Chittagong University Campus Event Management System

Database Project Report

Group-37

A project submitted to Dr. Rudra Pratap Deb Nath, Associate Professor, Department of Computer Science and Engineering, Chittagong University (CU) in partial fulfillment of the requirements for the Database Systems Lab course. The project is not submitted to any other organization at the same time.

Table 1: Details of Group-37

Roll Id	Name	Date	Supervisor Approval
22701065	Sanzid Islam Mahi	15.12.2024	

Acknowledgments

We would like to express our gratitude to Dr. Rudra Pratap Deb Nath for providing valuable guidance and suggesting resources like *Meetup.com* [5], which helped shape the direction of this project.

Contents

List of Figures

List of Tables

Listings

Abstract

The **Campus Event Management System** is a web-based platform designed to simplify the creation, management, and registration of campus events. Built with React.js for the front-end, Node.js for the back-end, and MySQL for the database, the system ensures a seamless experience for students and teachers. The platform allows users to browse events by categories, register for events of interest, and receive event updates. Users can also create and manage their own events. Key features include a user-friendly interface, dynamic event creation, and event registration tracking. This system streamlines event coordination, fostering engagement and connectivity within the campus community. Looking ahead, potential enhancements include the development of a mobile app, automated email notifications, and improved validation mechanisms. The **Campus Event Management System** successfully integrates modern web technologies to address event management challenges, promoting collaboration and communication among users.

1 Introduction

We submit this database project as part of our CSE-414, Database Management Systems lab course. The goal of this project is to develop a database-driven web application, called **"Campus Event Management System"**, which enables students and teachers to organize and participate in events on campus. This system simplifies the process of event creation, registration, and management. It is accessible via computers and mobile devices without requiring any additional downloads or installations.

The system leverages concepts such as Entity-Relationship (ER) diagrams, relational models, normalization, and other database design principles studied in our course. This section outlines the background, problem statement, and system development process.

1.1 Background and Motivation

The idea for this project stems from the challenges faced by students and teachers in organizing and discovering campus events. Currently, most campus events rely on manual processes, such as word-of-mouth communication, printed posters, or scattered social media announcements. This often leads to poor visibility, missed opportunities, and inefficient event management.

A survey conducted among students revealed that:

- 50–60% of students are unaware of events happening on campus.
- 20–30% find it difficult to register for events due to the lack of proper systems.
- Many event organizers struggle to manage registrations, track attendees, and gather post-event feedback.

To address these issues, our project, the *Campus Event Management System*, aims at providing a centralized platform for creating, managing, and participating in events. By implementing this system, we hope to foster greater engagement and enhance the campus experience for both students and teachers.

Additionally, inspiration for this project is drawn from platforms such as Meetup.com[5], which demonstrates effective ways to organize and manage events. Analyzing such platforms helped identify key features like event categorization, user registration, and organizer tools, which will guide the development of this system.

1.2 Problem Statement

Currently, there is no unified system for managing events on campus. Organizers face challenges in promoting their events and managing participant registrations efficiently. Similarly, students and teachers struggle to find events of interest, leading to low attendance and missed opportunities. The manual processes currently in place are time-consuming, disorganized, and lack proper data storage.

To overcome these limitations, we developed the **Campus Event Management System**, a web-based platform that enables:

- Event organizers to create, update, and promote events.
- Participants (students and teachers) to browse events, register, and provide feedback.
- Efficient management of event categories, locations, and registrations through a centralized database.

This system ensures better visibility of events, simplifies the registration process, and provides valuable data for organizers.

1.3 System Definition

A system can be defined as a group of components or subsystems that work together to achieve a specific goal. The **Campus Event Management System** is a secure, web-based platform that facilitates the creation and management of campus events. Key features of the system include:

- Centralized storage of event data, including event details, categories, locations, and organizers.
- Easy registration and participation management for students and teachers.
- CRUD (Create, Read, Update, Delete) operations for event organizers and administrators.
- Filtering and sorting of events based on categories, dates, or locations.

The system enhances event visibility and ensures efficient event management through a robust database design.

1.4 System Development Process

The development of the Campus Event Management System followed a structured process. The steps are illustrated in figure 1 and described below:

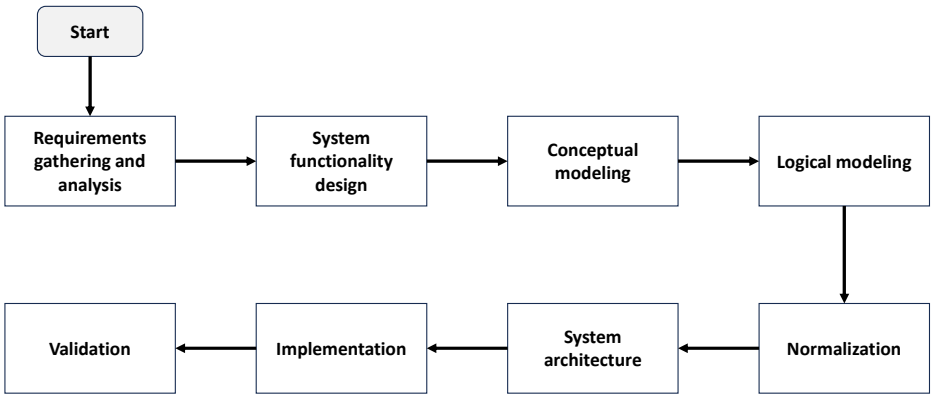

Figure 1: System Development Process

1. **Requirements Gathering and Analysis**: We conducted interviews and surveys to gather requirements from potential users, including students, teachers, and event organizers. The requirements is documented for further analysis.

2. **System Functionality Design**: A system model is designed to outline the platform's functionality, including event creation, registration, and management workflows.

3. **Conceptual Modelling (ER Diagram)**: Entities such as events, users, locations, and registrations is identified, and their relationships is represented through an ER diagram.

4. **Logical Modelling**: The ER diagram is mapped to a relational model to define tables, attributes, and relationships.

5. **Normalization**: The database design is normalized to eliminate redundancy and ensure data integrity.

6. **System Architecture**: A three-tier architecture is adopted, including the database layer, application logic layer, and user interface layer.

7. **Implementation (Physical Design)**: The system is implemented using technologies such as MySQL for the database, React.js for the front-end, and Node.js for the back-end.

1.5 Organization

Section 1 gives the overview of the project, including the background, motivation, and problem statement. Section 2 describes how the project and the resources are managed. Section 3 explains the process of gathering and analyzing system requirements. Section 4 presents the conceptual model of the system, including the ER diagram and entity relationships. Section 5 maps the conceptual model onto a relational schema and discusses the logical design. Section 6 discusses the normalization process to eliminate redundancy and ensure data integrity.

Section 7 outlines the system architecture, describing the components and their interactions. Section 8 provides details on the implementation process, including technologies and tools used. Section 9 focuses on the validation of the system through testing and verification of functionality. Section 10 explains the deployment process of the system. Finally, the conclusion and the pointers to the future work are outlined in Section 11.

2 Project Management

The process of guiding a project from the outset through its lifespan is referred to as project management. To finish a project within the parameters of time, quality, and scope is the primary purpose of project management. Project management includes initiation, monitoring and control, tools and resources, and closure 11. This section describes the key aspects of managing the development of the Campus Event Management System.

2.1 Initiation

The idea for the project is inspired by the growing need for a system that simplifies event organization and participation in a campus environment. Managing events manually can lead to inefficiencies, data loss, and communication gaps. The aim is to automate the process using a web-based solution.

2.1.1 Project Vision

The project's vision is to develop a secure and efficient Campus Event Management System that facilitates event creation, registration, and categorization for students and teachers, enhancing user engagement within a campus setting.

2.2 Monitoring and Control

Throughout the project, a structured workflow is followed to ensure timely progress and consistent quality. The following steps is taken to monitor and control the project activities:

- A personal timeline is created to break down tasks into smaller milestones, ensuring steady progress over weeks.
- Regular progress checks is performed after each major milestone, such as database design, front-end development, and back-end integration.
- Debugging and testing phases is incorporated at every stage to identify and resolve issues early.

