CLUB's EVENT and BUDGET MANAGEMENT SYSTEM- https://dbms-nextjs.vercel.app/

1. Project Overview

The project aims to create a comprehensive database system for managing various entities related to a university club's event operations. The system is designed to handle entities such as Students, Clubs, and Admins, each with specific roles and interactions. Students can RSVP for events, Clubs can request event scheduling and budget allocations, and Admins oversee the approval process and budget distribution. The system is built using Next.js for the frontend, React components for user interface design, and Supabase for backend services.

2. Database Design

Entities and Their Attributes

The database design involves several key entities, each with specific attributes. Here's a breakdown of these entities:

1. **Students:**
   - **Attributes:** RollNumber (Primary Key), Name, Email
   - Students log in using their college email IDs (ending with `@snu.edu.in`) and their college roll numbers as passwords.
2. **Clubs:**
   - **Attributes:** ClubID (Primary Key), ClubName, Email, Password
   - Clubs log in using their club email IDs (also ending with `@snu.edu.in`). Passwords are stored securely in the database.
3. **Admins:**
   - **Attributes:** AdminID (Primary Key), Email, Password
   - Admins have their own email IDs and passwords for authentication.
4. **Events:**
   - **Attributes:** EventID (Primary Key), ClubID (Foreign Key referencing Clubs), EventName, Date, Venue, Status, RequestedBudget, AllocatedBudget, Reason
   - Events are associated with specific clubs. The status can be 'Scheduled,' 'Rejected,' or 'Pending.'
5. **BudgetRequests:**
   - **Attributes:** RequestID (Primary Key), EventID (Foreign Key referencing Events), RequestedAmount, AllocatedAmount, Reason, Status, AdminFeedback
   - Budget requests are submitted by clubs for event scheduling and additional funding.
   - Status for a request can be 'Approved', 'Rejected' or 'Pending'.
6. **RSVPs:**
   - **Attributes:** RSVPID (Auto-generated), StudentEmail (Foreign Key referencing Students), EventID (Foreign Key referencing Events)

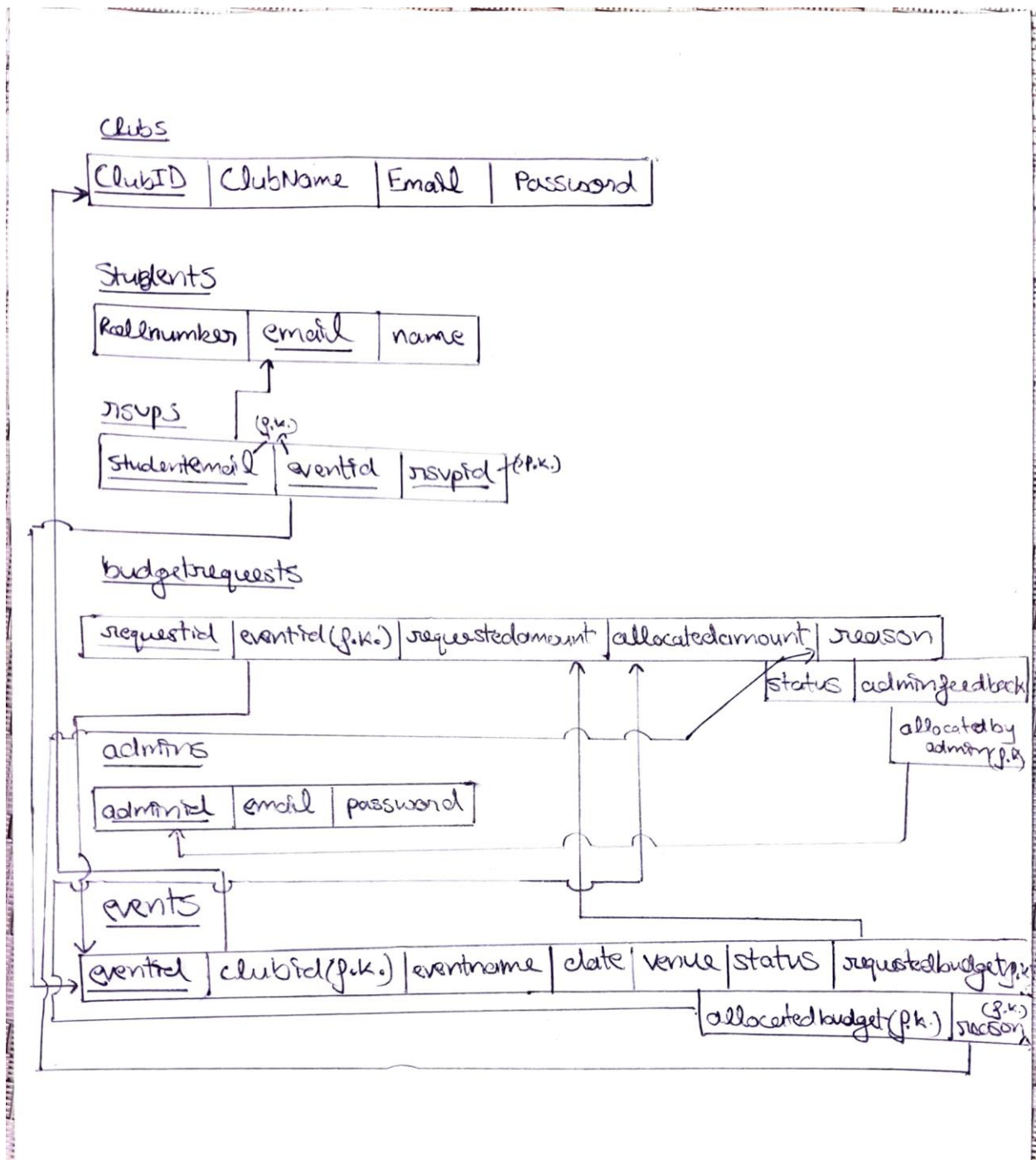o   Students can RSVP for upcoming events.

