**Software Design Specifications**

**for**

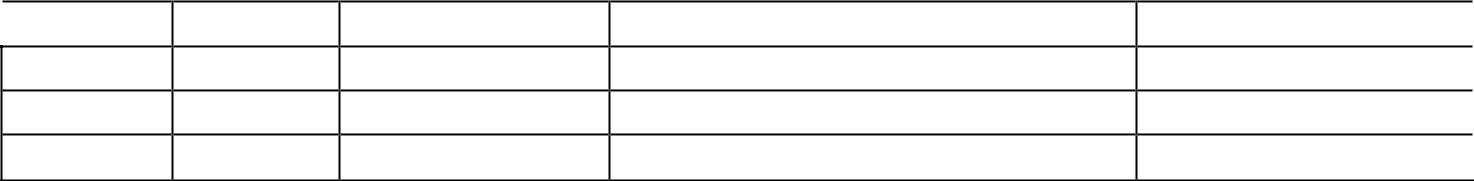
**Event Management System**

**Document Information**

|  |  |
| --- | --- |
| **Title: Event Management**  **system** |  |
| **ProjectManager: Narthkannai . K** | **Document Version No: 1** |
|  | **Document Version Date: 09-04-25** |
| **Prepared By:**  **Sachethan Pilli**  **Sameera Nallamothu**  **Chandana Priya**  **Susmitha Manthena**  **Sanjana Singam**  **Chaitanya Goud** | **Preparation Date: 09-04-25** |
|  |  |

**Version History**

**Ver. No.  Ver. Date  Revised By  Description  Filename**

****

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **1 INTRODUCTION ............................................................................................................................................................** | | **4** |
| 1.1 | PURPOSE ...................................................................................................................................................................... | 4 |
| 1.2 | SCOPE .......................................................................................................................................................................... | 4 |
| 1.3 | DEFINITIONS, ACRONYMS, AND ABBREVIATIONS ........................................................................................................ | 4 |
| 1.4 | REFERENCES ................................................................................................................................................................ | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **USE CASE VIEW ............................................................................................................................................................** | | **4** |
|  | 2.1 | USE CASE ..................................................................................................................................................................... | 4 |
| **3** | **DESIGN OVERVIEW .....................................................................................................................................................** | | **4** |
|  | 3.1 | DESIGN GOALS AND CONSTRAINTS ............................................................................................................................. | 5 |
|  | 3.2 | DESIGN ASSUMPTIONS ................................................................................................................................................. | 5 |
|  | 3.3 | SIGNIFICANT DESIGN PACKAGES ................................................................................................................................. | 5 |
|  | 3.4 | DEPENDENT EXTERNAL INTERFACES ........................................................................................................................... | 5 |
|  | 3.5 | IMPLEMENTED APPLICATION EXTERNAL INTERFACES ................................................................................................. | 5 |
| **4** | **LOGICAL VIEW .............................................................................................................................................................** | | **5** |
|  | 4.1 | DESIGN MODEL ............................................................................................................................................................ | 6 |
|  | 4.2 | USE CASE REALIZATION .............................................................................................................................................. | 6 |
| **5** | **DATA VIEW ....................................................................................................................................................................** | | **6** |
|  | 5.1 | DOMAIN MODEL .......................................................................................................................................................... | 6 |
|  | 5.2 | D ATA MODEL (PERSISTENT DATA VIEW)...................................................................................................................... | 6 |
|  | *5.2.1 Data Dictionary....................................................................................................................................................* | | *6* |
| **6** | **EXCEPTION HANDLING .............................................................................................................................................** | | **6** |
| **7** | **CONFIGURABLE PARAMETERS ..............................................................................................................................** | | **6** |
| **8** | **QUALITY OF SERVICE ................................................................................................................................................** | | **7** |
|  | 8.1 | AVAILABILITY.............................................................................................................................................................. | 7 |
|  | 8.2 | SECURITY AND AUTHORIZATION ................................................................................................................................. | 7 |
|  | 8.3 | LOAD AND PERFORMANCE IMPLICATIONS ................................................................................................................... | 7 |
|  | 8.4 | MONITORING AND CONTROL ....................................................................................................................................... | 7 |

**1** **Introduction**

The Software Design Specifications (SDS) document outlines the architectural and design approach of the College Event Management Website intended for use by students, faculty, and event organizers. This document presents a structured representation of the system components, interfaces, data flow, and design principles that will guide the development of the application.