2.3 Contribution and Responsibilities

As the sole developer, we is responsible for every stage of the project, from system design to implementation. Key contributions include:

- **Front-end Development:** Designed a user-friendly interface for event creation, registration, and browsing using *Visual Studio Code*[11].
- **Back-end Development:** Developed server-side logic using *Node.js*[8] and connected the system to the *MySQL* database.
- **Database Design:** Designed a robust relational database schema in *MySQL Workbench*[6] to handle users, events, and registrations.
- **System Testing:** Ensured the system is functional and bug-free through unit testing and integration testing.
- **Report Writing:** Documented the entire development process, including system architecture, database design, and implementation details.

2.4 Tools and Technologies Used

The following tools and technologies is utilized for building the Campus Event Management System:

- **Code Editor:** Visual Studio Code[11] Visual Studio Code, a versatile and powerful code editor, is used for writing and managing all the code, including the front-end, back-end, and database scripts.
- **Front-end:** React.js[10], HTML, CSS, and JavaScript
  React.js is used as the front-end library to build a dynamic and responsive user interface. HTML, CSS, and JavaScript is used to structure, style, and enhance the interactivity of the application.
- **Back-end:** Node.js[8]
  Node.js is used to develop the server-side logic and APIs for the system, ensuring seamless communication between the front end and the database.
- **Database:** MySQL Workbench[7]
  MySQL Workbench is used to design and implement the relational database schema, ensuring efficient storage and retrieval of data.
- **Version Control:** GitHub[3]
  GitHub is used to manage and version-control the code throughout the project lifecycle.

- **Documentation:** LaTeX[4] and Overleaf[9]
  LaTeX is used to write this report, ensuring professional formatting and presentation. Overleaf provided an online collaborative environment for managing the report.

- **Diagramming Tools:** Draw.io[1]
  Draw.io is used to design the ER diagram and system architecture diagrams.

- **AI Assistance:** ChatGPT[12]
  ChatGPT is used to assist with generating ideas, refining project documentation, and providing technical guidance throughout the development process.

2.5 Closure

Working on the Campus Event Management System has been a rewarding experience. The successful completion of this project is made possible through a systematic approach to development and the use of appropriate tools and technologies. This project not only improved our technical skills in web development, database design, and system architecture but also deepened our understanding of project management concepts.

3 Requirement Gathering and Analysis

Requirement analysis in this project involves identifying the data that need to be stored in the database and understanding how they will be accessed by different users. The system is being developed after discussing the needs with one of the campus event organizers. Ensuring clear and complete requirements before beginning is essential for creating a system that meets the needs of all stakeholders.

3.1 Process of Requirement Analysis

3.1.1 Discussion with an Event Organizer and students

Discussions with an event organizer and a student form the core of the analysis. Through these conversations, the aim is to identify the problems encountered in organizing campus events, managing attendees, and allocating resources. The following are some key points of the discussion.

- **Question 1:** What are some challenges in managing events and facilitating connections between students and teachers?
- **Event Organizer:** We face issues in organizing events across various interests, as we have to coordinate with departments and manage resources such as locations and materials. In addition, students may not know about events that match their interests, limiting opportunities to connect.
- **Question 2:** How does the current system limit participant engagement?
- **Student:** Without a centralized platform, it is challenging to keep track of events and connect with others who share similar interests. We want a way to see all available events, know who's attending, and connect with participants after events.
- **Question 3:** What improvements would you like to see in event notifications and updates?
- **Event Organizer:** Notifications for event updates, changes, or cancellations should be instant and accessible. Participants often don't get updates in time, which can make engagement difficult.

3.1.2 Understanding the Requirement

After discussing these points, it is concluded that the system must handle all event-related data automatically, minimizing errors and delays. The system should allow organizers and participants to manage and access information seamlessly and ensure reliable, timely communication of updates.

Basic information requirements identified are:

- **Event Information**: Event name, date, time, location, and organizer.
- **Participant Information**: Name, ID, email.
- **Location Information**: area details, capacity and availability for reservation.

3.2 Identifying Stakeholders

Since the system is being developed to address event management challenges faced on campus, the key stakeholders are:

- **Event Organizers** (teachers, students, and staff involved in the planning and management of events).
- **Participants** (students, teachers).
- **Campus Administration** (managing location bookings and maintaining event records).

The goal of this project is to create a user-friendly and effective solution for all these stakeholders.

3.3 System Specification

After analyzing the requirements and identifying the stakeholders, a **web-based system** has been selected for development. The next step is to design a conceptual model, focusing on a high-level view to confirm that it aligns with the specified requirements.

The system will include features for event creation, participant registration, resource management, and real-time notifications, ensuring that all necessary information is accessible and manageable for everyone involved.

4 Conceptual Model for Campus Event Management System

The conceptual model for the Campus Event Management System is developed using the Entity-Relationship Model (ERM)[2]. The ERM is chosen because of its clarity in representing data structures and its relevance to our coursework. ER diagrams define the logical structure of the database, showing how data elements and their relationships interact within the system.

4.1 Entity-Relationship Model

An **entity-relationship diagram (ERD)** visually represents the data structure of the Campus Event Management System, showing how entities and their attributes relate to each other.

Key Concepts in the ER Model:

- **Entity**: An entity represents a real-world object or concept, such as an event or user. Entities are represented as tables in the database, and each row represents a specific instance.
- **Attribute**: Attributes represent the properties or characteristics of an entity. For example, attributes of the `Event` entity include `event_name`, `event_date`, and `location_id`.
- **Relationship**: Relationships define the interactions between entities. Common types of relationships include:
  - **One-to-One**
  - **One-to-Many**
  - **Many-to-One**
  - **Many-to-Many**

4.2 Detailing the Entity-Relationship Model

From the analysis, the following **entities** and **relationships** have been identified for the Campus Event Management System.

4.2.1 Entity Types

1. **User**: Represents users of the system with attributes:
   - `user_id` (Primary Key)
   - `name`
   - `email`
   - `password`
   - `contact_number`
   - `profile_picture`
2. **Event**: Represents each campus event, with attributes:
   - `event_id` (Primary Key)
   - `event_name`
   - `description`
   - `event_date`
   - `start_time`
   - `end_time`
   - `max_attendees`
   - `category_id` (Foreign Key referencing `Event_Category.category_id`)
   - `venue_id` (Foreign Key referencing `venue.venue_id`)
   - `user_id` (Foreign Key referencing `user.user_id`)
3. **Location**: Represents locations of the venues:
   - `location_id` (Primary Key)
   - `location_name`
4. **Venue**: Represents locations where events are held, with attributes:
   - `venue_id` (Primary Key)
   - `venue_name`
   - `location_id` (Foreign Key referencing `Location.location_id`)
5. **Event Category**: Represents categories for events, with attributes:
   - `category_id` (Primary Key)

- `category_name`

6. **Notifications**: Represents the notifications after an event update:
   - `notification_id` (Primary Key)
   - `event_id` (Foreign Key referencing `event.event_id`)
   - `notification_text`
   - `created_at`
7. **Reviews**: Represents the reviews given by user after events end:
   - `review_id` (Primary Key)
   - `event_id` (Foreign Key referencing `event.event_id`)
   - `user_id` (Foreign Key referencing `user.user_id`)
   - `review_text`
   - `rating`
   - `review_date`

4.2.2 Relationship Types

The relationships between the entities are as follows:

- **organized_by**: Connects `User` and `Event`, where each user (acting as an organizer) can manage multiple events.
- **registers**: Connects `User` (as a participant) and `Event`, where a participant can register for multiple events, and each event can have multiple participants.
  - `username` (Foreign Key referencing `User.username`)
  - `event_id` (Foreign Key referencing `Event.event_id`)
  - `registration_date`
- **held_in**: Connects `Event` and `Venue`, where each event is held at a specified venue.
- **located_in**: Connects `Venue` and `Location`, where each venue is located in a specified location.
- **categorized_by**: Connects `Event` and `Event_Category`, the category of each event.

