STATE**

****

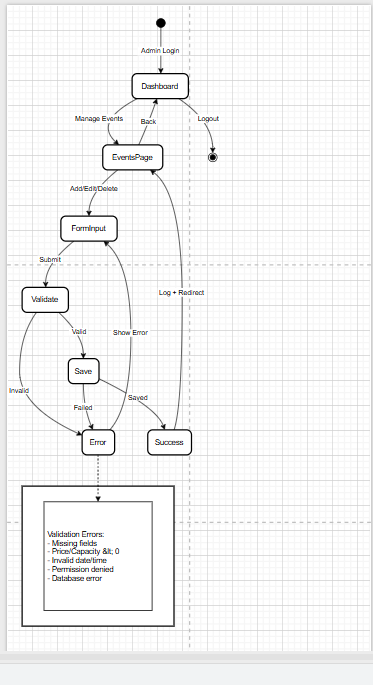
**LOG ENTRY STATUS**

****

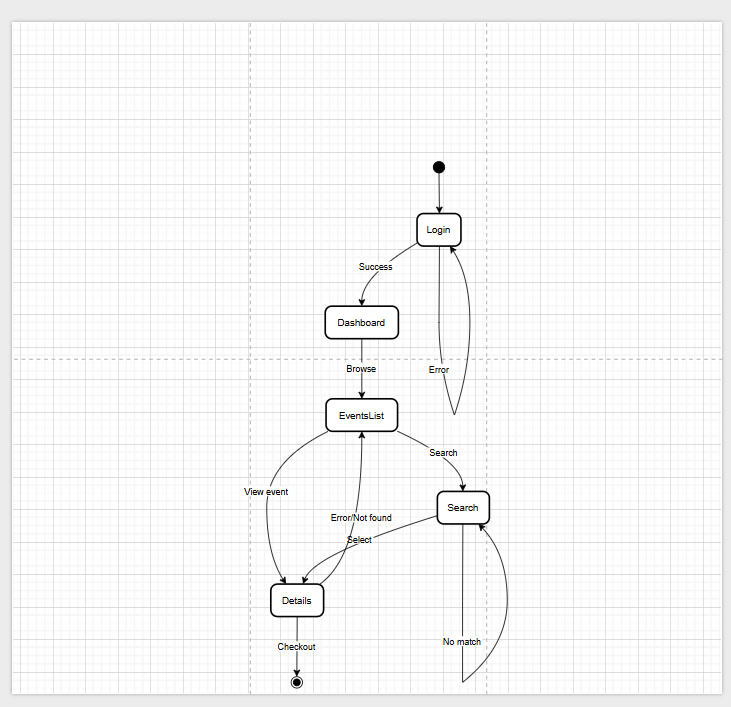
**EVENT STATUS**

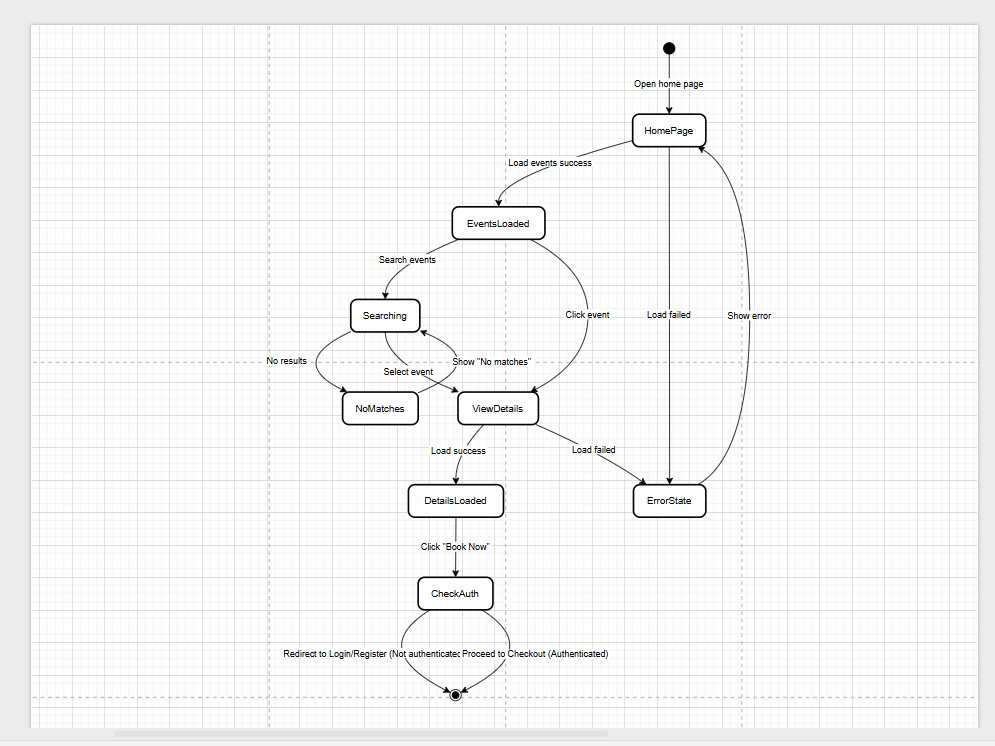
****

**Admin**

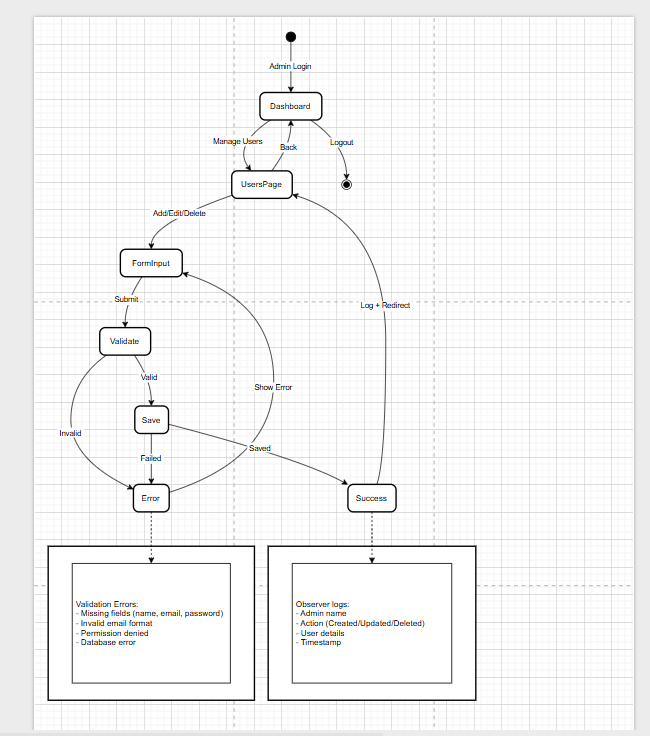


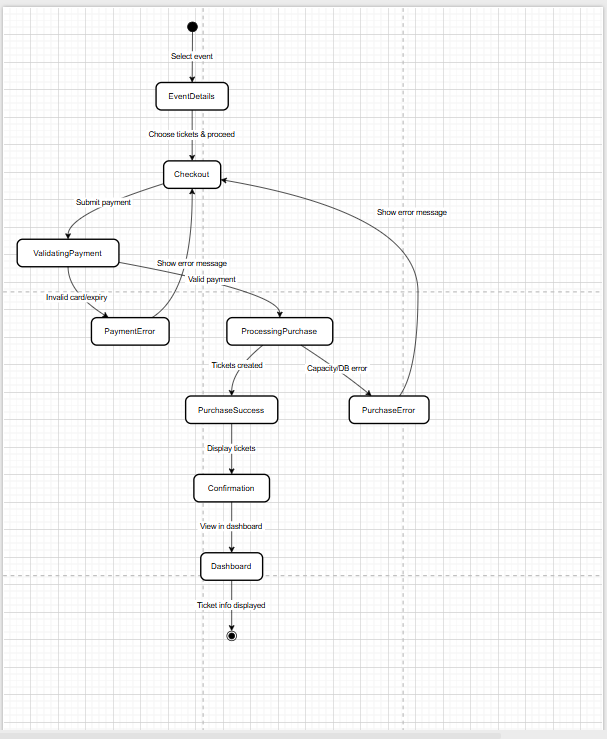