**1.1** **Purpose**

The purpose of this document is to define and describe the software design for the

College Event Management Website. It serves as a technical blueprint that guides in

implementing the required features and modules of the system. This SDS is part of the

overall project documentation and aligns with the software requirements specification

(SRS).

This document is intended for:

**Team Members** – to understand the architecture and design needed to build and maintain the application.

**Professors** – to review and evaluate the system’s internal structure, logic, and design decisions.

**1.2** **Scope**

This document applies to the design of the College Event Management Website and all its software components. It focuses on the design architecture, module descriptions, database schema, and interface structures. The website includes features such as:

User registration and authentication

Event creation and management

Registration tracking

Expense Tracking

All front-end, back-end, and database components are covered within this design specification.

**1.3** **Definitions, Acronyms, and Abbreviations**

SDS Software Design Specifications

SRS Software Requirements Specifications

UI User Interface

UX User Experience

API Application Programming Interface

DB Database

HTML Hypertext Markup Language

CSS Cascading Style Sheets

JS JavaScript

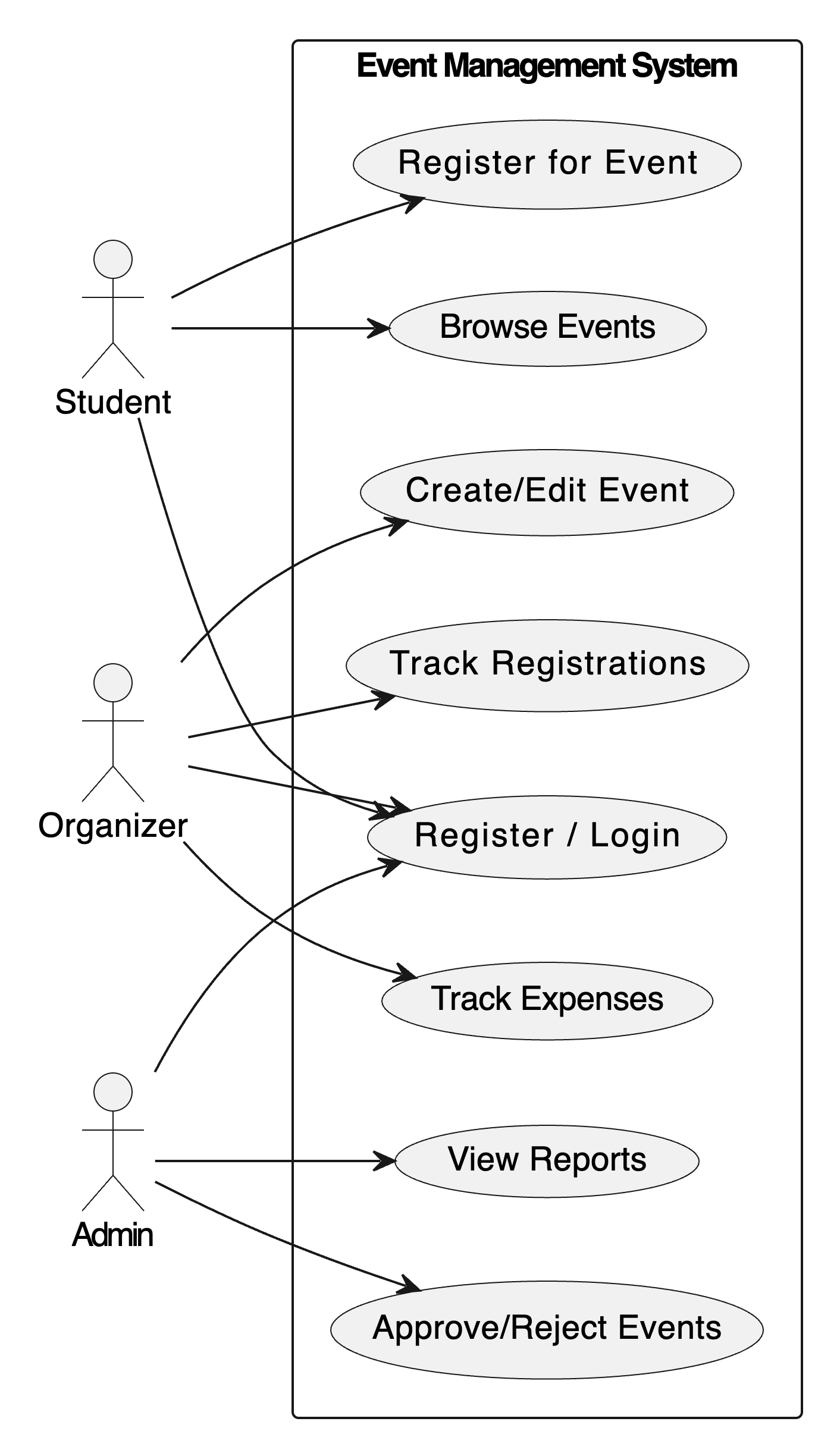
SQL Structured Query Language

**1.4** **References**

•⁠ ⁠IEEE Software Design Specification Template

• UML Diagrams: Use Case, Class, Sequence

**2** **Use Case View**

This section outlines the primary use cases relevant to the software design of the College Event Management Website. These use cases represent central functionality and cover major interactions among the user roles—**Student**, **Organizer**, and **Admin**. Each use case involves multiple design components including the front-end, back-end, and database layers. The design also considers validation, state transitions, and feedback to the users.  
  