- **has_updates**: Connects `Event` and `Notifications`, where each update on event stored as a notification.
- **gives**: Connects `User` and `Reviews`, where each user can give review to the attended events.
- **about**: Connects `Event` and `Reviews`, where each event can have a review.

This ER model provides a high-level structure for managing campus events, representing how data is organized within the Campus Event Management System. The relationships between entities allow efficient linking of users, events, locations, and categories, facilitating seamless interactions within the system.

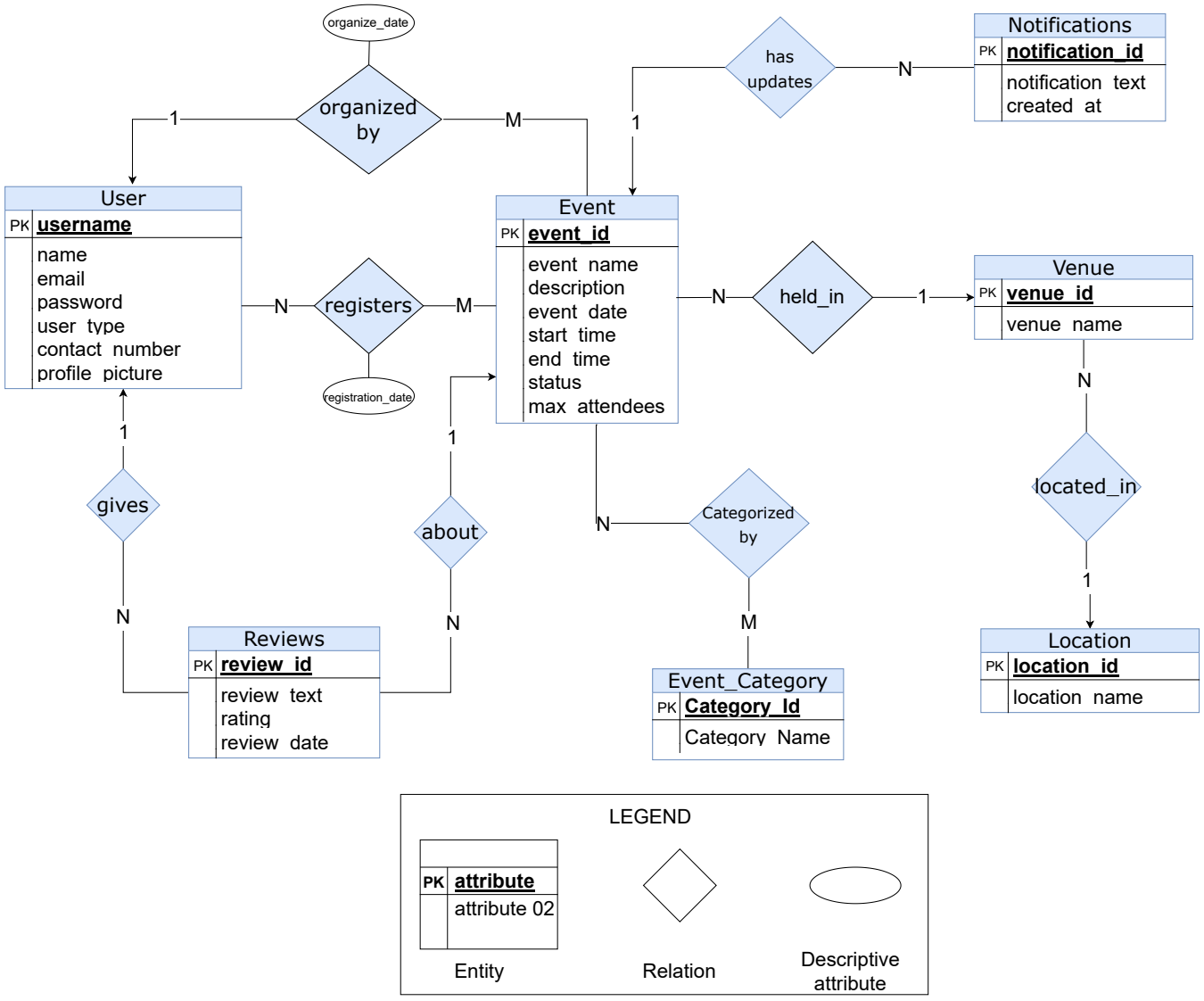

Figure 2: ER Diagram

5 Logical Modelling

After defining the conceptual model, we proceed with the logical data model for the Campus Event Management System. The logical model outlines the structure of the data items and their relationships in a formal way. For our system, we use the relational model.

5.1 Relational Model

In a relational model, data and their relationships are represented through a collection of interlinked tables. Each table contains rows and columns, where each column represents an attribute of an entity, and each row holds a record. The relational schema defines how each table (relation) is structured, specifying the table name and a list of attribute names, each tied to a specific domain.

5.1.1 Tables and Attributes

- **User**
  `user(user_id, username, email, password, contact_number, profile_picture)`
  Primary Key: username

- **Location**
  `location(location_id, location_name)`
  Primary Key: location_id

- **Venue**
  `venue(venue_id, venue_name, location_id)`
  Primary Key: venue_id **Foreign Keys:**
  location_id → location(location_id)

- **Event_Category**
  `event_category(category_id, category_name)`
  Primary Key: category_id

- **Event**
  `event(event_id, event_name, description, event_date, start_time, end_time, status, max_attendees, category_id, venue_id)`
  Primary Key: event_id
  Foreign Keys:

category_id → event_category(category_id)
venue_id → venue(venue_id)

- **Registers**
  `registers(username, event_id, registration_date)`
  Primary Key: (username, event_id)
  Foreign Keys:
  username → user(username)
  event_id → event(event_id)

- **Notifications**
  `notifications(notification_id, event_id, notification_text, created_at)`
  Primary Key: (notification_id)
  Foreign Keys:
  event_id → event(event_id)

- **Reviews**
  `reviews(user_id, event_id, rating, review_text, review_date)`
  Primary Key: (username, event_id)
  Foreign Keys:
  user_id → user(user_id)
  event_id → event(event_id)

6 Normalization of Database Tables

Normalization is a process used to minimize or remove data redundancy in a set of relations. The primary goal of normalization is to eliminate anomalies that can occur during data insertion, update, or deletion, thus making the database consistent, dependency-preserving, and free of redundancy. Below, we analyze the normalization forms, specifically the 3rd normal form (3NF), for each table in the campus event management database.

6.1 Normalizing user Table

The `user` table 2 has the attributes:

$$\{\texttt{user\_id}, \texttt{name}, \texttt{email}, \texttt{password}, \texttt{contact\_number}, \texttt{profile\_picture}\}$$

user_id	name	email	password	contact_number	profile_picture
1	Sanzid	sanzid.csecu@gmail.com	$2b$10$tiwYa...	1871114187	uploads\1733514884948.jpg
2	Nazmul Hasan	nazmul@gmail.com	$2b$10$WcW	173	uploads\1733478884284.jpg
3	Aryan	aryan@gmail.com	$2b$10$9hdd	11	uploads\1733479421720.jpg
4	Dewan Shahedul Islam	akik@gmail.com	$2b$10$twiS	1629900045	uploads\1733427010881.jpg
5	Zaheen	zaheen@gmail.com	$2b$10$scyq	1871114187	uploads\1733497014057.jpg

Table 2: User Table

Based on the demo data, we identify the following functional dependencies and analyze the candidate keys, prime attributes, and non-prime attributes to determine its normalization status.

- **Functional Dependencies:**
  - $\underline{\texttt{user\_id}} \rightarrow$ `user_id, name, email, password, contact_number, profile_picture`
  - `email`$\rightarrow$ `user_id, name, email, password, contact_number, profile_picture`
- **Candidate Keys:** `user_id, email`
- **Prime Attributes:** `username, email`
- **Non-Prime Attributes:** `name, password, contact_number, profile_picture`

- **Analysis:** Based on our analysis, all the functional dependencies in the `user` table have a candidate key on the left-hand side. This satisfies the requirements of 3NF, and no transitive dependencies are present. All non-prime attributes (name, email, password, user_type, department, and contact_number) are fully functionally dependent on the candidate key, `username`.
- **Conclusion:** The `user` table is in 3NF.