Relationships

- **Students and Events:**
  - o   Students can RSVP for events. The `RSVPs` table captures this relationship.
- **Clubs and Events:**
  - o   Clubs schedule events. The `Events` table associates events with specific clubs.
- **Admins and Budget Requests:**
  - o   Admins review and approve/reject budget requests. The `BudgetRequests` table links requests to admins.
- **Events and Budget Requests:**
  - o   Budget requests are associated with specific events. The `BudgetRequests` table references the `Events` table.
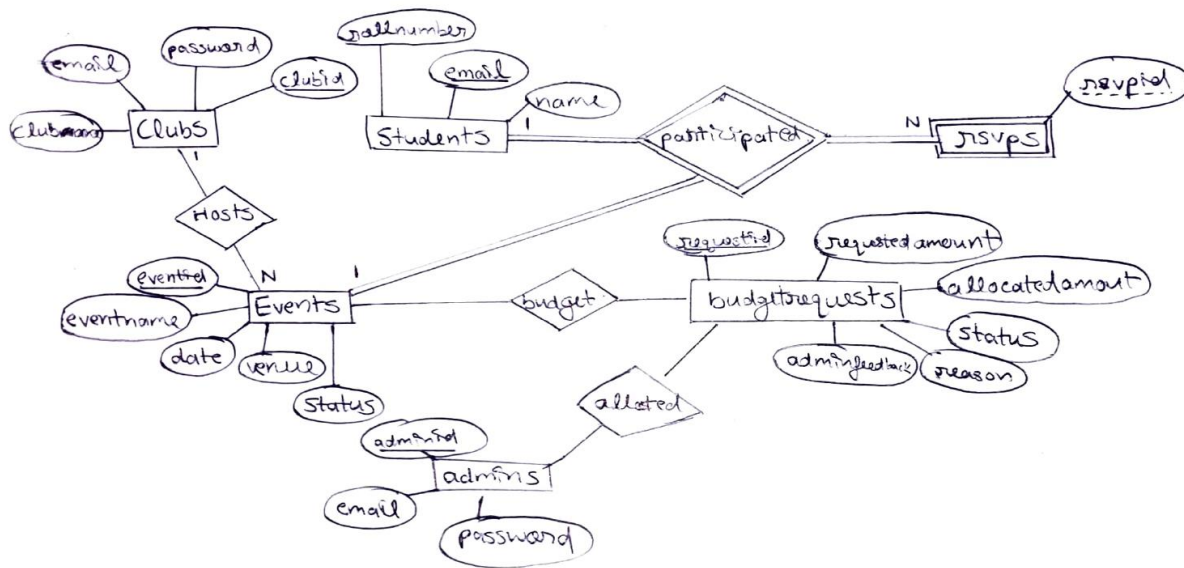
Database Schema-
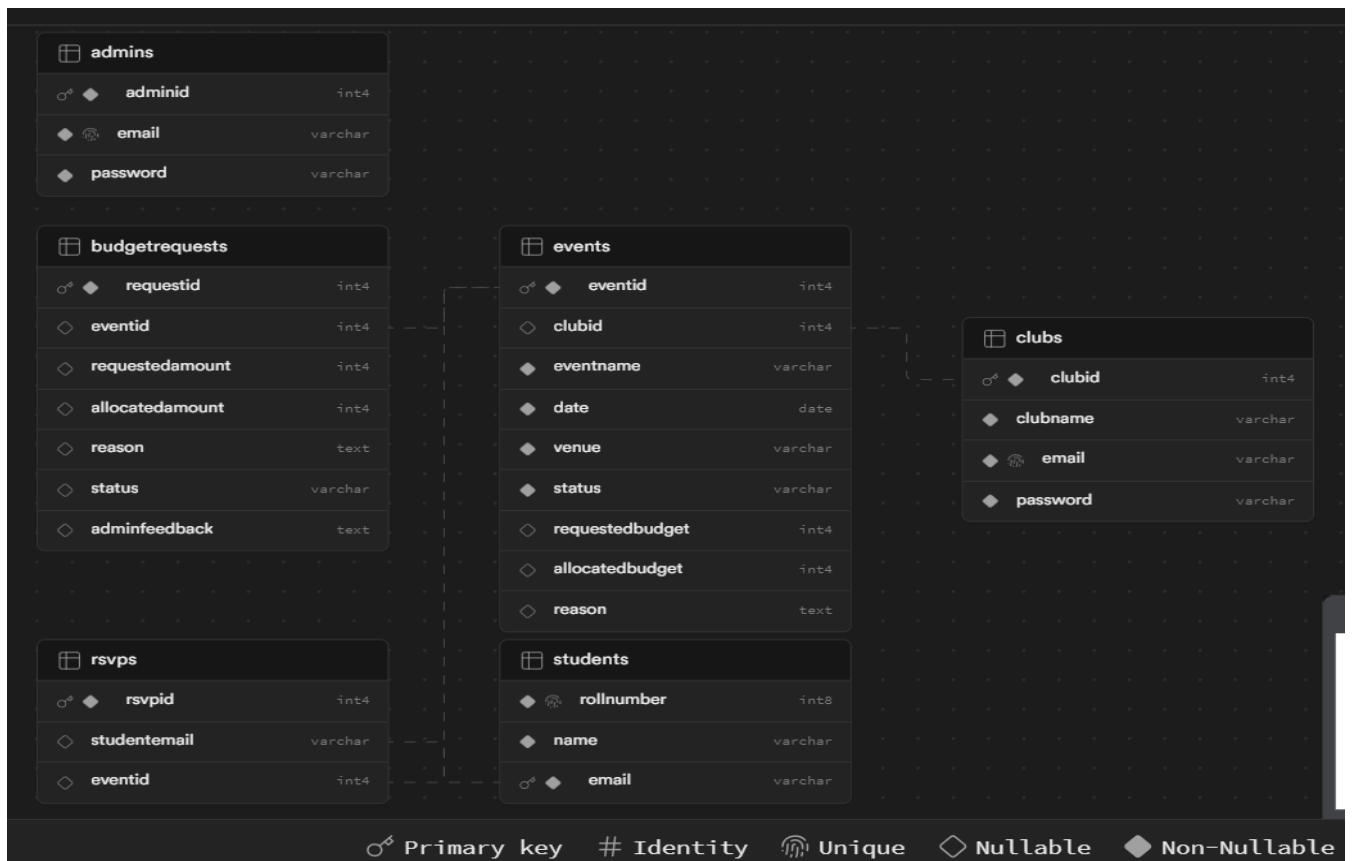
This schema captures the relationships and constraints necessary for our project.

Relational Model-

Clubs

ClubID	ClubName	Email	Password

Students

Rollnumber	email	name

rsvps

Studentemail	eventid	rsvpid (p.k.)

(p.k.)

budgetrequests

requestid	eventid (f.k.)	requestedamount	allocatedamount	reason	status	adminfeedback	allocatedby admin(f.k)

admins

adminid	email	password

events

eventid	clubid (f.k.)	eventname	date	venue	status	requestedbudget	allocatedbudget (p.k.)	reason (f.k.)