**2.1** **Use Case**

Below are the brief descriptions and usage steps for key use cases.

**2.1.1 Use Case: Register/Login**

**Description:**

Allows users (students, organizers, and admins) to create an account or authenticate themselves to access respective dashboards.

**Steps:**

1.User enters university email and password.

2.System validates credentials.

3.If valid, user is redirected to the corresponding dashboard.

4.If invalid, an error message is shown.

**2.1.2 Use Case: Create/Edit Event**

**Actor:** Organizer

**Description:** Organizer can create new events or update existing event details such as name, venue, date, description, and budget

**Steps:**

1.Organizer logs in and navigates to “Create New Event”.

2.Fills in event form and submits.

3.Event is stored in the database with status = pending.

4.Admin can later approve or reject.

**2.1.3 Use Case: Approve/Reject Event**

**Actor:** Admin

**Description:** Admin reviews pending event submissions and approves or rejects them.

**Steps:**

1.Admin logs in and views pending events.

2.Clicks “Approve” or “Reject”.

3.System updates the event status in the database.

4.Organizer gets notified.

**2.1.4 Use Case: Register for Event**

**Actor:** Student

**Description:** Students can view approved and ongoing events and register for them.

**Steps:**

1.Student logs in and views events.

2.Selects an event and clicks “Register”.

3.Registration details are stored.

4.Confirmation is shown to the student.

**2.1.5 Use Case: Track Expenses**

**Actor:** Organizer

**Description:** Organizer can add actual expenses against their planned event budget.

**Steps:**

1.Organizer opens the event expense tracker.

2.Inputs category-wise expenses.

3.System calculates total spent and remaining budget.

**2.1.6 Use Case: View Reports**

**Actor:** Admin

**Description:**Admin can view registration reports, event summaries, and feedback.

**Steps:**

1.Admin navigates to the Reports section.