6.2 Normalizing venue Table

The `venue` table 3 has the attributes:

$$\{\texttt{venue\_id}, \texttt{venue\_name}, \texttt{location\_id}\}$$

venue_id	venue_name	location_id
28	room 309	18
29	room 310	18
30	room 413	18
31	room 414	18
32	virtual classroom	18
33	Electrical Lab	18
34	Reading room	26
35	Tv room	26
36	Hall field	26
37	Guest room	26
38	Swimming pool	26
39	Badminton court	26

Table 3: Venue Table

- **Functional Dependencies:**
  - `venue_id` $\rightarrow$ `venue_name, location_id`
- **Candidate Keys:** `venue_id`
- **Prime Attributes:** `venue_id`
- **Non-Prime Attributes:** `venue_name, location_id`

- **Analysis:** The functional dependency has a candidate key (`venue_id`) on the left-hand side. This satisfies the conditions of 3NF, and there are no transitive dependencies. All non-prime attributes (`venue_name`, `location_id`) are fully functionally dependent on `venue_id`.
- **Conclusion:** The `venue` table is in 3NF.

6.3 Normalizing location Table

The `location` table 4 has the attributes:

$$\{\texttt{location\_id}, \texttt{location\_name}\}$$

location_id	location_name
1	Dept of Bengali
2	Dept of English
13	Dept of Management
18	Dept of Computer Science and Engineering
19	Dept of Law
20	Alaol Hall
21	Shah Jalal Hall
23	F Rahman Hall
26	Shaheed Abdur Rab Hall

Table 4: Location Table

- **Functional Dependencies:**
  - `location_id` $\rightarrow$ `location_name`
- **Candidate Keys:** `location_id`
- **Prime Attributes:** `location_id`
- **Non-Prime Attributes:** `location_name`
- **Analysis:** The functional dependency has a candidate key (`location_id`) on the left-hand side. This satisfies the conditions of 3NF, and there are no transitive dependencies. All non-prime attributes (`location_name`) are fully functionally dependent on `location_id`.
- **Conclusion:** The `location` table is in 3NF.

6.4 Normalizing event_category Table

The `event_category` table 5 has the attributes:

$$\{\texttt{category\_id}, \texttt{category\_name}\}$$

category_id	category_name
1	sports
2	music
3	drama
6	seminar
7	workshop
17	recreational
18	orientation
19	debate
20	career development
21	hackathon

Table 5: Event_category Table

- **Functional Dependencies:**
  - `category_id` → `category_name`
- **Candidate Keys:** `category_id`
- **Prime Attributes:** `category_id`
- **Non-Prime Attributes:** `category_name`
- **Analysis:** The functional dependency has a candidate key (`category_id`) on the left-hand side, meeting 3NF requirements. There are no transitive dependencies, and all non-prime attributes are fully functionally dependent on `category_id`.
- **Conclusion:** The `event_category` table is in 3NF.

6.5 Normalizing event Table

The `event` table 6 has the attributes:

{ `event_id`, `event_name`, `description`, `event_date`, `start_time`, `end_time`, `status`, `max_attendees`, `category_id`, `venue_id` }

event_id user_id	event_name	description	event_date	start_time	end_time	max_attendees	category_id	venue_id
1 2	Zebra Crossing	a	12/28/2024	7:17:00	2:19:00	12	NULL	8
2 1	1234	f	12/12/2024	22:03:00	22:03:00	12	3	27
3 1	Python Program	python	12/13/2024	1:24:00	7:25:00	25	7	5
4 4	Dance Class	.	12/21/2024	1:27:00	5:22:00	2	13	15
5 4	Hiking Trip	.	12/6/2024	4:24:00	4:22:00	40	25	26
6 4	Singing lesson	/	12/21/2024	4:23:00	1:25:00	12	13	18
7 4	Programming contest 2025	.	1/8/2025	9:00:00	13:00:00	60	9	18
9 3	Guitar lesson	kkk.	12/28/2024	18:07:00	16:13:00	12	13	18
10 1	1234	a	12/5/2024	22:09:00	12:08:00	12	1	27

Table 6: Event Table

- **Functional Dependencies:**
  - `event_id` → `event_name`, `description`, `event_date`, `start_time`, `end_time`, `status`, `max_attendees`, `category_id`, `venue_id`
- **Candidate Keys:** `event_id`
- **Prime Attributes:** `event_id`
- **Non-Prime Attributes:** `event_name`, `description`, `event_date`, `start_time`, `end_time`, `status`, `max_attendees`, `category_id`, `venue_id`
- **Analysis:** All functional dependencies have a candidate key (`event_id`) on the left-hand side. There are no transitive dependencies, and all non-prime attributes are fully functionally dependent on `event_id`, fulfilling 3NF.
- **Conclusion:** The `event` table is in 3NF.

6.6 Normalizing registers Table

The `registers` table 7 has the attributes:

{`user_id`, `event_id`, `registration_date`}

- **Functional Dependencies:**
  - `(user_id, event_id)` → `registration_date`

user_id	event_id	registration_date
1	4	12/6/2024
1	5	12/6/2024
2	10	12/6/2024
3	1	12/5/2024
5	1	12/6/2024

Table 7: Event Registration Information Table

- **Candidate Keys:** `(user_id, event_id)`
- **Prime Attributes:** `user_id, event_id`
- **Non-Prime Attributes:** `registration_date`
- **Analysis:** The only functional dependency has a composite candidate key (`username, event_id`) on the left-hand side, fulfilling 3NF. There are no transitive dependencies, and the non-prime attribute `registration_date` is fully functionally dependent on the composite key.
- **Conclusion:** The `registers` table is in 3NF.

6.7 Normalizing Reviews Table

The `reviews` table 8 has the attributes:

{`review_id`, `event_id`, `user_id`, `review_text`, `rating`, `review_date`}

- **Functional Dependencies:**
  - `review_id` → `event_id, user_id, review_text, rating, review_date`

review_id	event_id	user_id	review_text	rating	review_date
7	5	1	good	2	12/7/2024
8	10	2	nice	4	12/7/2024

Table 8: Event Review Information Table

- **Candidate Keys:** `review_id`
- **Prime Attributes:** `review_id`
- **Non-Prime Attributes:** `event_id, user_id, review_text, rating, review_date`
- **Analysis:** The functional dependency `review_id` → `event_id, user_id, review_text, rating, review_date` indicates that `review_id` is the primary key. All attributes are functionally dependent on the primary key, and there are no transitive dependencies. Thus, the table satisfies the requirements of 3NF.
- **Conclusion:** The `reviews` table is in 3NF.

6.8 Normalizing Notification Table

The `notification` table 9 has the attributes:

{`notification_id`, `event_id`, `notification_text`, `created_at`}

- **Functional Dependencies:**
  - `review_id` → `event_id, user_id, review_text, rating, review_date`

notification_id	event_id	notification_text	created_at
1	5	Event "CU Hackathon" updated. New Date: 2024-12-14.	12/7/2024
2	5	Event "CU Hackathon" updated. Start Time: 10:00.	12/8/2024

Table 9: Notification Information Table

- **Candidate Keys:** `notification_id`
- **Prime Attributes:** `notification_id`
- **Non-Prime Attributes:** `event_id, notification_text, created_at`
- **Analysis:** The functional dependency `notification_id` → `event_id, notification_text, created_at` indicates that `notification_id` is the primary key. All attributes are functionally dependent on the primary key, and there are no transitive dependencies. Thus, the table satisfies the requirements of 3NF.
- **Conclusion:** The `reviews` table is in 3NF.

