Club Hub Final Report

Dr. Ramin Moazzeni

CS 157A

6 December 2024

Hoang Nguyen 016063619 hoang.k.nguyen@sjsu.edu

Nathan Durrant 017121078 nathan.durrant@sjsu.edu

Nathan Cohn 016839641 nathan.cohn@sjsu.edu

Shervan Shahparnia 017264195 shervan.shahparnia@sjsu.edu

Aaron Sam 014765140 aaron.Sam@sjsu.edu

Description and Application Goals:

Club Hub is a web-based platform designed to help university clubs manage their activities and improve communication among members. It provides tools for planning events, managing memberships, hosting discussions in forums, and encouraging community involvement.

The application addresses common challenges faced by university clubs, such as disorganized communication, difficulty managing events and memberships, and limited engagement tools. By centralizing these features, Club Hub makes it easier for both club leaders and members to stay organized and involved.

1. **Improve Communication**: Provide forums where club members can share ideas, ask questions, and discuss topics in an organized way.
2. **Simplify Event Management**: Allow club leaders to create and manage events, and enable members to view event details and sign up easily.
3. **Organize Memberships**: Offer features for users to join, leave, or request membership in clubs while letting leaders manage roles and approvals.
4. **Encourage Engagement**: Use notifications and discussion forums to help members stay connected and informed about club activities.
5. **Increase Efficiency**: Automate tasks like event sign-ups, membership tracking, and role assignments to save time for both members and leaders.
6. **Ensure Accessibility**: Provide an easy-to-use interface that works well on all devices.

Functional Requirements:

1. **User Registration and Profile Management:**
    - Registration/Login:
        - Users can register for an account by providing a school email, student ID. first name, last name, username, password.
        - Users can log in to their accounts with their username and password.
    - Profile Management:
        - Users can logout from their accounts.
2. **Member Functionality:**
    - Participate in Forums:
        - Members can create a post with title and original content/question within the club forum.
        - Members can reply to other members' posts in club forums.
    - Sign up for Events

        - Members can view the list of events posted by the club.
        - Members can sign up for events they plan on attending.
3. **Club Management:**
    - Club Owner Management:
        - Owners can create a club with a description.
    - Membership Management:
        - Users can see and search the list of clubs they can join.
        - Users can see past club descriptions and past events.
        - Users can join clubs.
4. **Events Announcement Management:**
    - Events Management:
        - Owner can create and announce events with various details (name, description, date, time, location, etc).
        - Members can view announcements.
5. **Forum/Discussion Board:**
    - Forum:
        - Club members can create a new thread post
        - Club members can participate in threads by replying to the main original post or other replies.

Architecture Overview

Club Hub employs a **2-Tier Database Management System (DBMS) Architecture**, characterized by a direct interaction between the application layer and the database layer. This architecture is designed to handle data storage, retrieval, and transaction processing efficiently, supporting dynamic user interactions via a web-based client-server model.

Back-End Architecture

The **server-side** is built with **Node.js**, using **Express.js** as the web application framework. It manages all API endpoints, processes requests, and interacts directly with the **MySQL database** for data manipulation and transaction fulfillment.

Key Features of the Back-End:

1. **Database Management**:
    - A **local MySQL database instance** serves as the core data store.
    - Queries are managed using **mysql** and **mysql2** libraries for efficient database interaction.

   - Implements database security features like parameterized queries to prevent SQL injection.
2. **Authentication and Security**:
   - User authentication is managed through **jsonwebtoken (JWT)**, ensuring secure session management.
   - Passwords are hashed using **bcrypt**, providing robust protection against breaches.
   - Cross-origin requests are enabled via **CORS**, allowing secure communication between the front-end and back-end.
3. **Environment Variables**:
   - Sensitive data like database credentials and JWT secrets are stored securely using the **dotenv** file, ensuring proper separation of configuration from code.
4. **Development Tools**:
   - **Nodemon** is used for hot-reloading during development, streamlining the server development process.

Front-End Architecture

The **client-side** is developed using **React.js**, a component-based library, enabling a dynamic and responsive user interface.