2.Selects filters (event, date, etc.).

3.System displays or downloads report data.

1. **Design Overview**

This section outlines the overall architecture and major design considerations for the

College Event Management Website. The system is designed using a modular

architecture where different functional areas such as authentication, event

management, and user interaction are separated for maintainability and scalability.

**3.1** **Design Goals and Constraints**

**Design Goals:**

1.**User-Centric Experience:**

Develop a responsive and user-friendly web portal that ensures a smooth and intuitive experience for students, organizers, and administrators.

2.**Secure Authentication and Role Management:**

Implement a secure login system with role-based access control to distinguish between students, organizers, and admin functionalities.

3.**Efficient Event Lifecycle Management:**

Enable easy and efficient creation, approval, registration, tracking, and management of college events through a streamlined interface.

**Design Constraints:**

1.**Cross-Device Accessibility:**

The system must function seamlessly on both desktop and mobile browsers to support varied user access points.

2.**Time and Resource Limitations:** The project is constrained by a limited development timeline and relies on a small team of student developers with part-time availability.

**3.2** **Design Assumptions**

* Users will primarily access the website through a browser (Chrome, Firefox).
* All event organizers using the system will be pre-approved by the admin to

ensure only verified users have access to event creation and management

Features.

* Database schema is assumed to be relational and normalized.

**3.3** **Significant Design Packages**

The software design follows a **Layered (N-Tier) Architecture**, where each package/module is organized based on its responsibility in the system. The system is structured into presentation, application, business logic, and data access layers, each encapsulating related functionality.

#### **1. Authentication Package**

* **Layer**: Application & Business Logic Layers
* **Responsibilities**:
  + Handles user login and signup
  + Manages role-based access (Admin, Organizer, Student)
  + Works with JWT for secure authentication

#### **2. Event Management Package**

* **Layer**: Business Logic & Data Access Layers
* **Responsibilities**:
  + Allows organizers/admins to create, edit, delete, and view events
  + Interfaces with the database to persist event data
  + Ensures events follow approval workflows (pending → approved → completed)

#### **3. Registration Module**

* **Layer**: Application & Business Logic Layers
* **Responsibilities**:
  + Enables students to register or unregister for events
  + Validates eligibility and registration rules
  + Interacts with the database for registration tracking

#### **4. Admin Panel**

* **Layer**: Presentation & Application Layers
* **Responsibilities**:
  + Web-based UI for managing users, events, and system settings
  + Provides data insights like participation stats and financial summaries
  + Connects with services like EventService and UserService to fetch and update data

#### **5. Reporting & Expense Module**

* **Layer**: Business Logic Layer
* **Responsibilities**:
  + Tracks expenses for events
  + Generates summary reports in CSV

**3.4** **Dependent External Interfaces**

The table below lists the public interfaces this design requires from other modules or applications.

|  |  |  |
| --- | --- | --- |
| **External Application and Interface Name** | **Module Using the Interface** | **Functionality/ Description** |
| University Student Database (if integrated) | Authentication Module | Verifies user identity or pre-populates student info (if allowed). |

**3.5** **Implemented Application External Interfaces (and SOA web services)**

|  |  |  |
| --- | --- | --- |
| Interface Name | Module Implementing the Interface | Functionality / Description |
| POST /register | Authentication Module | Registers a new user with validation, password hashing, and DB insertion. |
| POST /login | Authentication Module | Authenticates user and returns JWT token. |
| POST /create-event | Event Module | Creates a new event (for organizers). Requires JWT token. |
| GET /events | Event Module | Retrieves all events for users (excluding denied). |
| POST /approve-event/:id | Event Module | Approves an event (admin only). |
| POST /deny-event/:id | Event Module | Denies an event (admin only). |
| POST /register-event | Registration Module | Registers a user for an event. Checks event capacity. |
| GET /my-registrations | Registration Module | Lists events registered by the logged-in user. |
| POST /add-expense | Expense Module | Adds an expense to an event (organizer only). |
| GET /expenses/:eventId | Expense Module | Fetches all expenses for a given event. |
| GET /expenses/totals/:eventId | Expense Module | Gets total expenses and remaining budget for an event. |

**4** **Logical View**

This section describes the design of the Event Management System in terms of layers

and module responsibilities. It includes class design and use case realization based on

actual components implemented in the Node.js backend.