7 System Architecture

We have illustrated how our system works in the figure 3 given below: The code running in the browser (front-end) handles user input and

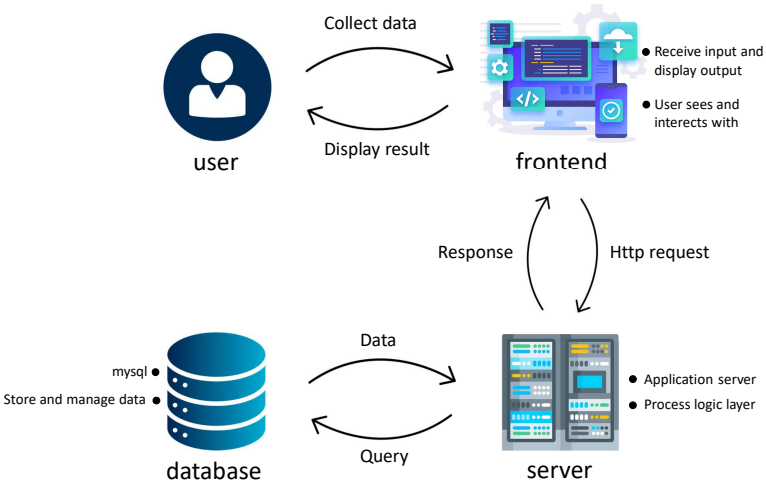

Figure 3: System Architecture

displays output, while the server (back-end) processes HTTP requests and interacts with the database.

7.1 Working Process of System Interface

System architecture refers to the design of a software system that includes multiple components. The architecture ensures that the system is reliable, maintainable, and scalable over time. In our system:

– **Front-End Layer:** Built with **React**[10], it handles user interactions and displays dynamic data.

- **Back-End Layer:** Developed using **Node.js**[8] and **Express**[**?**], it processes requests and communicates with the database.
- **Database Layer:** **MySQL**[7] is used to store and retrieve the required data.

When a user accesses the application, the browser sends a request to the server where the system is hosted. The server processes the request, interacts with the database, and sends the response back to the browser, where React dynamically updates the content.

7.2 Client-Side and Server-Side Code

- **Client-Side Code (React):** Executed on the user's browser, React is responsible for managing the user interface, handling input, and displaying dynamic content using a component-based architecture.
- **Server-Side Code (Node.js):** Executed on the server, Node.js processes client requests, handles business logic, and connects to the database.

HTTP APIs allow different parts of the application to request and exchange data seamlessly. Relational data is stored in tables, with each row representing a record.

7.3 System Functionality

The project architecture can be divided into three main components:

7.3.1 Front-End Part (Client/User Side)

The front-end is built using modern web technologies:

1. React.js
2. HTML
3. CSS
4. JavaScript

React provides a dynamic and modular structure for rendering the user interface efficiently.

7.3.2 Back-End Part (Server Side)

The back-end processes HTTP requests, performs business logic, and communicates with the database. Technologies used include:

1. Node.js
2. Express.js (Web Framework for Node.js)

7.3.3 Database (Data Layer)

The database layer manages data storage and retrieval. For this project:

- **MySQL:** A relational database used to store and organize data.
- **SQL Queries:** Used to retrieve, insert, and manipulate data.

7.4 Front-End and Back-End Design Components

7.5 React (Front-End)

React is a JavaScript library used for building dynamic and modular user interfaces. It efficiently updates and renders components based on user interactions.

7.5.1 CSS

CSS is used to style the React components, control the layout, and design responsive interfaces with modern styling features like animations and transitions.

7.5.2 JavaScript

JavaScript powers the React framework and handles client-side interactions, such as event handling and state management.

7.6 Node.js (Back-End)

Node.js is a JavaScript runtime that enables server-side scripting. Combined with Express.js, it handles HTTP requests, API routes, and business logic.

7.6.1 MySQL (Database)

MySQL serves as the core database for storing user information, event data, and registrations. SQL queries are executed through the back-end Node.js server.

8 Implementation

In this section, we'll go through how to connect a database to a website, how to get data out of the database, and how to construct a front-end. Our system has already been outlined Section 7. We'll demonstrate a portion of implementation of our project that utilizes the procedures stated in this section.

8.1 Create, Read, Update, and Delete from Database

To implement our system we had to do operations in database like create, update, delete and read.

8.1.1 Create Table

SQL query for creating user table is given below in Listing 1

```
create table user(
    user_id int primary key auto_increment,
    name varchar(50),
    email varchar(25),
    password varchar(200),
    contact_number varchar(11),
    profile_picture varchar(255)
);
```

Listing 1: SQL Query for creating user table

SQL query for creating location table is given below in Listing 2

```
create table location(
        location_id int primary key
           auto_increment,
        location_name varchar(25)

);
```

Listing 2: SQL Query for creating location table

SQL query for creating venue table is given below in Listing 3

```sql
create table Venue(
        venue_id int primary key auto_increment,
    venue_name varchar(25),

    location_id int,
    foreign key(location_id) references location
       (location_id)
);
```

Listing 3: SQL Query for creating venue table

SQL query for creating *event_category* table is given below in Listing 4

```sql
create table event_category(
        category_id int primary key
           auto_increment,
    category_name varchar(25)
);
```

Listing 4: SQL Query for creating `event_category` table

SQL query for creating event table is given below in Listing 5

```sql
create table event(
        event_id int primary key auto_increment
           ,
    event_name varchar(25),
    description varchar(100),
    event_date date,
    start_time time,
    end_time time,
    max_attendees int,

    category_id int,
    venue_id int,
    user_id int,
        foreign key (category_id) references
           event_category(category_id),
        foreign key (venue_id) references Venue(
           venue_id),
    foreign key (user_id) references user(
       user_id)
```

```
16
17 );
```

Listing 5: SQL Query for creating event table

SQL query for creating registers table is given below in Listing 6

```
1  create table registers(
2          user_id int,
3      event_id int,
4      registration_date date,
5
6      primary key(user_id,event_id),
7      foreign key (user_id) references user(
          user_id),
8      foreign key(event_id) references event(
          event_id)
9  );
```

Listing 6: SQL Query for creating registers table

SQL query for creating notifications table is given below in Listing 7

```
1  CREATE TABLE notifications (
2      notification_id INT AUTO_INCREMENT PRIMARY
          KEY,
3      event_id INT NOT NULL,
4      notification_text VARCHAR(255) NOT NULL,
5      created_at TIMESTAMP DEFAULT
          CURRENT_TIMESTAMP,
6      FOREIGN KEY (event_id) REFERENCES event(
          event_id) ON DELETE CASCADE
7  );
```

Listing 7: SQL Query for creating notifications table

8.1.2 Read data from table

SQL query for reading data from user table is given below in Listing 8

```
1
2  select * from user;
```

Listing 8: SQL Query for reading user table

8.1.3 Update data of table

SQL query for updating data from user table is given below in Listing 9

```
1
2 UPDATE user SET name = 'Sanzd␣I' WHERE user_id =
      1;
```

Listing 9: SQL Query for updating data from user table

8.1.4 Delete data from table

SQL query for deleting data from event table is given below in Listing 10

```
1
2 DELETE FROM event where event_id = `$id`;
```

Listing 10: SQL Query for deleting data from event table

8.2 Database Connection

Listing 11 shows the database connection code for MySQL.

```
1
2 // Import MySQL2 library
3 const mysql = require("mysql2");
4
5 // Create database connection
6 const db = mysql.createConnection({
7   host: "localhost",      // Database host
8   user: "root",           // Database user
9   password: "hello@mysql", // User password
10  database: "campus_event" // Database name
11 });
12
13 // Connect to the database
14 db.connect((err) => {
15   if (err) {
16     console.error("Database␣connection␣failed:",
           err);
17   } else {
```

```
    console.log("Connected to MySQL database.");
  }
});

// Export the connection
module.exports = db;
```

Listing 11: Node.js Code for Database Connection

8.3 Authentication

Listing 12 shows the back-end code for authentication of user.

```

const jwt = require('jsonwebtoken');

// Middleware to verify JWT and authenticate
   users
const authenticate = (req, res, next) => {
  const token = req.headers.authorization?.split
     (' ')[1]; // Extract token from "Bearer <
     token>"
  // console.log(token);
  // console.log(token);
  if (!token) {
    return res.status(401).json({ message: '
       Access denied. No token provided.' });
  }

  try {
    const decoded = jwt.verify(token, 'secret');
    req.user = decoded; // Attach decoded
       payload (e.g., user ID) to the request
       object
    // console.log(req.user);
    // console.log(req.user.user_id);``
    next();
  } catch (err) {
    console.log("gudu");
    return res.status(403).json({ message: '
       Invalid or expired token.' });
```

```
  }
};

module.exports = {
  authenticate,
};
```