Key Features of the Front-End:

1. **API Communication**:
   - The front end communicates with the server via **Axios**, a promise-based HTTP client that simplifies asynchronous request handling.
   - It facilitates CRUD operations and real-time updates, enhancing the application's interactivity.
2. **Routing**:
   - **React Router DOM** manages client-side routing, allowing seamless navigation between pages without full-page reloads.
   - Features like protected routes ensure authenticated access to specific application sections.
3. **Validation and Feedback**:
   - User input is validated with the **validator** library to enhance data integrity and improve the user experience.
   - Front-end form validation works in conjunction with back-end validation for comprehensive error handling.
4. **Performance and Testing**:
   - Performance metrics are gathered using **web-vitals** to monitor the app's efficiency and optimize its responsiveness.

    - Component and user interaction testing are performed using libraries like **@testing-library/react**.

Inter-Tier Communication

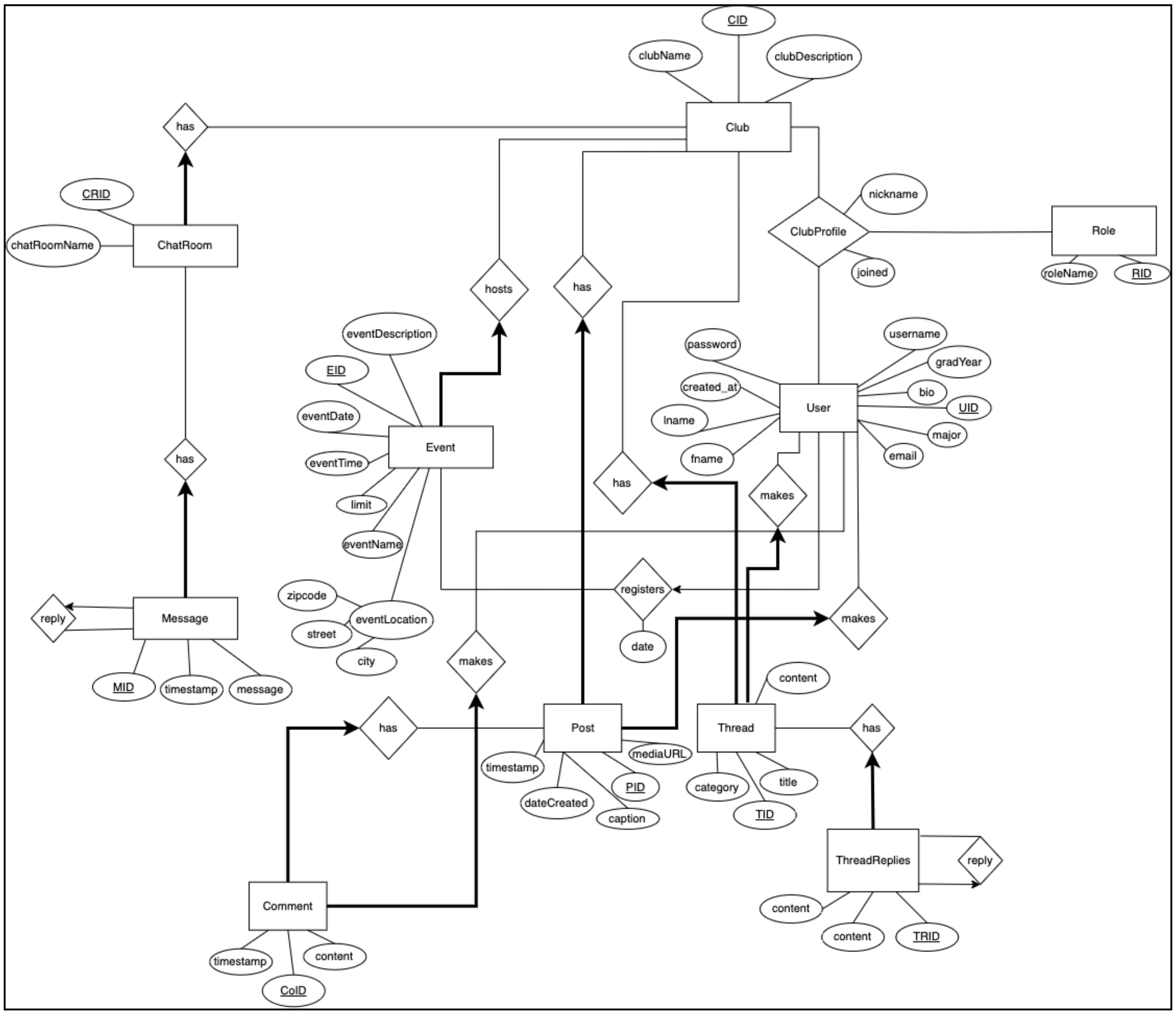
The communication between the React client and Express server relies on RESTful APIs. The back-end serves JSON responses to the front-end, allowing efficient data exchange. The system ensures high performance and scalability by adhering to principles like:

- **State Management**: Local component states and prop drilling are employed in the front end for dynamic data rendering.
- **Asynchronous Operations**: Axios enables non-blocking requests, ensuring a smooth user experience.
- **Error Handling**: Both front-end and back-end include robust error management, providing meaningful feedback to the user.

Technological Stack

1. **Programming Languages**:
    - **JavaScript** (Node.js for back-end and React.js for front-end).
2. **Database**:
    - **MySQL** (Relational Database Management System).
3. **Server Frameworks**:
    - **Express.js** (Back-End RESTful APIs).
4. **Libraries/Tools**:
    - Axios, Validator, JWT, Bcrypt, React Router DOM, Nodemon, dotenv, etc.

EER Model



Major Design Decisions

- Framework Selection: The application is built using React.js for the front end and Node.js with Express.js for the back end, ensuring seamless integration with a MySQL database.
- Database Design: The database design adheres to BCNF normalization to enhance data integrity and optimize query performance.
- Role Differentiation: Introduced distinct roles for members and owners, enabling tailored permissions for managing clubs and participating in activities.

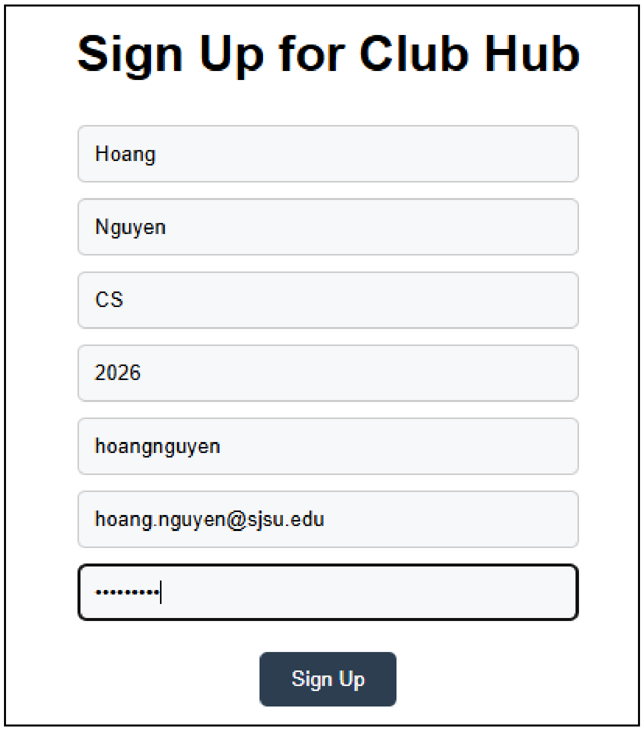
- Feature Prioritization: Focused on developing core functionalities first (e.g., user authentication, club creation) and added enhancements (e.g., search functionality, Dark Mode).

Implementation Details

- Authentication: Developed secure user registration and login mechanisms using custom authentication logic in Express.js. Passwords are hashed using bcrypt, and session management is handled with jsonwebtoken (JWT).
- Club and Event Management: Users can create clubs, post announcements, and host events. Club members can register for events hosted by their club.
- Forum and Club Interaction: Users can create threads and reply with nested comments to facilitate discussions.
  Clubs can manage member communications through posts and announcements.
- Search Functionality: Implemented keyword-based searches for clubs, threads, and events using optimized SQL queries.
- Frontend: Designed a user-friendly interface, with features like Dark Mode for accessibility.
- Backend: Exposed API endpoints for CRUD operations. Used Axios to handle communication between the React front end and the MySQL database.

