# Rose-Hulman Club App

# Final Report

Matthew Morscher, Zixuan Yang, Zhou Zhou

# Table of Contents

## Executive Summary

This final report begins with a problem description that introduces the problem our app has solved. Next, we introduce how we solved this problem using both frontend and backend tools. Key challenges faced throughout the project will be talked about next. A full analysis of security concerns will be discussed following the challenges. We then overview our entire database and talk about its advantages and disadvantages. Then, an appendix is included with the relational schema and entity relational schema. Finally, a references page finishes up the report.

## Introduction

This is the final report for the Rose-Hulman Club-App that provides students with a mobile app for all club information and news. We will be delivering the problem we solved, solutions we took to solve this, key challenges we faced, and the design of our database, including the Relational Schema and Entity Relation Diagram. We will be revisiting items mentioned in the Security Analysis and Problem Statement as well.

## Problem Description

Many students attend the club fair in the beginning of the year and sign up for clubs they are interested in. There are two main issues with the club fair though. First, it is only once a year, which limits club exposure. The most popular other way to join a club is to find out about it from friends. Thus, if a student does not sign up for a club at the fair, it is likely they will not be joining that club during the year. Second, the only main form of communication for clubs is via email, which often bombards students with notifications. The lack of access to up-to-date club information and limited notification options make this app very desirable.

Our goal is to provide students with a centralized location for all club information on a mobile `platform. We aim to allow students to be able to perform anything club related (besides the actual activity) activity on this app. We hope to improve the student’s overall club experience and our project will be a success if we are able to deliver all of the features described below while making our end users happy.

| **Feature Number** | **Feature Name** | **Explanation** |
|---|---|---|
| 1 | Android Application | The system will be available for any Android smartphone |
| 2 | Club Information | User can view clubs and their respective information including the club name, type, description, files, and upcoming events |
| 3 | Login | User can log in with their Rose-Hulman credentials using RoseFire |
| 4 | Join/Leave from Club | User can join or leave a club with the click of a button |
| 5 | Subscribe/Unsubscribe from Club | User can subscribe or unsubscribe from a club without affecting their Member status |
| 6 | Events | Users can see all upcoming events for clubs on their homepage with their title and description |
| 7 | Sign up/Cancel Events | User can sign up or cancel |

| | | |
|---|---|---|
| | | joining club events |

# Solutions Descriptions

## Front-End Discussion

The front-end was designed for an Android application while being as simple as possible to use. When a user signs in, they are directed to a page where you can choose to see clubs or events. If the user chooses clubs, the user will see a list of all clubs at Rose with the ability to join or subscribe to those clubs. The user can click into a club to view a more detailed page as well. The user will then see the club name, type, description, media files, and upcoming events specific to that club. If the user is a manager of that club, they can see which users are signed up and subscribed to that club. If the user chooses events, the user will see a page with all upcoming events. Here, the user can click into an event and see a more detailed view including the event title and description, which club is hosting the event, and who is attending the event. The user can also choose their profile and only see the clubs they are a member of or subscribed to and which events they are signed up for.

## Back-End Discussion

The backend implements node.js and uses Let's Encrypt to provide our server with security. We use RoseFire to allow each user to sign in with their Rose-Hulman username and password. The server makes calls to the database to to get the user, clubs, events, and more. The database is currently located on the Titan Database and will expire on 2/23/17. You can see a more indepth analysis of the database in the Database Design Section.

## Key Challenges

We faced a few key challenges that will be talked about below:

1. Using an Android app
    a. By using the Android application, this required a secure certificate on our server. When we initially made the server, we did not see the benefit of setting it up with a certificate so we did not. After we realized this, we created a ticket with EIT but it took about a week to get the certificate. After a while, we went on our own and used Let's Encrypt instead.
    b. As well, it turns out both Android network library and Express library leaves out request body when performing a GET request. Due to this, we had to transition from using that to cookies. Once we did that and integrated, all worked as designed.
2. Encryption
    a. We hoped to use Kerberos to authenticate users as Rose-Hulman users, but after multiple attempts of trying to integrate it, we were unable to complete this task due to libraries that lacked documentation and were extremely old. Instead, we transitioned to use RoseFire.
    b. We began using RoseFire but when we attempted to set it up we kept running into “Invalid Algorithm” errors. We eventually figured out this was due to the way the app was initially registered with RoseFire and once we changed that it worked.
3. Stored Procedures
    a. The stored procedures themselves were not difficult to create, but what was difficult was to do multiple stored procedures within each function as needed. We ran into an issue where each stored procedure was trying to do too much at once and wasn’t returning the right data. Once we broke it up into smaller stored procedures, everything worked.
4. Small Mistakes
    a. During the project, just small little things kept causing us problems. That could be spelling errors, inconsistent namings of functions, and other similar issues.