Listing 12: Node.js Code for authentication

8.4 HomePage

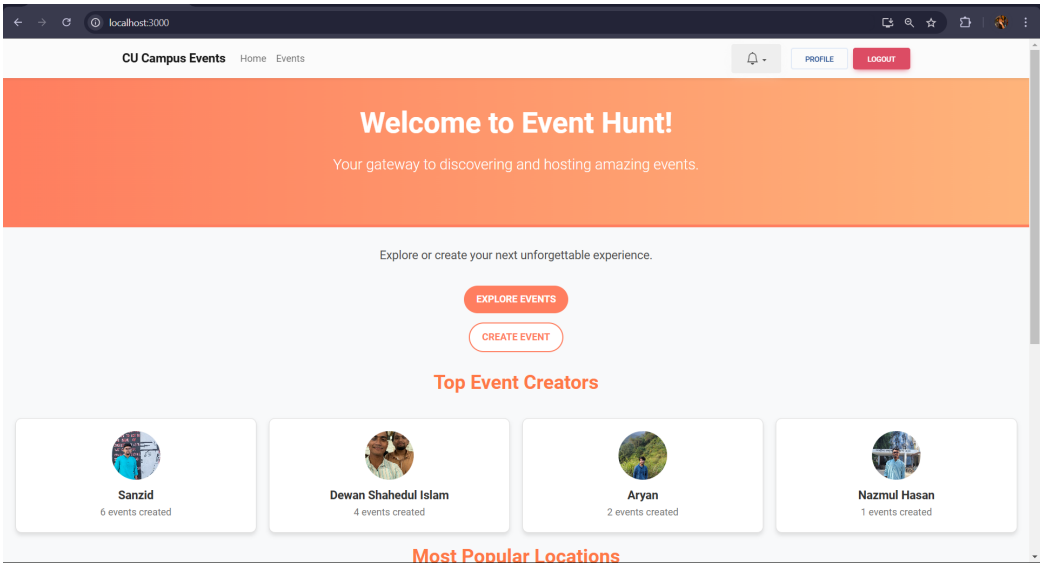

Figure 4: Home Page

Listing 13 shows the front-end code of homepage.

```
import React, { useEffect, useState } from '
    react';
import PropTypes from 'prop-types';
import { Link } from 'react-router-dom';
import axios from 'axios';
import Footer from './footer';
import '../all-css/HomePage.css';

const HomePage = ({ isLoggedIn }) => {
```

```
9   const [popularLocations, setPopularLocations]
      = useState([]);
10  const [topCreators, setTopCreators] = useState
      ([]);
11
12  // Fetch data on component mount
13  useEffect(() => {
14    const fetchPopularLocations = async () => {
15      try {
16        const response = await axios.get('http
            ://localhost:5000/locations/
            popularLocations');
17        if (response.status === 200)
            setPopularLocations(response.data);
18      } catch (error) {
19        console.error('Error fetching popular
            locations:', error.message);
20      }
21    };
22
23    const fetchTopCreators = async () => {
24      try {
25        const response = await axios.get('http
            ://localhost:5000/topList/
            getTopCreators');
26        if (response.status === 200)
            setTopCreators(response.data);
27      } catch (error) {
28        console.error('Error fetching top
            creators:', error.message);
29      }
30    };
31
32    fetchPopularLocations();
33    fetchTopCreators();
34  }, []);
35
36  return (
37    <div className="homepage-container text-
        center">
38      {/* Hero Section */}
```

```
39          <header className="hero-section">
40            <h1 className="hero-title">Welcome to
                 Event Hunt!</h1>
41            <p className="hero-subtitle">Your
                 gateway to discovering and hosting
                 amazing events.</p>
42          </header>
43
44          {/* CTA Section */}
45          <div className="cta-section">
46            {!isLoggedIn ? (
47              <div className="cta-buttons">
48                <p className="cta-text">Log in or
                     sign up to explore events
                     happening right now!</p>
49                <Link to="/login"><button className=
                     "cta-button primary">Login</
                     button></Link>
50                <Link to="/register"><button
                     className="cta-button secondary">
                     Register</button></Link>
51              </div>
52            ) : (
53              <div className="cta-buttons">
54                <p className="cta-text">Explore or
                     create your next unforgettable
                     experience.</p>
55                <Link to="/show-events"><button
                     className="cta-button primary">
                     Explore Events</button></Link>
56                <Link to="/create-event"><button
                     className="cta-button secondary">
                     Create Event</button></Link>
57              </div>
58            )}
59          </div>
60
61          {/* Top Creators Section */}
62          <section className="top-creators-section">
63            <h2 className="section-title">Top Event
                 Creators</h2>
```

```
64          <div className="creators-grid">
65            {topCreators.length > 0 ? (
66              topCreators.map((creator) => (
67                <div key={creator.user_id}
                      className="creator-card">
68                  <img
69                    src={`http://localhost:5000/${
                          creator.profile_picture}`}
70                    alt={creator.name}
71                    className="creator-profile-pic
                          "
72                  />
73                  <div className="creator-name">{
                        creator.name}</div>
74                  <div className="creator-events"
                        >{creator.count_events}
                        events created</div>
75                </div>
76              ))
77            ) : (
78              <p className="no-creators-message">
                    No top creators to display at the
                     moment.</p>
79            )}
80          </div>
81        </section>
82
83        {/* Popular Locations Section */}
84        <section className="popular-locations-
             section">
85          <h2 className="section-title">Most
               Popular Locations</h2>
86          <div className="locations-grid">
87            {popularLocations.length > 0 ? (
88              popularLocations.map((location) => (
89                <div key={location.location_id}
                      className="location-card">
90                  <h3>{location.location_name}</h3
                        >
91                  <p>{location.total_events}
                        events hosted here</p>
```

```
92                </div>
93              ))
94            ) : (
95              <p className="no-locations-message">
                  No popular locations to display
                  at the moment.</p>
96            )}
97          </div>
98        </section>
99
100       <br />
101       <Footer />
102     </div>
103   );
104 };
105
106 // Prop type validation
107 HomePage.propTypes = {
108   isLoggedIn: PropTypes.bool.isRequired,
109 };
110
111 export default HomePage;
```

Listing 13: React Code for homepage

8.5 Profile Page

Listing 14 shows the front-end code of profile_page.

```
1
2  import React, { useState, useEffect } from "
     react";
3  import axios from "axios";
4  import { Link, useNavigate } from "react-router-
     dom";
5  import ProfileInfo from "./ProfileInfo";
6
7  import ToggleButtons from "./ToggleButtons";
8  import EventCard from "./EventCard";
9  import EventCardOfRegister from "../showEvents/
     EventCard";
```