Customer



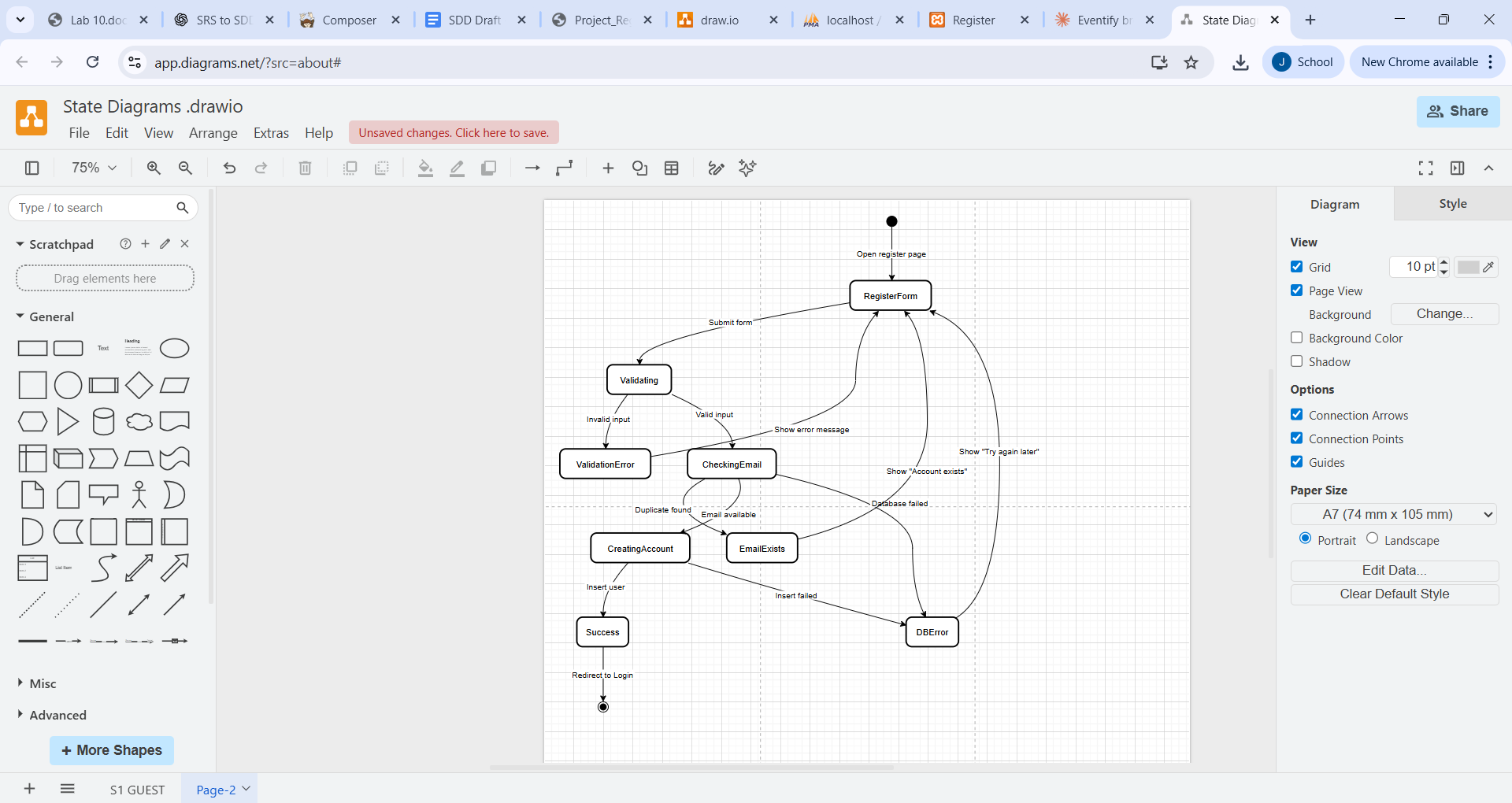
Guest

Admin

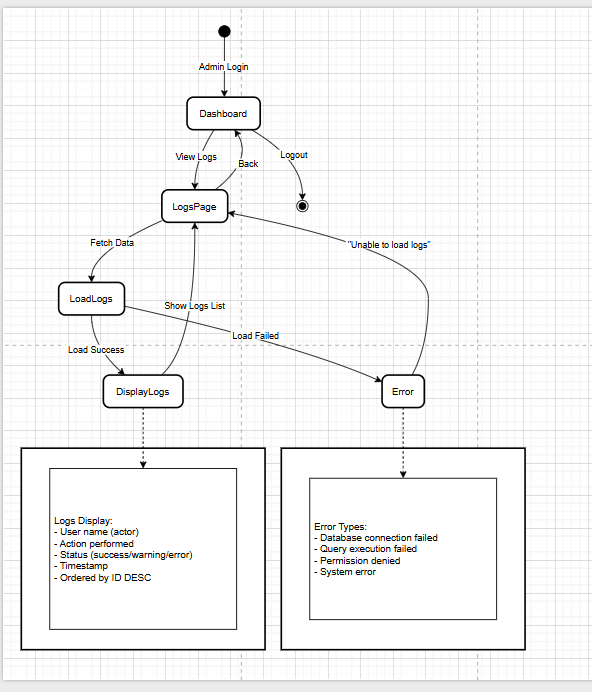


**user**

**Guest**

****

Admin



|  |  |
| --- | --- |
| Team member | Tasks |
| Seif Ahmed | Coding: Controllers  models  Views  entry files  JavaScript  styling  DB scheme  Document:  Diagrams: class, context, architecture, use case and ER diagram  DB design  User interface design |
| Jana Hassan | Coding:  Automated unit testing  Document:  Diagrams: state and sequence  Rest of the document |