# Security and Analysis

## Privacy Analysis

For our database, all usernames, names, and emails will be public to anyone with an account. When a user creates an account, they will use their Rose-Hulman username and their email will be generated by automatically adding “@rose-hulman.edu” to their username. As this app is designed to be used for Rose-Hulman students and their email will not be valid without a valid Rose-Hulman username, all information we have to be displayed is viable through the Rose-Hulman network. A user’s information will only be displayed when they are signed up for a club/event.

## Security Analysis

The implication of security of the system involves the following. In terms of data security, we need to guard against SQL injection attacks that can potentially sabotage the entire database. Should the project be implemented completely, possible breaching points include: officer adding textual description to the club and events, adding new officers, and searching for rooms by room name; student searching for clubs by club name, searching for events by event name, and rooms by room name. Most of these can be guarded against by the backend server. The database driver we use inherently includes proper escaping for frontend logics. On the other hand, any implementer of the frontend should also perform escaping on their end, in case there exists an injection type that affects the frontend as opposed to the backend, but it’s out of our control. Only the officers of a club can have the access to the data of the members and subscribers of that club. In order to exercise the rule, we require that the frontend acquire a JWT via RoseFire API, and supply that token in the form of Cookie. Then the identity of the user is verified with RoseFire API by the server. If no such credentials are provided, only limited information about clubs and events can be viewed, and no user information is ever displayed. Lastly, in order to avoid being attacked with high-volume requests, we adapted Nginx reverse proxy for Node server, and delegate the authentication responsibility to RoseFire.

## Entity Integrity Analysis

### User Table

The primary key is the rose_username that is a 10 character varchar. As well, there is a name field that is a 70 character varchar and an email that is a 30 character varchar. None of these fields can be null and the rose_username and email must be unique.

### Member_Of Table

The primary keys are the rose_username and club_name. The rose_username is a 10 character varchar and the club_name is a 50 character varchar. Neither of these values can be null.

Sign_Up Table

The primary keys are the rose_username and club_name and event_id. The rose_username is a 10 character varchar and the event_id is an integer. None of these values can be null.

Subscribe Table

The primary keys are the rose_username and club_name. The rose_username is a 10 character varchar and the club_name is a 50 character varchar. Neither of these values can be null.

Club Table

The primary key is the club_name that is a 50 character varchar. As well, each club has a type (Cars, Robotics, etc.) that is a 16 character varchar. There is also a description for each club that is a 200 character varchar. The description can be null, but none of the other values can be.

Manage Table

The primary keys are the rose_username and club_name. The rose_username is a 10 character varchar and the club_name is a 50 character varchar. There is also a title of each club officer in this table that is a 50 character varchar. None of these values can be null.

Event Table

The primary key is the event_id, which is an integer. There is also a title for the event which is a 100 character varchar and a description which is a 200 character varchar. The description can be null but the others do not have to be.

Media File Table

The primary key is the file_id which is an integer. There is also a file_path which is a 256 character varchar, a file_type which is a 10 character varchar, and a the club_name the file is related to, which is a 50 character varchar. None of these values can be null.

Hold Table

The primary keys are the club_name and the event_id. The club_name is a 50 character varchar and the event_id is an integer. Neither of these values can be null.

Room Table

The primary key is the room_number, which is a 20 character varchar. There is also a building for the location that is a 20 character varchar. Neither of these values can be null.

Reserve Table

The primary keys are the time_start, time_end, and room_number. The time_start and time_end are of type smalldatetime. The room_number is a 20 character varchar and the event_id is an integer.