ER Model-

Functional Dependencies and Normalisation

The functional dependencies for each table in our database schema indicate that the database is normalized to at least the Third Normal Form (3NF). Here's a breakdown of how each table adheres to the normalization rules:

1. **Students Table**:
   - **Functional Dependencies**:
     - Email → RollNumber, Name
     - RollNumber → Email, Name
   - **Normalization**: Each student has a unique email and roll number, and these attributes are functionally dependent on each other. This eliminates redundancy and ensures that all non-key attributes are fully functionally dependent on the primary key (Email).

2. **Clubs Table**:
   - **Functional Dependencies**:
     - ClubID → ClubName, Email, Password
     - Email → ClubID, ClubName, Password

- o **Normalization**: The `ClubID` uniquely identifies each club, and all other attributes in the table are functionally dependent on it. There are no partial or transitive dependencies, which satisfies 2NF and 3NF.

3. **Admins Table**:
   - o **Functional Dependencies**:
     - AdminID → Email, Password
     - Email → AdminID, Password
   - o **Normalization**: Like the Clubs table, the `AdminID` serves as the primary key, and all attributes are dependent on it, adhering to 3NF.

4. **Events Table**:
   - o **Functional Dependencies**:
     - EventID → ClubID, EventName, Date, Venue, Status, RequestedBudget, AllocatedBudget
     - ClubID, Date → EventID, EventName, Venue, Status, RequestedBudget, AllocatedBudget
     - Eventname, date, clubid → Eventid
   - o **Normalization**: The `EventID` is the primary key, and all event details depend on it. The composite dependency on `ClubID` and `Date` ensures that an event is uniquely identified by the club and the date, which is a business rule enforced by the schema.

5. **BudgetRequests Table**:
   - o **Functional Dependencies**:
     - RequestID → EventID, RequestedAmount, AllocatedAmount, Reason
   - o **Normalization**: The `RequestID` is the primary key, and all budget request details are dependent on it, satisfying 3NF.

6. **RSVPs Table**:
   - o **Functional Dependencies**:
     - RSVPID → StudentEmail, EventID
     - StudentEmail, EventID → RSVPID
   - o **Normalization**: The `RSVPID` is the primary key, and there are no transitive dependencies, which complies with 3NF.   In summary, the database schema adheres to the rules of 3NF because:

- All tables have a primary key that uniquely identifies their records.
- There are no partial dependencies; non-key attributes are fully functionally dependent on the primary key.
- There are no transitive dependencies; non-key attributes are only dependent on the primary key and not on other non-key attributes.

This level of normalization helps prevent update, insert, and delete anomalies, and ensures data integrity within the database.

Normalisation

1. **First Normal Form (1NF)**:
   - **How**: Each table has a primary key: `RollNumber` for `Students`, `ClubID` for `Clubs`, `AdminID` for `Admins`, `EventID` for `Events`, `RequestID` for `BudgetRequests`, `RSVPID` for `RSVPs`, `RejectedRequestID` for `RejectedRequests`. The values in each column are atomic (no repeating groups or arrays).
   - **Why**: Ensures that each table represent a relation and has a unique identifier.

2. **Second Normal Form (2NF)**:
   - **How**: All tables are already in 1NF, and those with composite primary keys (`RSVPs`, `BudgetRequests`, `RejectedRequests`) do not have any partial dependency on a subset of the key.
   - **Why**: Eliminates redundancy and ensures that each attribute is fully functionally dependent on the primary key.

3. **Third Normal Form (3NF)**:
   - **How**: In addition to being in 2NF, all attributes in each table are directly dependent on the primary key and not on any other non-prime attribute (no transitive dependencies). For example, `Email` in the `Students` table is dependent on `RollNumber` (the primary key), and not on any other attribute.
   - **Why**: Prevents data anomalies and ensures that non-key attributes do not depend on other non-key attributes.

The schema is well-structured to avoid common anomalies and redundancy, which I think is consistent with 3NF.

4. Frontend Design and User Interaction

Overview

The frontend of the club management system is designed to provide a seamless and intuitive user experience. It serves as the interface between the users (Students, Clubs, Admins) and the backend database, facilitating various operations such as event RSVPs, event scheduling, budget requests, and administrative approvals.

Design Considerations

- **Responsive Design:** Ensures compatibility across various devices and screen sizes.
- **User-Friendly Navigation:** Clear and concise menus guide users through different functionalities.
- **Accessibility:** Adheres to best practices for accessibility to accommodate all users.

User Types and Functionalities

1. **Students:**
   - **Login:** Students log in using their college email ID and roll number.
   - **Event Browsing:** Upcoming events are displayed on the left side of the dashboard.
   - **RSVP:** Students can RSVP for events by clicking the RSVP button beneath each event listing.
2. **Clubs:**
   - **Login:** Clubs log in using their club email ID and password.
   - **Event Management:** Scheduled events and RSVP counts are visible on the left side of the dashboard.
   - **Budget Management:** Clubs can request new events, request reimbursements, and check the status of requests through buttons on the right side.
3. **Admins:**
   - **Login:** Admins log in using their email ID and password.
   - **Request Review:** Admins can view pending requests and review approved requests, with options to approve or reject and allocate budgets.

User Interface Components

- **Home Page:** Offers options to select user type (Student, Club, Admin).
- **Login Page:** Redirects to the respective login page based on the user type selected.

- **Dashboard:** Customized dashboards for each user type with relevant functionalities and information.

Interaction Flow

- **Case 1: Student**
  - After login, students are presented with a dashboard to view and RSVP for events.
- **Case 2: Clubs**
  - Club members have a dashboard to manage events and budget requests.
- **Case 3: Admin**
  - Admins have access to a dashboard for approving requests and managing events.

Technologies Used

- **Next.js:** A React framework used for building the user interface.
- **React Components:** Reusable components that provide a dynamic and interactive user experience.
- **CSS Frameworks:** Used for styling and ensuring a consistent look and feel.