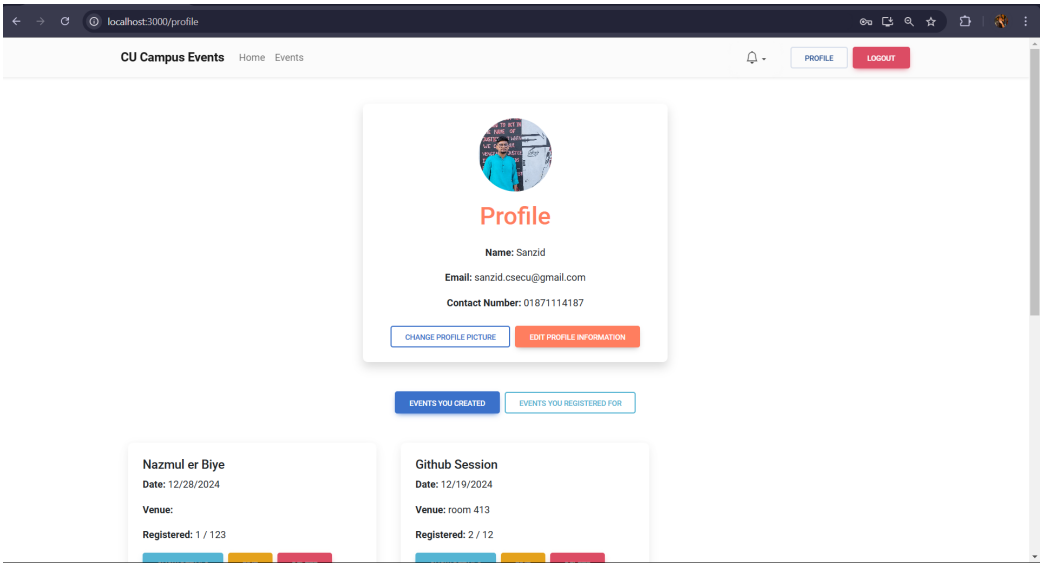

Figure 5: Profile Page

```
10 import { fetchEvents, deleteEvent } from "../../
     services/eventService";
11 import { fetchVenues } from "../../services/
     venueService";
12
13 import EventDetailsModal from "../showEvents/
     EventDetailsModal";
14 import EditEventModal from "./EditEventModal";
15 import ShowRegisteredUsersModal from "./
     ShowRegisteredUsersModal";
16 import ShowReviewsMOdal from "./ShowReviewsModal
     ";
17 import "../../all-css/Profile.css"
18 import RegisteredEventCard from "./
     RegisteredEventCard";
19 const Profile = () => {
20   const [userProfile, setUserProfile] = useState
       (null);
21   const [events, setEvents] = useState([]);
22   const [registeredEvents, setRegisteredEvents]
       = useState([]);
23   const [showCreatedEvents, setShowCreatedEvents
       ] = useState(true);
```

```
24    const [registeredUsers, setRegisteredUsers] =
          useState([]);
25    const [eventReviews, setEventReviews] =
          useState([]);
26    const [showReviewsModal , setShowReviewsModal]
          = useState(false);
27    const [eventReviewsStatus,
          setEventReviewsStatus] = useState(false);
28    const [showRegisteredUsersModal,
          setShowRegisteredUsersModal] = useState(
          false);
29    const [registered_users_status,
          set_registered_users_status] = useState('')
          ;
30    const navigate = useNavigate();
31    ////////
32
33    const [venues, setVenues] = useState([]);
34    const [selectedEvent, setSelectedEvent] =
          useState(null);
35    const [editEvent, setEditEvent] = useState(
          null);
36
37    ///////
38    useEffect(() => {
39      const token = localStorage.getItem("token");
40  //////////////
41  loadEvents();
42  loadVenues();
43  ////////////
44      if (!token) {
45        navigate("/login");
46        return;
47      }
48
49      axios
50        .get("http://localhost:5000/auth/profile",
              {
51          headers: { Authorization: `Bearer ${
              token}` },
52        })
```

```
53        .then((response) => {
54          setUserProfile(response.data);
55          fetchCreatedEvents(response.data.user_id
              );
56          fetchRegisteredEvents(response.data.
              user_id);
57        })
58        .catch((error) => {
59          console.error("Error fetching profile:",
                error);
60          navigate("/login");
61        });
62    }, [navigate]);
63
64    const fetchCreatedEvents = (userId) => {
65      axios
66        .get(`http://localhost:5000/events/user/${
            userId}`)
67        .then((response) => setEvents(response.
            data))
68        .catch((error) => console.error("Error
            fetching created events:", error));
69    };
70
71    const fetchRegisteredEvents = (userId) => {
72      axios
73        .get(`http://localhost:5000/events/
            registered/${userId}`)
74        .then((response) => setRegisteredEvents(
            response.data))
75        .catch((error) => console.error("Error
            fetching registered events:", error));
76    };
77    const fetchRegisteredUsers = async (eventId)
        => {
78      try {
79        const response = await axios.get(`http://
            localhost:5000/events/registered_users/
            ${eventId}`);
80        setRegisteredUsers(response.data);
81        set_registered_users_status(response.
```

```
        status);
      setShowRegisteredUsersModal(true);
      console.log(response);
      console.log(eventId);
    } catch (error) {
      console.error("Error fetching registered
        users:", error);
      alert("Failed to fetch registered users.")
        ;
    }
  };
  const fetchEventReviews = async(eventId) =>{
    try{
      const response = await axios.get(`http://
        localhost:5000/reviews/${eventId}/
        allReviews`);
      setEventReviews(response.data);
      setEventReviewsStatus(response.status);
      setShowReviewsModal(true);
    }
    catch(error){
      console.error("Error fetching event
        reviews:", error);
      alert("Failed to fetch Reviews");
    }
  }


  const handleEditEvent = (eventId) => {

  };

  //////////////
  const loadEvents = async () => {
    const token = localStorage.getItem("token");
    axios
    .get("http://localhost:5000/auth/profile", {
        headers: { Authorization: `Bearer ${
          token}` },
    })
      .then((response) => {
```

```
116           fetchCreatedEvents(response.data.user_id
                  );
117         })
118         .catch((error) => {
119           console.error("Error␣fetching␣profile:",
                  error);
120           navigate("/login");
121         });
122
123   };
124
125   const loadVenues = async () => {
126     try {
127       const data = await fetchVenues();
128       setVenues(data);
129     } catch (error) {
130       console.error(error);
131     }
132   };
133   const handleDelete = async (eventId) => {
134     if (window.confirm("Are␣you␣sure␣you␣want␣to
          ␣delete␣this␣event?")) {
135       try {
136         await deleteEvent(eventId);
137         alert("Event␣deleted␣successfully!");
138         loadEvents();
139       } catch (error) {
140         alert("Failed␣to␣delete␣event.");
141       }
142     }
143   };
144   ////////////
145
146   if (!userProfile) return <div>Loading...</div
          >;
147
148   return (
149     <div className="container␣mt-5">
150       <ProfileInfo userProfile={userProfile} />
151       <ToggleButtons onToggle={
              setShowCreatedEvents} showCreatedEvents
```

```
          ={showCreatedEvents} />
      {showCreatedEvents ? (
        <div className="container␣mt-5">

      {events.length === 0 ? (
        <>
        <h2 className="text-center␣mt-5">You
           have no events.</h2>
        <div className="text-center␣my-5">
        <Link to= '/create-event'>
            <button className="btn␣btn-danger">
               Create now</button>
        </Link>
        </div>
        </>
      ) : (
        <div className="row">
          {events.map((event) => (
            <EventCard
              key={event.event_id}
              event={event}
              venues={venues}
              onShowDetails={setSelectedEvent}
              onEdit={setEditEvent}
              onDelete={handleDelete}
              onShowRegisteredUsers={
                 fetchRegisteredUsers}
              onShowReviews={fetchEventReviews}

            />
          ))}
        </div>
      )}

      {selectedEvent && (
        <EventDetailsModal
          event={selectedEvent}
          venues={venues}
          onClose={() => setSelectedEvent(null)}
        />
      )}
```

```
189
190         {editEvent && (
191           <EditEventModal
192             event={editEvent}
193             onClose={() => setEditEvent(null)}
194             onSave={loadEvents}
195           />
196         )}
197       </div>
198         ) : (
199           <div className="row mt-5">
200             {/* {registeredEvents.map((event) => (
201               <RegisteredEventCard
202                 key={event.event_id}
203                 event={event}
204                 venues={venues}
205                 id = {userProfile.user_id}
206                 onShowDetails={setSelectedEvent}
207                 token = {localStorage.getItem('
                      token')}
208               />
209             ))} */}
210             {registeredEvents.length === 0 ? (
211       <p>You haven't registered for any events yet
            .</p>
212     ) : (
213       registeredEvents.map((event) => (
214         <RegisteredEventCard
215           key={event.event_id}
216           event={event}
217           venues={venues}
218           id={userProfile.user_id}
219           onShowDetails={setSelectedEvent}
220           token={localStorage.getItem('token')}
221         />
222       ))
223     )}
224           </div>
225         )}
226         {selectedEvent && (
227           <EventDetailsModal
```

```
228             event={selectedEvent}
229             venues={venues}
230             onClose={() => setSelectedEvent(null)}
231           />
232         )}
233         {showRegisteredUsersModal && (
234   <ShowRegisteredUsersModal
235     users={registeredUsers}
236     onClose={() => setShowRegisteredUsersModal(
          false)}
237     state = {registered_users_status}
238   />
239  )}
240         {showReviewsModal && (
241   <ShowReviewsMOdal
242     users={eventReviews}
243     onClose={() => setShowReviewsModal(false)}
244     state = {eventReviewsStatus}
245   />
246  )}
247
248
249     </div>
250   );
251 };
252
253 export default Profile;
```