Demonstration of Example System Run

1. **User Registration, Login, Logout**:
   - A new user registers with their first name, last name, major (optional), graduation year (optional), username, school email and password. Username and password are very important because they will use these two to login.
   - Users will be required to create a strong password based on the requirements. Each password will be hash before storing into the database.
   - User can also logout of their account


Sign Up for Club Hub

Hoang

Nguyen

CS

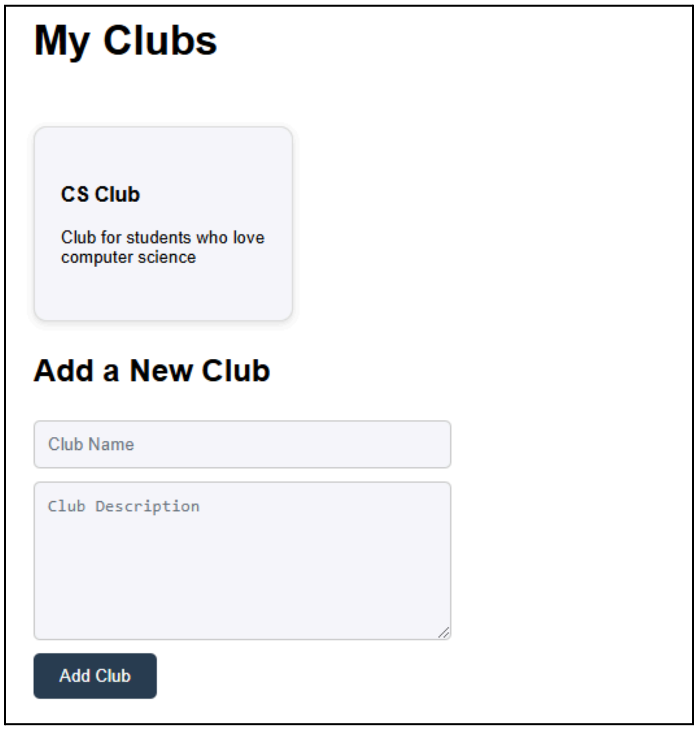
2026

hoangnguyen

hoang.nguyen@sjsu.edu

•••••••••

2. **User Dashboard**:
   - The user views the clubs that they joined, included the one that they created
   - The user will also have a one time option to create their own club