Security Measures

- **Authentication:** Secure login process with validation of email formats and password storage.
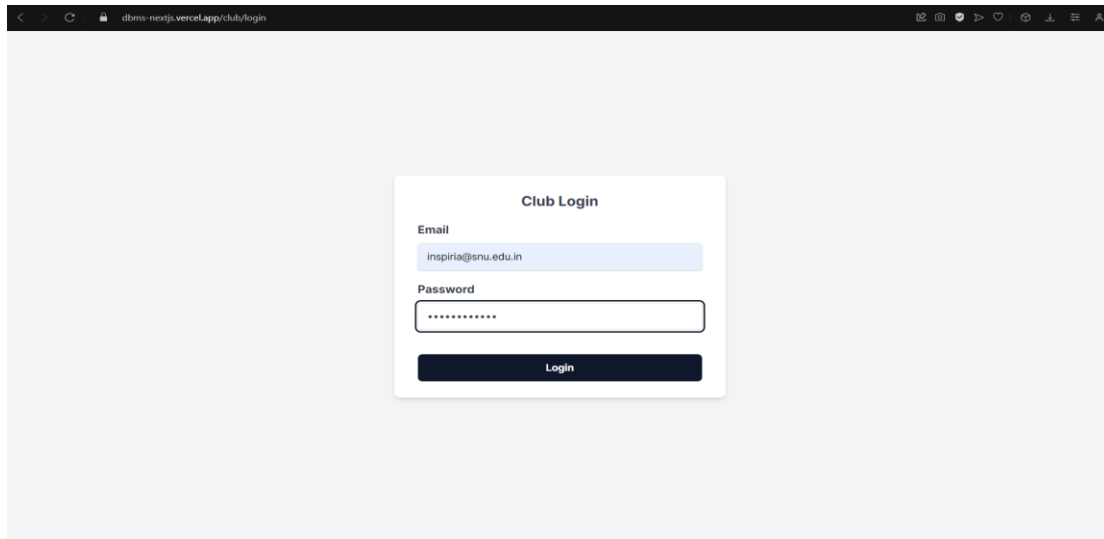
Frontend Tutorial

Main Page

Club Login

Club Main Page

Club registering New Event.
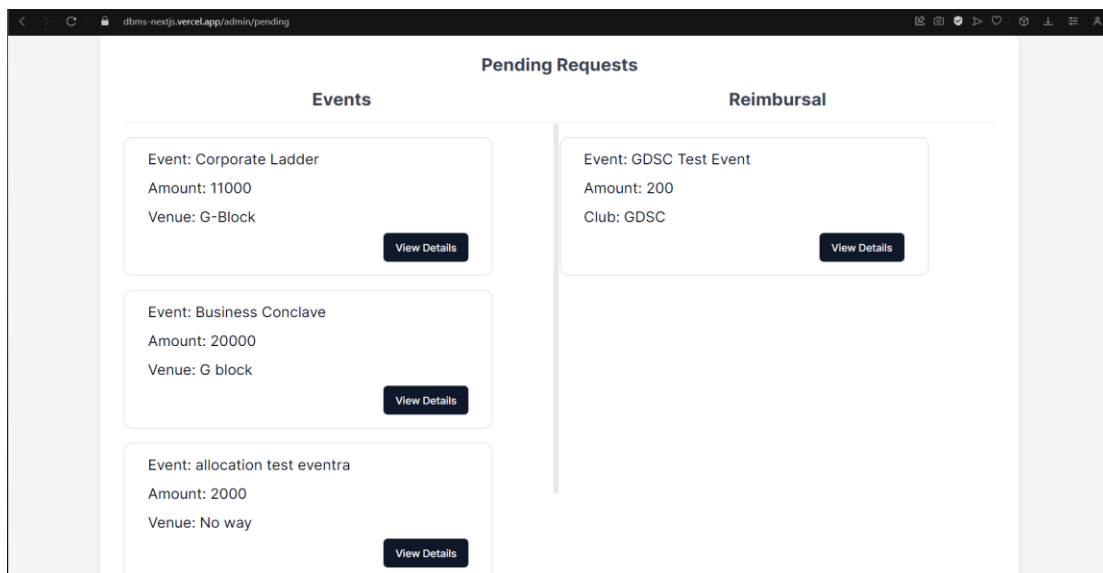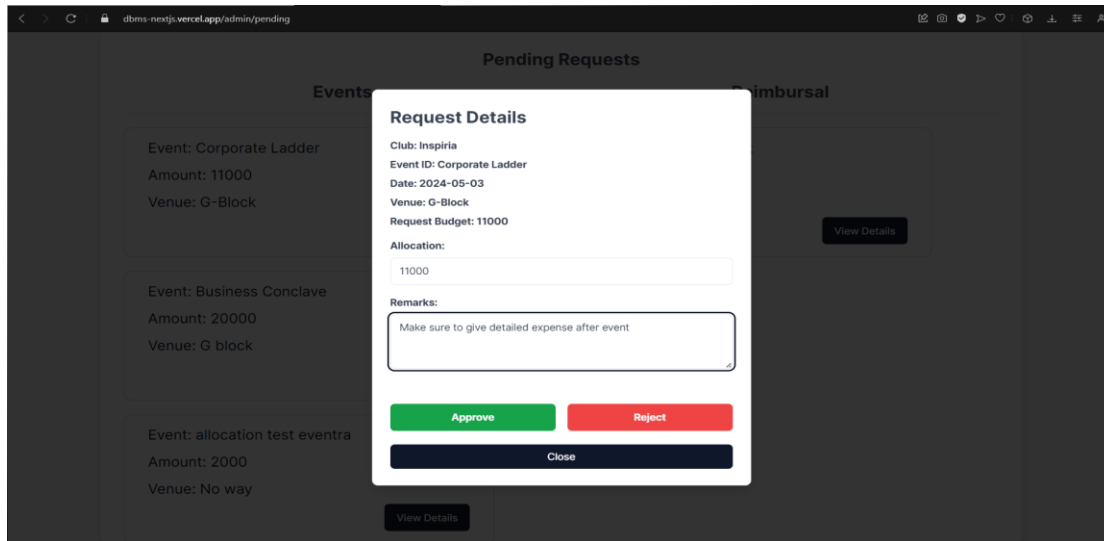