## Referential Integrity Analysis

- User Table
  - Update: Cascade
  - Delete: Cascade
  - FK: None
- Member_Of Table
  - Update: Cascade
  - Delete: Cascade
  - FK: Member_Of_User
- Sign_Up Table
  - Update: Cascade
  - Delete: Cascade
  - FK: Sign_Up_event_id, Sign_Up_User
- Subscribe Table
  - Update: Cascade
  - Delete: Cascade
  - FK: Subscribe_Club, Subscribe_User
- Club Table
  - Update: Cascade
  - Delete: Cascade
  - FK: None
- Manage Table
  - Update: Cascade
  - Delete: Cascade
  - FK: Manage_Club, Manage_User
- Event Table
  - Update: Cascade
  - Delete: Cascade
  - FK: None
- Media File Table
  - Update: Cascade
  - Delete: Cascade
  - FK: Media_File_Club_Name
- Hold Table
  - Update: Cascade
  - Delete: Cascade
  - FK: Hold_Club_Name, Hold_Event_id
- Room Table
  - Update: Cascade
  - Delete: Cascade
  - FK: Room_event_id

Reserve Table

Update: Cascade

Delete: Cascade

FK: Reserve_Event_id, Reserve_Room

## Business Rule Integrity Analysis

When a club creates an event, the Hold table must be updated telling which club is holding which event. As well, when an event is created and they reserve a location, the Room table will be updated to show the appropriate location. When the owner of the club/app adds an officer, new entries are created in the Manages table and new privileges are given to those specific users.

# Database Design

## Stored Procedures

| **Stored Procedure Name** | **Procedure Purpose** |
|---|---|
| getUserRegistered | Checks if the user is already registered. Returns either nothing or the user that is registered. |
| createNewUser | Adds a new user to the database |
| fetchRegisteredUser | Fetches the user information (username, name and email) |
| fetchUserClubMember | Fetches all clubs the user is a member of |
| fetchUserClubSubscribe | Fetches all clubs the user is subscribed to |
| fetchUserOfficer | Fetches all clubs the user is an officer for |
| fetchEventsForUser | Fetches all events the user is signed up to attend |
| addUserToClub | Makes the user a member of the club |
| subscribeUserToClub | Subscribes the user to the club |
| unsignUpForClub | Removes the user from being a member of the club |
| unsubscribeClub | Unsubscribes the user from the club |
| getOneClubByClubName | Gets the club name, type, and description |
| getAllMembersByClubName | Gets all members of the club |
| getAllSubscribersByClubName | Gets all subscribers of the club |
| getAllManagersByClubName | Gets all managers/officers of the club |
| getAllFilesByClubName | Gets the files the club has |
| fetchClubNames | Gets the names of all the clubs |
| signUserUpForEvent | Signs the user up for events |

| | |
|---|---|
| removeUserFromEvent | Removes the user from attending the event |
| createClubEvent | Create an event with given parameters |
| cancelEvent | Cancels the given club event |
| fetchAllEvents | Fetches all event information and the club hosting the event |
| fetchUserEvents | Fetches the event information, club hosting the event, and whether or not the user is signed up for the event. |
| fetchEventsByClub | Fetches the event information and those attending for a specific club |
| fetchEventsSignedUp | Returns the users who are signed up for the event |
| fetchEvent | Fetches the event information given an event_id. |
| fetchEventByID | Fetches the users signed up for the the specific event |
| fetchAllRooms | Fetches all rooms that are being reserved |
| fetchSomeRooms | Fetches all room info within 3 days of the date provided |
| checkIsOfficer | Checks if the user is an officer for a club |
| getClubFromEvent | Gets the club name from the eventID |

## Views

| **View Name** | **View Purpose** |
|---|---|
| User_View | View for all user information |
| Club_List_View | Has all club information and the members and subscribers and the officers |

## Indexes

We have indexes on each table on the primary keys and a few foreign keys. We implemented these when creating the tables on the database.