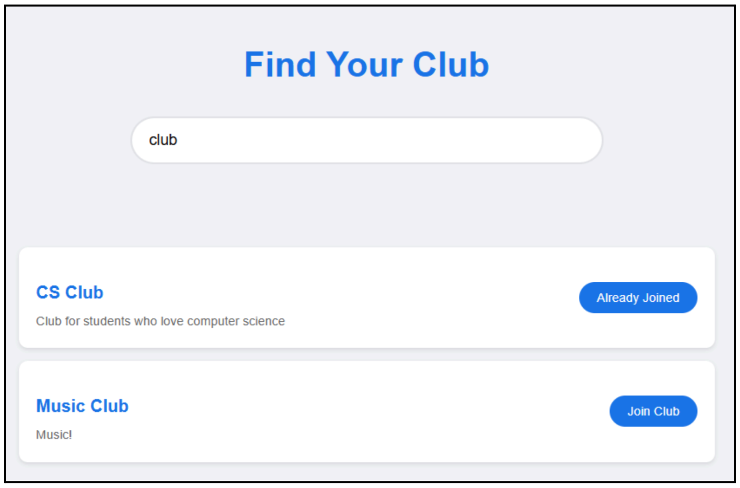

My Clubs

CS Club

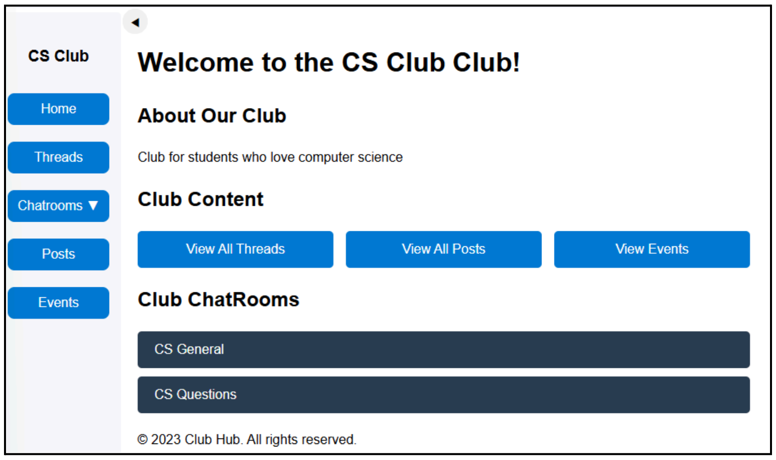
Club for students who love computer science

Add a New Club

3. **Club Search**:
    - The user can search up club based on the club name using keywords
    - If they already join a club or they are the owner of that club, it will said already joined
    - If not they can join the club and a confirm message will pop up

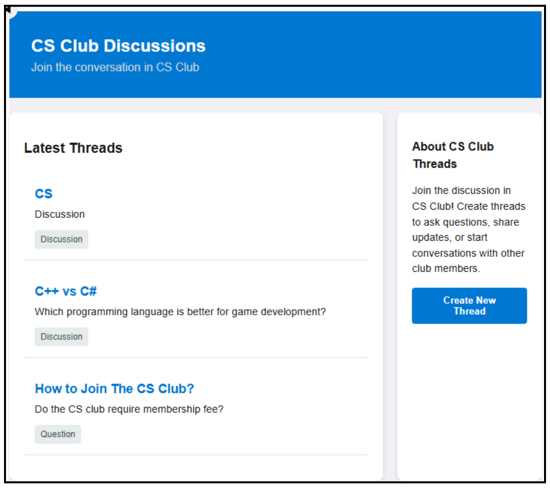


4. **Club Dashboard**:
    - The club dashboard should have the club name, description, navigation sidebar to the threads, posts, events, and list of chat rooms



5. **Club Management**:
    - Currently, there are two specific roles, club owner and club member. Owners have more permission in creating the content of the club. Member can only mostly view and comment except threads that are open to everyone
6. **Forum Interaction**:
    - Users engage in discussions by creating threads and posting replies.

○ Threads are open to all club members. Member can view list of threads, create new post, and add comments to each individual post



CS Club Discussions

Join the conversation in CS Club

Latest Threads

CS
Discussion
Discussion

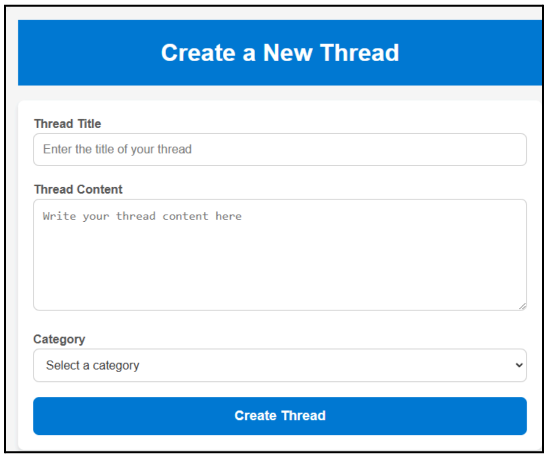
C++ vs C#
Which programming language is better for game development?
Discussion

How to Join The CS Club?
Do the CS club require membership fee?
Question

About CS Club Threads

Join the discussion in CS Club! Create threads to ask questions, share updates, or start conversations with other club members.