Admin Login

Admin Main Page
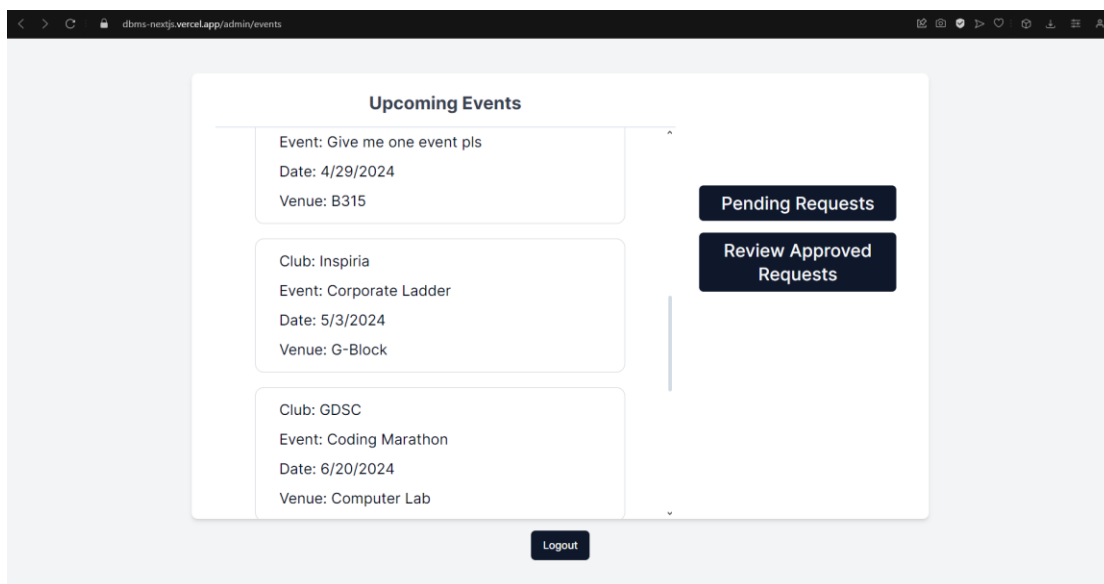

Admin Pending Request Page.

Admin Approval for Event Registration.

Event is Scheduled after Approval.

Event showing Scheduled on Club Request Page.

Student Login

Student Main page where they can RSVP.
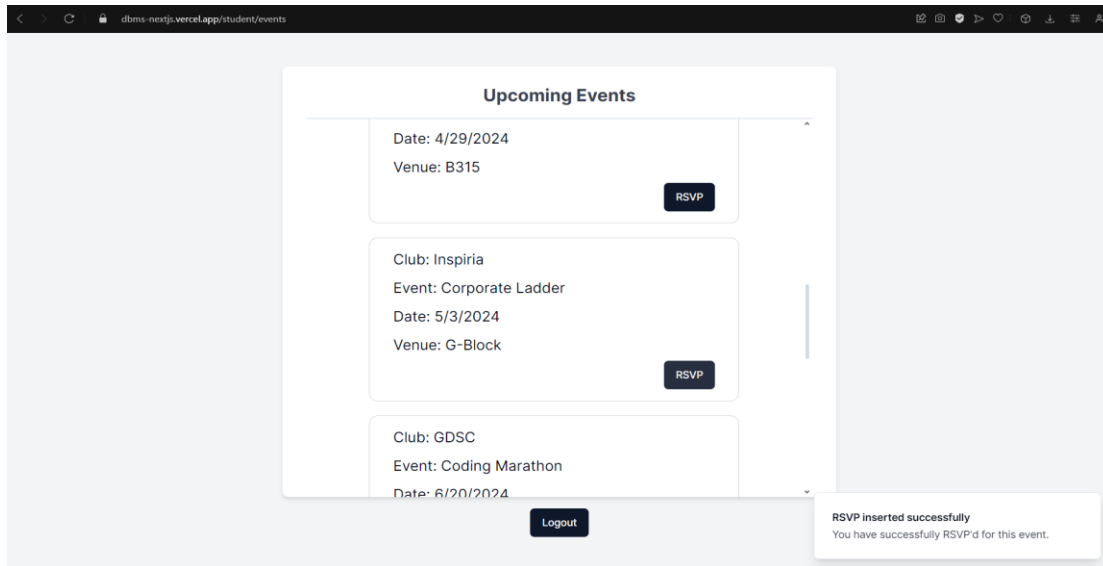
Number of RSVPs shows on club page after someone RSVP.
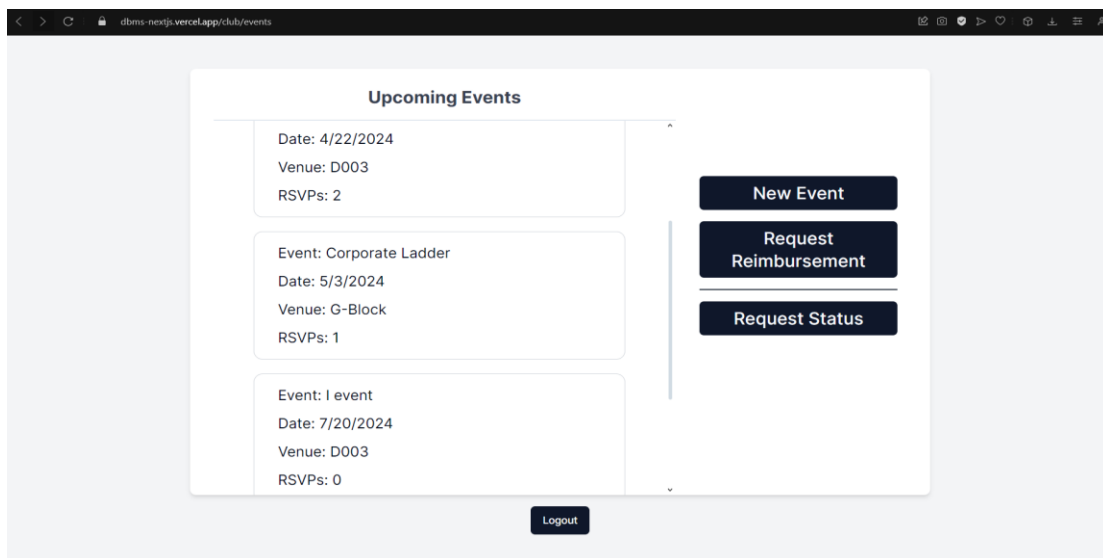
Club Request Reimbursement.

Reimbursement Status pending before admin Approval.