**4.1 Design Model**

The system is logically divided into modules that handle specific responsibilities. These modules are designed with modularity and maintainability in mind, and reflect a layered (N-tier) structure.

The main modules are:

#### **1. Authentication Module**

* **Responsibilities**: Handles user registration, login, and JWT authentication.
* **Main Components**:
  + register (POST /register) – validates input and stores hashed password.
  + login (POST /login) – checks credentials and generates JWT.
  + verifyToken – middleware that checks validity of JWT tokens.

#### **2. Event Module**

* **Responsibilities**: Manages event creation, approval, and visibility.
* **Main Components**:
  + create-event – organizers create events.
  + approve-event, deny-event – admins approve or reject events.
  + get-events, get-approved-events – retrieves event lists based on status.

#### **3. Registration Module**

* **Responsibilities**: Manages student registrations and cancellations.
* **Main Components**:
  + register-event – students register for approved events.
  + cancel-registration – cancel existing registrations.
  + my-registrations – lists a student's registered events.

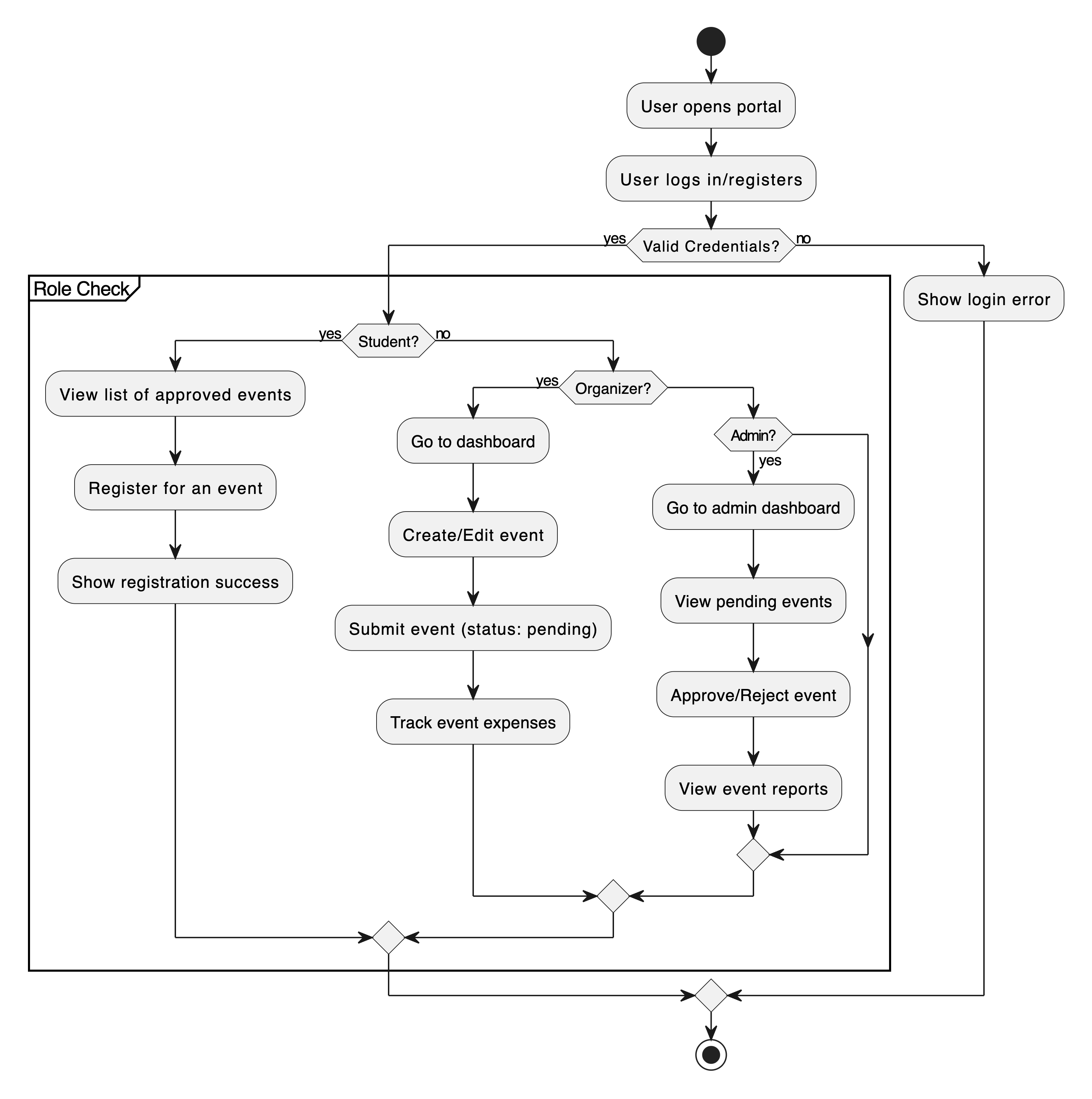
#### **4. Expense Module**

* **Responsibilities**: Tracks spending and budgeting per event.
* **Main Components**:
  + add-expense – adds new expenses.
  + get-expenses, get-expense-totals – retrieve and summarize spending data.