Create New Thread



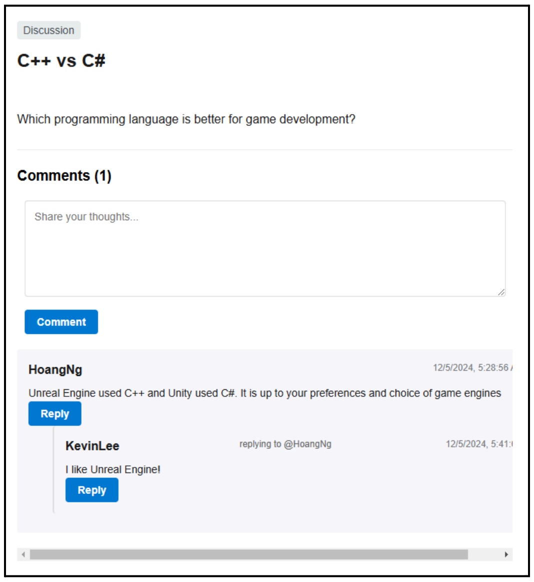
Create a New Thread

Thread Title
Enter the title of your thread

Thread Content
Write your thread content here

Category
Select a category

Create Thread



Discussion

C++ vs C#

Which programming language is better for game development?

Comments (1)

Share your thoughts...

Comment

HoangNg 12/5/2024, 5:28:56

Unreal Engine used C++ and Unity used C#. It is up to your preferences and choice of game engines

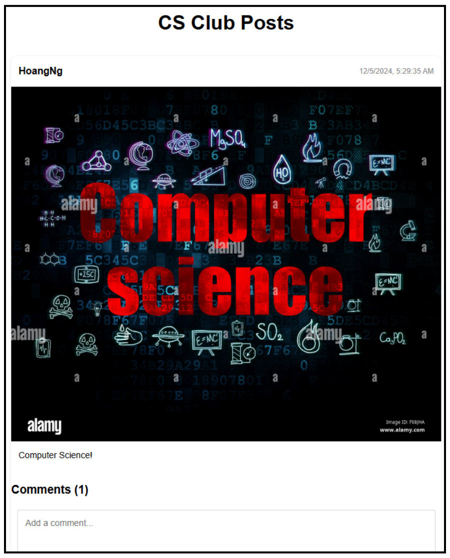
Reply

KevinLee replying to @HoangNg 12/5/2024, 5:41:

I like Unreal Engine!

Reply

7. **Announcement Posts**:
   - The post features are intended to be used as club announcements. Owner of the club can create a post with images and a description. Members can view and add comments to the post.



CS Club Posts

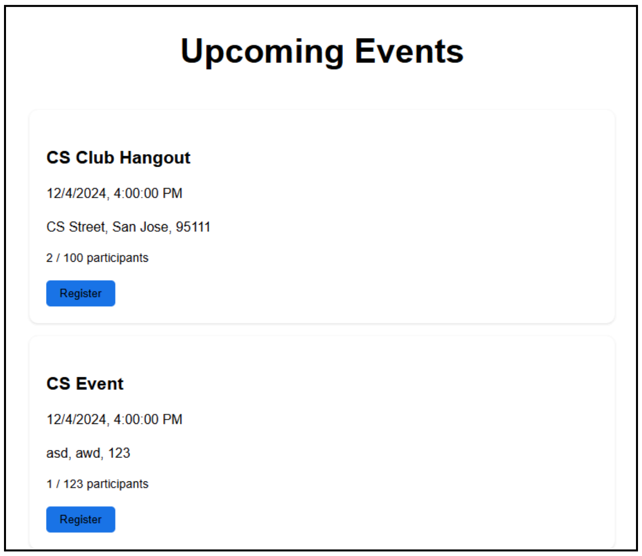
HoangNg

12/5/2024, 5:29:35 AM

Computer Science!

Comments (1)

8. **Club Events**:
   - Owners can create the events which include event title, date, participant capacity, and the location of where the event will be hosted
   - Members sign up for events using a streamlined interface
   - They can also unregister from the event