Listing 14: React Code for profile_Page

9 Validation

9.1 Website Review With Q&A

1. Does the site have all required meta descriptions?
   Ans: Yes.
2. Does the site have all required page titles?
   Ans: Yes.
3. Are the images on the site too large?
   Ans: No.
4. Is the site plagued by too many links?
   Ans: No.
5. Does the website use a responsive design?
   Ans: Yes.
6. Is your reporting data accurate?
   Ans: Yes.
7. Are your URLs too long?
   Ans: No.
8. Have images on the site been optimized?
   Ans: Yes.
9. How deeply has content on the site been optimized?
   Ans: Medium level.

Our system is user-friendly, and users have reported high satisfaction with its functionality. The **Campus Event Management System** allows students and teachers to easily create, organize, and join events on campus.

- Users can explore events based on categories like sports, adventure, workshops, and others from the homepage.
- Users can create events with relevant details like event title, description, location, category, and schedule.
- Users can register for events directly from the platform.
- Organizers can monitor the event participants.

Overall, our system provides an efficient platform to connect the campus community and improve event organization and participation.

9.2 User Manual for Users

Follow these steps to use the Campus Event Management System:

1. **Accessing the Website:**
   - Open the website from any browser. The homepage will display a list of top locations and creators and navigation options such as *Login/Explore Events*, *About*, and *Contact Us*.
2. **User Registration and Login:**
   - New users must click on the **Register** button to create an account.
   - Returning users can log in using their credentials by clicking on the **Login** button.
3. **Exploring Events:**
   - Logged-in users can browse events based on categories or use the search bar to find specific events.
   - Users can view details such as the event title, date, location, description, and number of participants.
4. **Registering for Events:**
   - To join an event, click the **Register** button on the event details page.
   - A confirmation message will appear after successful registration.
5. **Creating Events:**
   - Users can create events by clicking on the **Create Event** button.
   - Fill in the necessary details, such as:
     * **Event Title**
     * **Description**
     * **Category** (Sports, Workshop, Adventure, etc.)
     * **Location**
     * **Event Date and Time**
   - Submit the form to successfully create the event.
6. **Managing Events:**
   - Users can view all their events on the **Profile**.

- They can:
  * Edit event details.
  * Delete events.
  * Monitor registered participants.

7. **Logging Out:**
   - Users can log out from the website by clicking the **Log Out** button in the navigation bar.

9.3 User Feedback

Users, including both students and faculty, have provided positive feedback on the system's usability and functionality. The intuitive interface and clear navigation make event management simple and effective. The Director of Campus Activities stated:

> "The Campus Event Management System has significantly streamlined event organization and participation on campus. It is an excellent tool for building connections and fostering engagement."

10 Software Deployment

Deployment of websites is the process of getting a web application or website online and accessible to end users. It involves transferring the code, files, and data from a development environment to a production environment where all users can access it. The deployment procedure varies depending on the type of web application, developer framework, and hosting platform. Below, we outline the deployment steps for our **Campus Event Management System**:

1. **Determining Hosting Environment:**
   - A suitable hosting environment must be selected to meet the needs of the application, including compatibility with the programming language (Node.js), database (MySQL), and required server software.
2. **Obtaining the Web Application:**
   - The application can be downloaded from our GitHub repository:
     https://github.com/Sanzidislam/Event_Management_System_Project.git
3. **Preparing the Hosting Environment:**
   - Configure the hosting environment to support:
     * Node.js runtime.
     * MySQL database.
     * Required environment variables for database connections and security settings.
4. **Uploading the Web Application:**
   - Upload all project files, including React frontend code and Node.js backend code, to the hosting server. This can be done using:
     * **File Transfer Protocol (FTP)** tools.
     * **Web-based control panels** provided by the hosting service.
5. **Configuring the Web Application:**
   - Update the configuration files to include the necessary settings:
     * Set up database connections (MySQL credentials).

    - Define API routes for the backend.
    - Configure environment variables such as:
        - Database host, user, password, and name.
        - Port for the server.
        - API keys or security tokens, if applicable.

6. **Testing the Web Application:**
    - Verify that the website functions as expected after deployment. This includes:
        - Testing navigation and interactivity of the React frontend.
        - Testing API calls and data transactions via the Node.js backend.
        - Ensuring the MySQL database is operational and accessible.
        - Checking for errors in browser console logs and backend server logs.

7. **Deploying Updates:**
    - Periodically update the web application to maintain security and functionality:
        - Fetch the latest code from the GitHub repository.
        - Test new updates in a staging environment before deploying to production.
        - Redeploy the updated code to the hosting server.

If any problem occurs during deployment or usage, users are encouraged to contact our development team for assistance.

11 Conclusion and Future Work

The **Campus Event Management System** provides an efficient and user-friendly way to manage campus events, registrations, and users (students and teachers). The system simplifies event creation, user registration, and event participation while ensuring an organized platform for administrators to oversee events. Through the system, users can easily browse events, register for them, and even create new events as per their interests. The system is designed to enhance engagement and streamline event management in a campus environment.

The platform has achieved significant milestones:

- It allows users to register for events of their interest and receive updates on upcoming events.
- Event organizers (users) can efficiently create, edit, and manage events.

Despite these achievements, we recognize certain limitations and areas for improvement, which we aim to address in future iterations of the system:

- **Mobile Application:** Our system currently operates as a web-based platform. Developing a dedicated mobile application would provide users with enhanced accessibility and convenience, allowing them to manage events on the go.
- **Automated Notifications:** Presently, event notifications or reminders are not fully automated. In future updates, we plan to integrate an automated email notification system to remind users about event schedules and updates.
- **Enhanced Validation:** Users can currently register for overlapping events without restrictions. Introducing validation to prevent conflicting event registrations will improve system accuracy and usability.
- **Messaging System** Currently there is no messaging or commenting option for users to connect or do a query. Introducing a messaging system will increase the connectivity.

With our in-depth knowledge and continued technical growth, we are confident that we can address these limitations and enhance the system's functionality. Future iterations of our system will focus on im-

proving user experience, incorporating new features, and expanding platform compatibility.

In conclusion, our **Campus Event Management System** serves as a foundation for fostering connectivity, collaboration, and organization within a campus community. we aim to continue its evolution to meet the changing demands of users and maintain its relevance in event management platforms.

References

[1] Draw.io. https://en.wiki.bluespice.com/wiki/Manual:Extension/DrawioEditor.

[2] Entity-relationship model. https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model.

[3] Github. https://en.wikipedia.org/wiki/GitHub.

[4] Latex. https://en.wikipedia.org/wiki/LaTeX.

[5] Meetup. https://en.wikipedia.org/wiki/Meetup.

[6] Mysql. https://en.wikipedia.org/wiki/MySQL_Workbench.

[7] Mysql. https://en.wikipedia.org/wiki/MySQL.

[8] Node.js. https://en.wikipedia.org/wiki/Node.js.

[9] Overleaf. https://en.wikipedia.org/wiki/Overleaf.

[10] React. https://react.dev/.

[11] Visual studio code. https://en.wikipedia.org/wiki/Visual_Studio_Code.

[12] OpenAI. Chatgpt: Ai language model, 2024. Accessed: December 11, 2024.