Admin Approving Reimbursement.
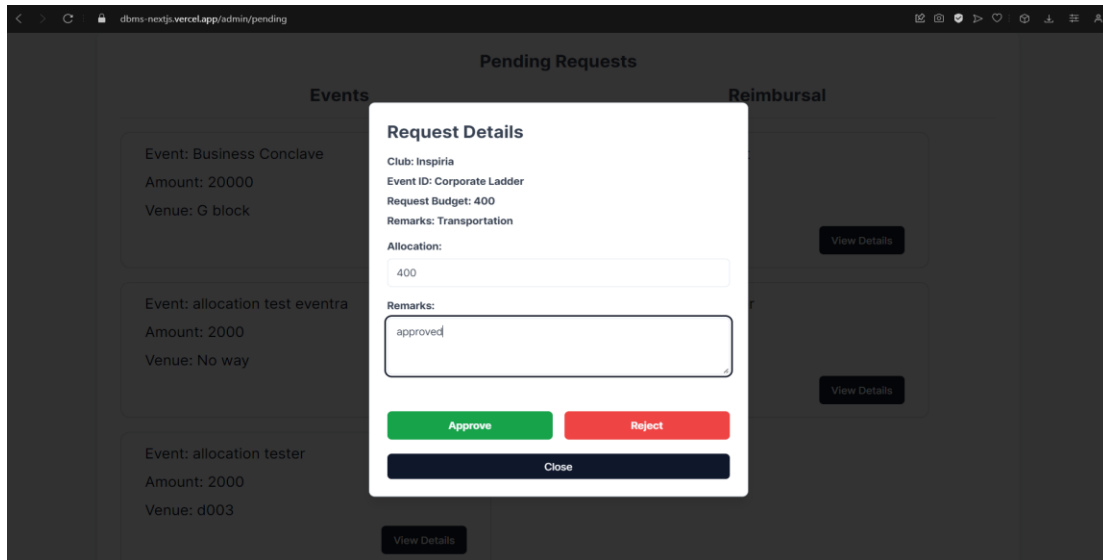
Reimbursement Status Approved on Club Page.
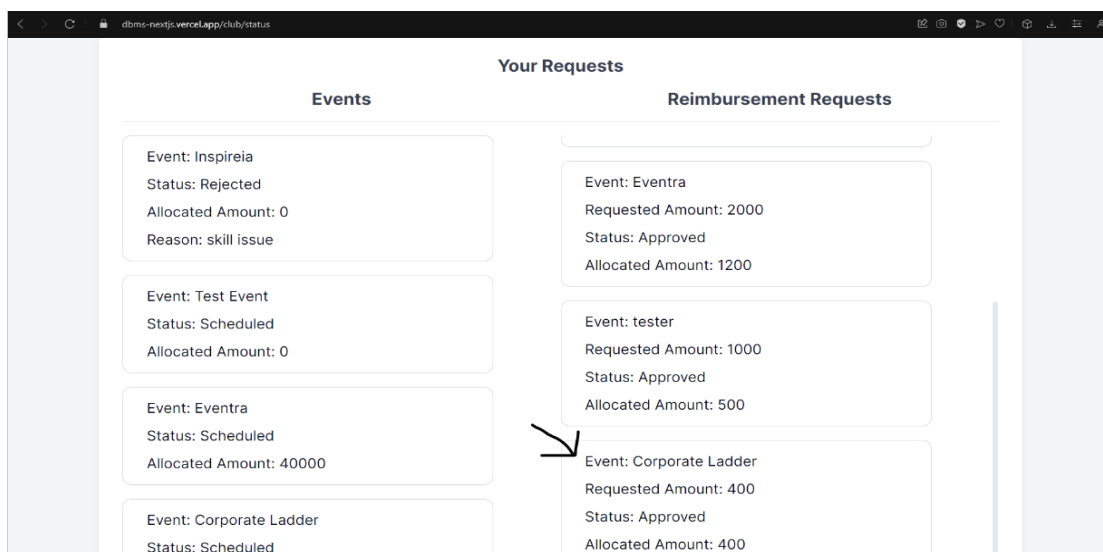
Admin Rejection.
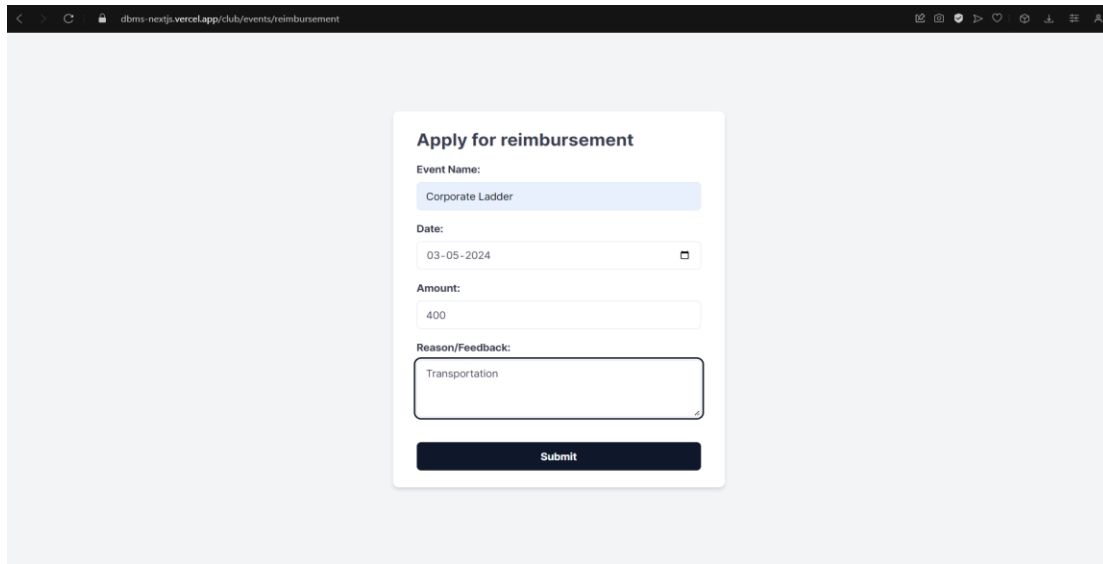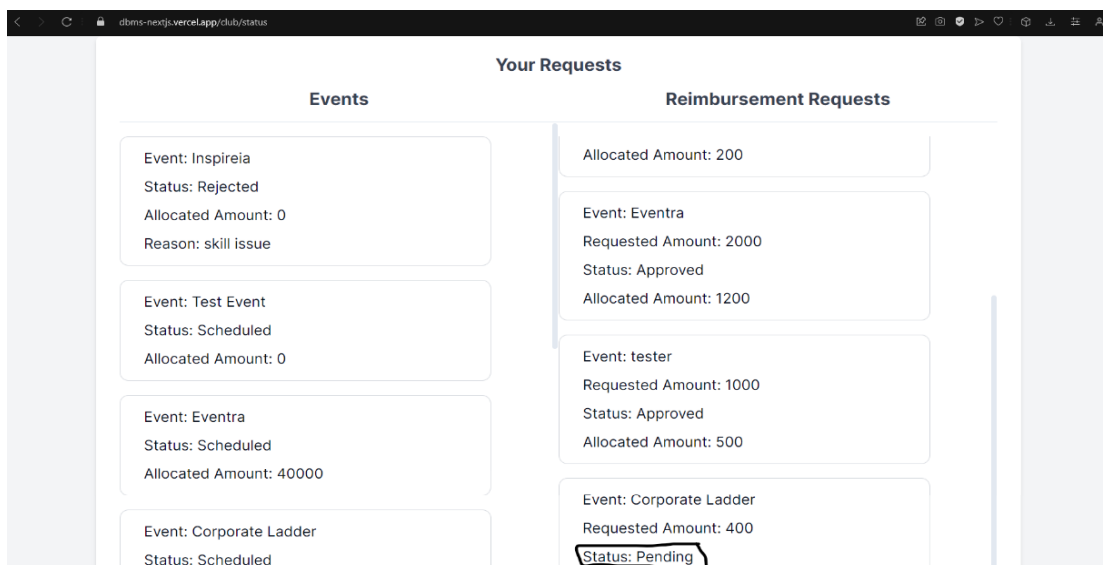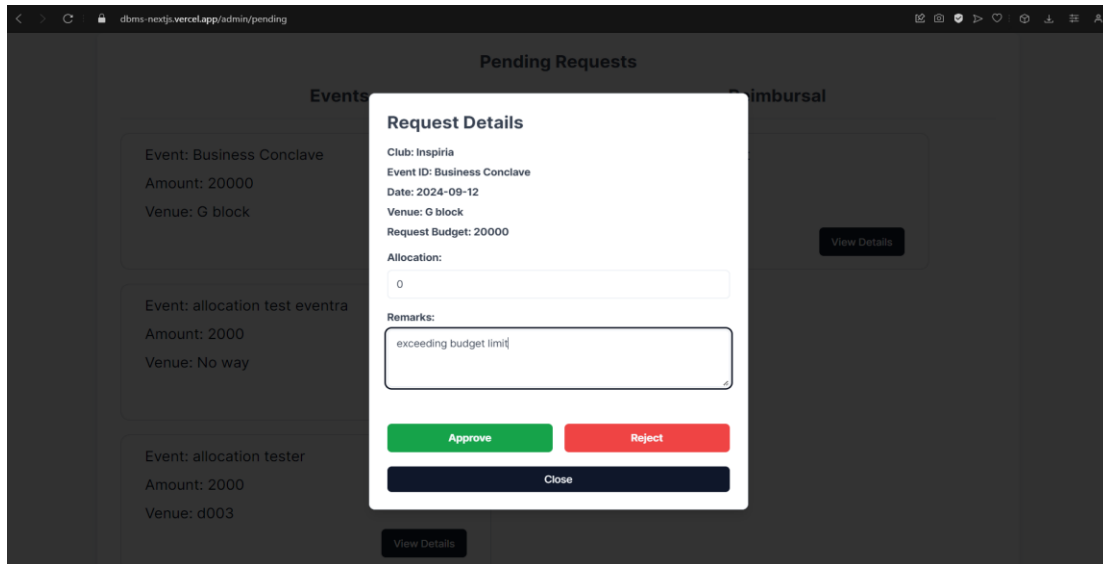