Upcoming Events

CS Club Hangout

12/4/2024, 4:00:00 PM

CS Street, San Jose, 95111

2 / 100 participants

CS Event

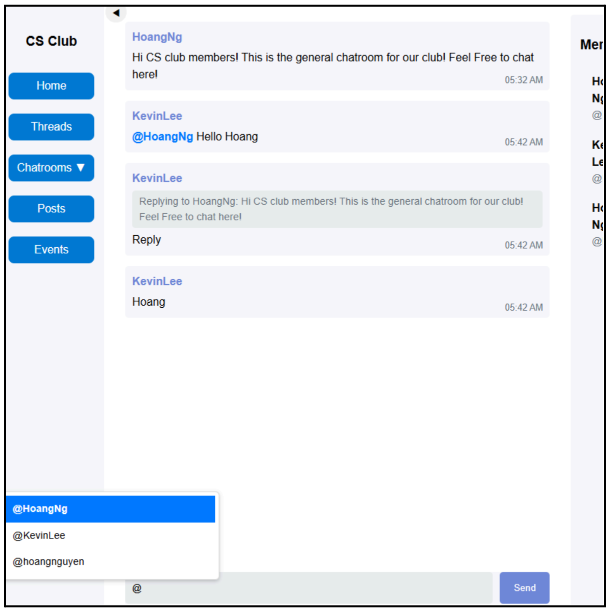
12/4/2024, 4:00:00 PM

asd, awd, 123

1 / 123 participants

9. **Chatrooms**:
   - Chatrooms can only be created by the owner of the club

- ○ Member can join a chatroom to talk to other members
- ○ User can view, send a message, reply to a message, and ping another user in the chatroom



HoangNg
Hi CS club members! This is the general chatroom for our club! Feel Free to chat here! 05:32 AM

KevinLee
@HoangNg Hello Hoang 05:42 AM

KevinLee
Replying to HoangNg: Hi CS club members! This is the general chatroom for our club! Feel Free to chat here!
Reply 05:42 AM

KevinLee
Hoang 05:42 AM

Link to Presentation + Demo:
https://drive.google.com/file/d/1GKz68Ypuwyq2cFkJOCKbAhzWHoQx1-Vp/view?usp=sharing

---

Relational Tables and Normalization To BCNF

1. User

Schema:

User (<u>UID: Int</u>, username: String, fname: String, lname: String, email: String, password: String, bio: String, major: String, gradYear: Int, created_at: Timestamp)

Functional Dependencies (FDs):

- UID → username, fname, lname, email, password, bio, major, gradYear, created_at
- username → fname, lname, email, password, bio, major, gradYear, created_at
- email → username, fname, lname, password, bio, major, gradYear, created_at

Candidate Key (CK):

- {UID, username, email}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies; all attributes are fully dependent on the candidate keys.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: User is in **BCNF**.

---

2. Club

Schema:

Club (CID: Int, name: String, description: String)

Functional Dependencies (FDs):

- CID → name, description
- name → CID, description

Candidate Key (CK):

- {CID, name}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: Club is in **BCNF**.

---

3. Event

Schema:

Event (EID: Int, name: String, date: Date, street: String, city: String, zipcode: Int, limit: Int, CID: Int)

Functional Dependencies (FDs):

- EID → name, date, street, city, zipcode, limit, CID

Candidate Key (CK):

- {EID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: Event is in **BCNF**.

---

4. EventRegistration

Schema:

EventRegistration (UID: Int, EID: Int, date: Date)

Functional Dependencies (FDs):

- UID, EID → date

Candidate Key (CK):

- {UID, EID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: EventRegistration is in **BCNF**.

---

5. Post

Schema:

Post (<u>PID: Int</u>, UID: Int, mediaURL: String, caption: String, timestamp: Timestamp, CID: Int)

Functional Dependencies (FDs):

- PID → UID, mediaURL, caption, timestamp, CID

Candidate Key (CK):

- {PID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: Post is in **BCNF**.

---

6. Comment

Schema:

Comment (<u>CoID: Int</u>, PID: Int, UID: Int, content: String, timestamp: Timestamp)

Functional Dependencies (FDs):

- CoID → PID, UID, content, timestamp

Candidate Key (CK):

- {CoID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: Comment is in **BCNF**.

---

7. ChatRoom

Schema:

ChatRoom (CRID: Int, name: String, CID: Int)

Functional Dependencies (FDs):

- CRID → name, CID

Candidate Key (CK):

- {CRID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: ChatRoom is in **BCNF**.