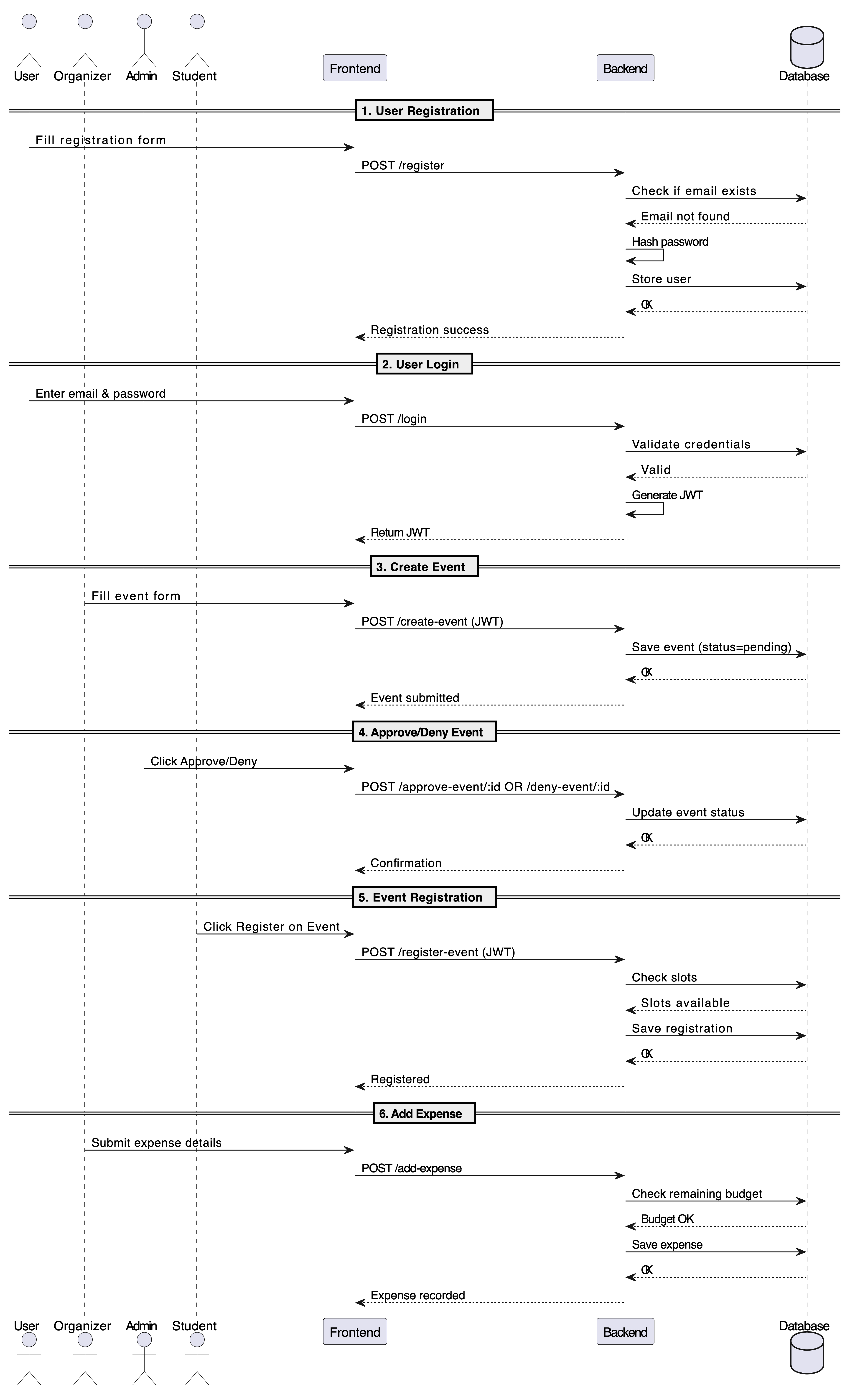
**4.2** **Use Case Realization**

This section describes how each major use case is implemented using the modules above

**ACTIVITY DIAGRAM**



**SEQUENCE DIAGRAM**



**1. User Registration**

* User submits registration data via /register.
* System checks if the email already exists.
* If not, password is hashed and data is stored in users.

#### **2.User Login**

* User submits credentials via /login.
* If valid, system returns a signed JWT.

#### **3. Create Event**

* Organizer sends event details to /create-event (JWT required).
* Event is stored in DB with status pending.

#### **4. Approve or Deny Event**

* Admin sends request to /approve-event/:id or /deny-event/:id.
* Event status is updated in the events table.

#### **5. Event Registration**

* Student submits registration via /register-event (JWT required).
* System checks if slots are available and stores registration.

#### **6. Add Expense**

* Organizer sends expense data via /add-expense.
* System checks remaining budget and stores the expense.

**5.Data View**

This application uses a **MySQL** database to store persistent data related to users, events, registrations, and expenses. The data is accessed through SQL queries inside route handlers using the mysql2 package.

**5.1** **Domain Model**

The domain consists of the following key entities:

* **User**: Represents students, organizers, and admins. Contains name, email, password (hashed), and role.
* **Event**: Represents an event created by an organizer. Contains event details like name, date/time, venue, budget, and status (pending/approved/rejected).
* **Registration**: Links users to events. Tracks registration status (Registered/Cancelled).
* **Expense**: Represents money spent in categories per event. Tracks actual amount spent and budget tracking.

**Relationships:**

* One **user** can create many **events**.
* One **user** can register for many **events**.
* One **event** can have many **registrations**.
* One **event** can have many **expenses**.