Rejection status showing on Club's request page.
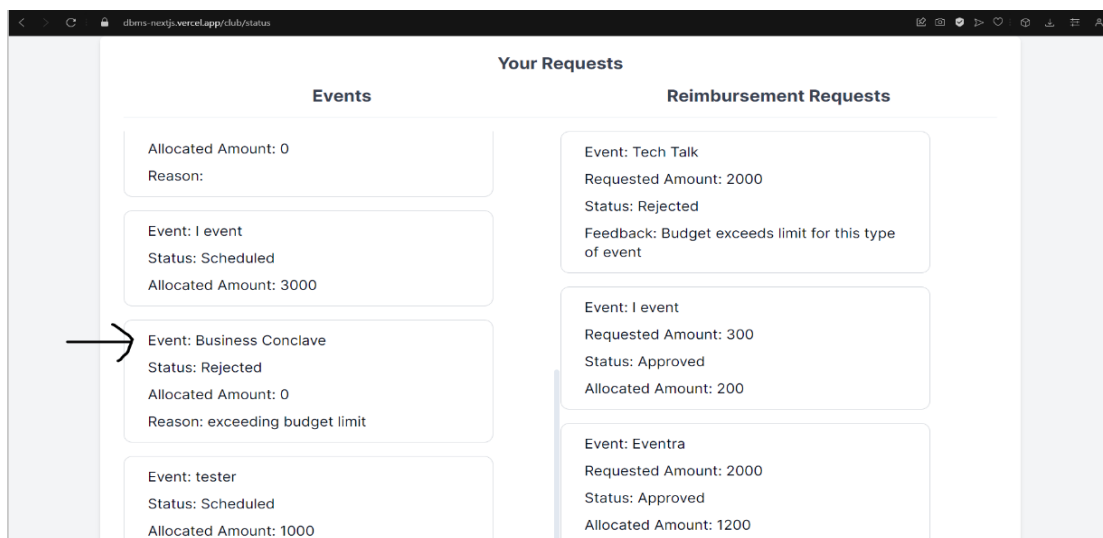
Approved Requests Page for Admin.
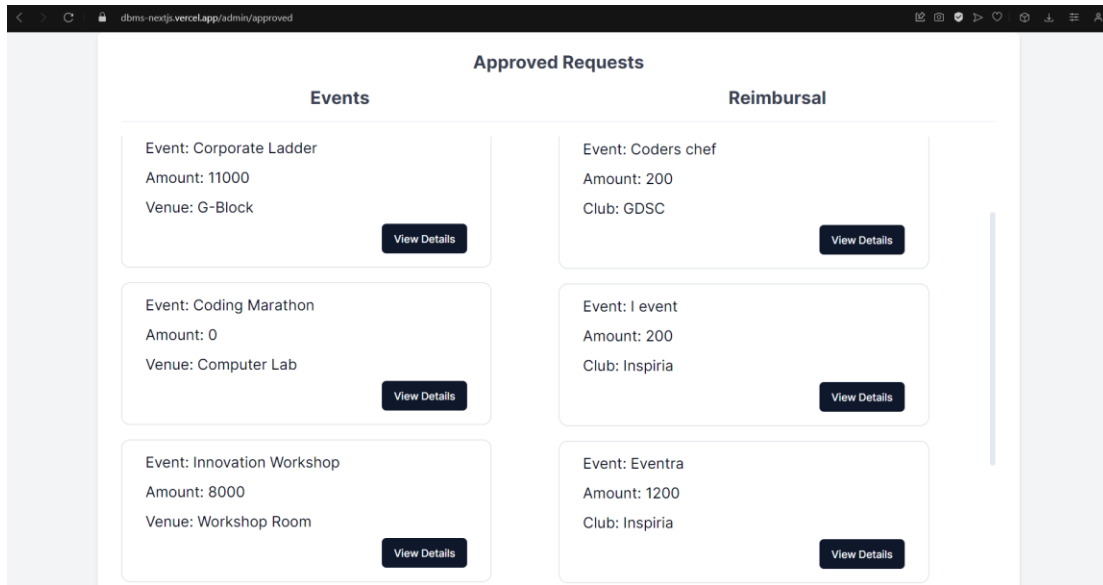
5. Backend Implementation and Trigger Functions