---

8. Message

Schema:

Message (MID: Int, CRID: Int, reply_to: Int, message: String, UID: Int, timestamp: Timestamp)

Functional Dependencies (FDs):

- MID → CRID, reply_to, message, UID, timestamp

Candidate Key (CK):

- {MID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: Message is in **BCNF**.

---

9. Thread

Schema:

Thread (TID: Int, UID: Int, title: String, category: String, content: String, CID: Int)

Functional Dependencies (FDs):

- TID → UID, title, category, content, CID

Candidate Key (CK):

- {TID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: Thread is in **BCNF**.

---

10. ThreadReply

Schema:

ThreadReply (TRID: Int, UID: Int, content: String, timestamp: Timestamp, TID: Int)

Functional Dependencies (FDs):

- TRID → TID, UID, content, timestamp

Candidate Key (CK):

- {TRID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: ThreadReply is in **BCNF**.

---

11. Role

Schema:

Role (RID: Int, name: String)

Functional Dependencies (FDs):

- RID → CID, name

Candidate Key (CK):

- {RID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: Role is in **BCNF**.

---

12. ClubProfile

Schema:

ClubProfile (UID: Int, RID: Int, CID: Int, nickname: String, joined: Date)

Functional Dependencies (FDs):

- UID, RID, CID → nickname, joined

Candidate Key (CK):

- {UID, RID, CID}

Normalization:

1. **1NF**: All fields are atomic.
2. **2NF**: No partial dependencies.
3. **3NF**: No transitive dependencies.
4. **BCNF**: Determinants in all FDs are superkeys.

Result: ClubProfile is in **BCNF**.

Conclusions

Club Hub is an intuitive website designed to streamline university club management and foster student engagement. It serves as a centralized platform for event planning, membership tracking, discussions, and community building. Using **React.js** for the front-end and **MySQL** for data storage ensures the application is fast, reliable, and capable of handling significant amounts of information. The 2-tier database system, facilitated by **Axios** for seamless communication between the client and server, provides a robust backbone for smooth operation. By addressing the needs of both club organizers and members, Club Hub simplifies club management and encourages student participation.

The current architecture is well-suited for small to medium-sized applications. However, as user demand grows, the system can evolve to a **3-tier architecture** by introducing a middle layer, such as a **GraphQL API**, to enhance scalability and decouple data processing. This forward-looking design allows for future enhancements, including integration with third-party services, improved database indexing, and real-time features using WebSockets, ensuring flexibility, maintainability, and a richer user experience.

The development of Club Hub provided valuable insights into project planning, teamwork, and technical execution. Effective planning and equitable task division were critical in maintaining project momentum and avoiding confusion. Selecting a tech stack familiar to the team and assigning tasks based on individual strengths enabled seamless collaboration. Regular meetings and progress check-ins ensured accountability and allowed for timely identification and resolution of challenges. Team members supported one another by stepping in or swapping tasks when necessary, fostering a cooperative environment.

A well-designed database using **Boyce-Codd Normal Form (BCNF)** proved essential for creating a reliable and efficient system by minimizing redundancy and optimizing query performance. Continuous testing throughout the project—via both **continuous integration** and **user acceptance testing (UAT)**—helped identify bugs early and refine the user interface. User-focused features, such as **Dark Mode** and a clean interface inspired by platforms like Canvas, made the platform accessible and appealing.

While Club Hub lays a solid foundation, several enhancements can improve its usability and functionality:

- **Real-time Notifications**: Event reminders, deadline alerts, and forum updates.
- **Recommendation System**: Personalized suggestions for clubs, events, and discussions based on user activity.
- **Cloud Hosting**: Transitioning from a local server to platforms like **AWS** or **Google Cloud** for improved scalability and reliability.
- **Advanced Analytics**: Tools for club administrators to analyze participation and engagement trends.
- **Forum Enhancements**: Features like rich text formatting, user tagging, and moderation tools to foster better discussions.
- **Mobile Accessibility**: A responsive mobile-friendly design or dedicated app to improve usability across devices.
- **Gamification**: Points, badges, and rewards to motivate user engagement.
- **University Integration**: Linking with student portals and event calendars for seamless adoption by schools.