**5.2 Data Model (persistent data view)**

The application uses the following tables:

* users
* events
* registrations
* Expenses

**5.2.1** **Data Dictionary**

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Column Name | Description | Type |
| users | id | Unique ID for the user | INT (PK) |
| users | name | User’s full name | VARCHAR |
| users | email | Email address (unique) | VARCHAR |
| users | password | Hashed password | VARCHAR |
| users | role | admin, organizer, or student | VARCHAR |
| events | id | Unique ID for the event | INT (PK) |
| events | organizer\_id | FK to users.id | INT (FK) |
| events | event\_name | Name of the event | VARCHAR |
| events | date\_time | Date and time of the event | DATETIME |
| events | venue | Location of the event | VARCHAR |
| events | description | Event description | TEXT |
| events | max\_students | Max registrations allowed | INT |
| events | total\_budget | Budget for the event | DECIMAL |
| events | status | Event status | VARCHAR |
| events | rejection\_reason | Reason if rejected | TEXT |
| events | created\_at | Timestamp of creation | TIMESTAMP |
| registrations | id | Unique ID for registration | INT (PK) |
| registrations | user\_id | FK to users.id | INT (FK) |
| registrations | event\_id | FK to events.id | INT (FK) |
| registrations | student\_name | Name of the student | VARCHAR |
| registrations | email\_id | Email of the student | VARCHAR |
| registrations | event\_name | Name of the event | VARCHAR |
| registrations | status | Registered or Cancelled | VARCHAR |
| registrations | registered\_at | Timestamp of registration | TIMESTAMP |
| expenses | id | Unique ID for expense | INT (PK) |
| expenses | event\_id | FK to events.id | INT (FK) |
| expenses | category | Category of expense | VARCHAR |
| expenses | actual\_spent | Amount spent | DECIMAL |
| expenses | total\_budget | Event budget | DECIMAL |
| expenses | total\_amount\_spent | Cumulative amount spent | DECIMAL |
| expenses | created\_at | Timestamp of entry | TIMESTAMP |

**6.Exception Handling**

The application uses basic exception handling to ensure stable operation and user

feedback during errors.

* **When Exceptions Occur:**
  + When required fields are missing (e.g., during registration or login).
  + When a user provides invalid credentials.
  + When trying to access a protected route without a valid token.
  + When database operations fail.
  + When registering for full events or non-existent events.
* **How They Are Handled:**
  + Errors are handled using if checks and try/catch blocks inside route functions.
  + Appropriate HTTP status codes are returned (e.g., 400 for bad requests, 401 for unauthorized, 500 for server/database errors).
  + The server sends a clear error message in the JSON response.
* **Logging:**
  + Errors are logged to the console using console.error().
* **Follow-up Action:**
  + Users receive an error message response.
  + Developers can use console logs to debug during development.

**7.Configurable Parameters**

This table describes the simple configurable parameters (name / value pairs).

|  |  |  |
| --- | --- | --- |
| **Configuration Parameter Name** | **Definition and Usage** | **Dynamic?** |
| MAX\_EVENT\_REGISTRATIONS | Maximum number of users allowed to register | YES |
| EVENT\_APPROVAL\_REQUIRED | Boolean flag to indicate if events need manager approval before publishing | YES |

**8. Quality of Service**

This section outlines the application's strategies and design considerations to ensure high availability, secure access, performance scalability, and system monitoring for a reliable event management experience.

**8.1** **Availability**

The application connects to a MySQL database using Node.js and Express. It handles database connection errors and prints logs for successful or failed connections. All routes are available as REST APIs and are designed to run continuously unless the server is manually stopped. There is no special downtime handling or scheduling system built-in.

**8.2** **Security and Authorization**

The system uses **JWT** for authentication and **bcryptjs** for password hashing. Users can register with roles like admin, organizer, or student. Access to protected routes is controlled using a middleware (verify Token) that checks the JWT. Admin-only routes check if the logged-in user's role is admin.*]*

**8.3** **Load and Performance Implications**

The application uses asynchronous calls and promise-based MySQL queries to handle requests. There is no load balancing or caching implemented. Data is stored in tables like users, events, registrations, and expenses, and these are queried directly without pagination or optimization for large loads.

**8.4** **Monitoring and Control**

There is no active monitoring or logging framework. The server uses basic console.log() statements for important actions like database connection and token generation. There are no daemons or background processes. Errors are sent in the API responses for debugging.