Backend Architecture

The backend of the club management system is built on Supabase, which provides a combination of database and backend services. It is responsible for handling data storage, retrieval, and manipulation, as well as enforcing business logic through trigger functions.

Trigger Functions

Trigger functions are crucial in maintaining data integrity and automating certain processes. Here are the trigger functions used in your project:

1. **checkifscheduled:** Ensures budget requests are only made for scheduled events.
2. **checkifscheduled_rsvp:** Verifies that RSVPs are only made for scheduled events.
3. **prevent_club_date_clashes:** Prevents clubs from scheduling multiple events on the same date.
4. **update_event_status_after_allocation:** Updates the event status to 'Scheduled' after budget allocation.
5. **update_event_status_after_rejection:** Sets the event status to 'Rejected' based on budget request rejections.
6. **validate_student_email_format:** Validates the format of student email addresses with the format @snu.edu.in.

Implementation Details

- **Data Handling:** CRUD (Create, Read, Update, Delete) operations are implemented to manage data across different tables.
- **Business Logic:** Trigger functions are set to automatically enforce business rules, such as preventing scheduling conflicts and ensuring proper budget request flows.

Challenges in Backend Development | Transactions on Update

One of the biggest challenges faced was managing concurrent actions by multiple admins. The initial approach was to implement a transaction property to lock requests being worked on by an admin. However, this led to issues where a locked request could remain inaccessible if an admin abruptly closed the website. This led us to eventually dropping the idea in the last minute rush.

Potential Solutions

To address this, a more robust transaction management system was needed. The solution was to implement a timeout mechanism that would release the lock on a request after a certain period of inactivity, thus preventing indefinite locking and allowing other admins to take over if necessary.

Code Example

Here's an example of a trigger function used in the project:

```
CREATE OR REPLACE FUNCTION prevent_club_date_clashes()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM Events
        WHERE ClubID = NEW.ClubID AND Date = NEW.Date AND status !=
'Rejected' AND EventID != NEW.EventID
    ) THEN
        RAISE EXCEPTION 'Another event by this club is already scheduled on
the same date.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

This trigger function is designed to prevent clubs from scheduling more than one event on the same date, ensuring that each club has a unique event schedule.

7. Challenges and Solutions

Overview

Throughout the development of the club management system, several challenges were encountered. These challenges required innovative solutions to ensure the system's functionality and reliability. This section will outline some of the significant challenges faced and the strategies employed to overcome them.

Challenge: Event Scheduling Conflicts

One of the initial challenges was managing event scheduling conflicts, especially when the same club attempted to schedule events on the same date as its other event.

Solution: A trigger function, `prevent_club_date_clashes`, was implemented to check for existing events on the proposed date. If a conflict was detected, the system would raise an exception, prompting the club to choose a different date or proceed with caution.

Challenge: Transaction Management | Failed Implementation

Another significant challenge was handling transactions, particularly when dealing with concurrent admin actions on budget requests.

Solution: A sophisticated transaction management system was needed, including a locking mechanism with a timeout feature. This ensured that if an admin did not complete their action within a specified timeframe, the request would be unlocked, allowing other admins to address it.

Challenge: Data Integrity

Maintaining data integrity, especially with multiple users interacting with the system, posed a challenge.

Solution: Trigger functions such as `update_event_status_after_allocation` and `update_event_status_after_rejection` were used to automatically update event statuses based on admin actions, thus maintaining data consistency.

Challenge: User Authentication and Middleware

Ensuring secure and reliable user authentication was crucial, given the sensitive nature of the data involved.

Solution: The system enforced strict authentication protocols and middleware, requiring students to log in with their college email IDs and roll numbers, while clubs and admins used their respective email IDs and passwords. Additionally, email format validation was implemented to prevent unauthorized access.

Challenge: System Usability

Creating a simple but user-friendly system that could be easily navigated by students, clubs, and admins was essential.

Solution: The frontend was designed with a focus on user experience, providing clear options and responsive design. React components were utilized to create an interactive interface that was both intuitive and efficient.

Summary

The challenges faced during the development of the club management system were diverse and required a range of solutions. From implementing trigger functions to manage data integrity to designing a user-friendly interface, each solution played a vital role in creating a robust and efficient system.

In the final section, we will provide a progress log and development timeline, detailing the from the project's inception to its completion.

8. Progress Log and Development Timeline

Introduction

A progress log and development timeline are essential components of any project report. They provide a historical record of the project's evolution, highlighting key milestones, challenges faced, and solutions implemented. This section will outline the progress made on the club management system from its conceptualization to its final implementation.

January – Ideation

- **Late January:** Decided on the idea of a Club management system and formulized our approach to the problem.

February - Project Inception

- **Early February:** The project was initiated with a focus on creating an Entity-Relationship Model (EER) and Relational Model to define the system's entities and their relationships.

March - Database and Backend Solutions

- **Mid-March:** The database schema was finalized, and tables were created in Supabase.
- **Late March:** Frontend structure was finalized. All the features that were to be implemented were finalized. The frontend design began, utilizing Next.js and React components.

April - Frontend Development, Backend Completion, Linking, and User Testing

- **Early April:** First review conducted. Reevaluated our progress and made changes to the idea. The initial idea of creating a math algorithm to assign a budget was dropped and a more manual was accepted. Frontend almost completed.
- **5-15 April:** User interface components were developed, focusing on responsive design and accessibility. Triggers and functions were created, and backend was finalised
- **Final Work:** The frontend and backend were integrated, ensuring seamless interaction between the two. User testing was conducted to gather feedback and refine the system. website was deployed locally.