## Triggers

| **Trigger Name** | **Trigger Purpose** |
|---|---|
| subscribeUserWhenMemberOf | When the user becomes a member of a club, they automatically are subscribed to the club |
| unsubscribeUserWhenNotMemberOf | When the user leaves a club, they automatically are unsubscribed from the club |

## Design Analysis

### Advantages

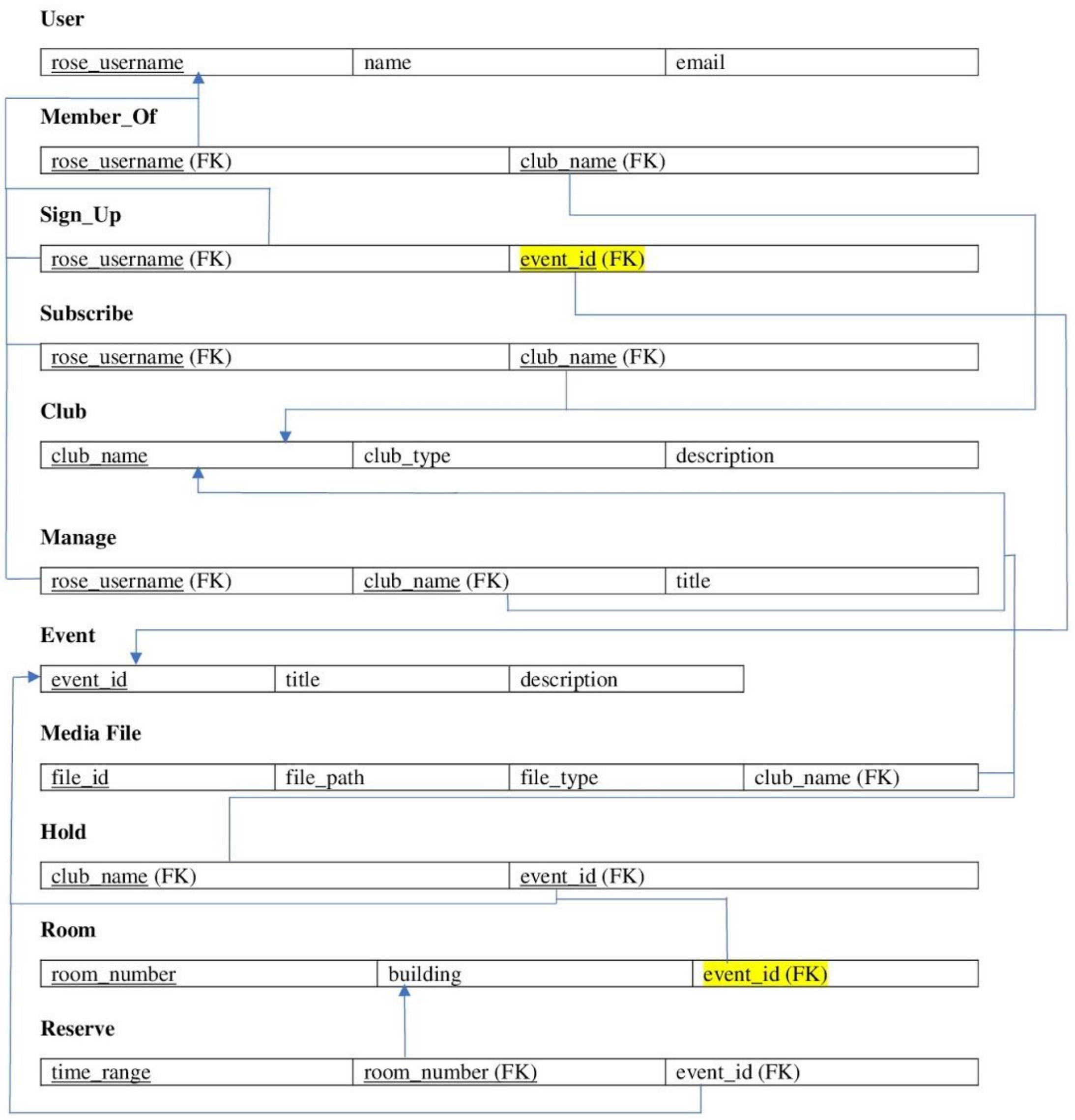
- Necessary PKs set to prevent unnecessary duplications
- Attribute names are clear and descriptive
- Rejection of information will occur to eliminate referential integrity constraints are not violated
- Appropriate integrity constraints are present
- No passwords are stored in the database
- Tokens are verified by  the web server

### Weaknesses

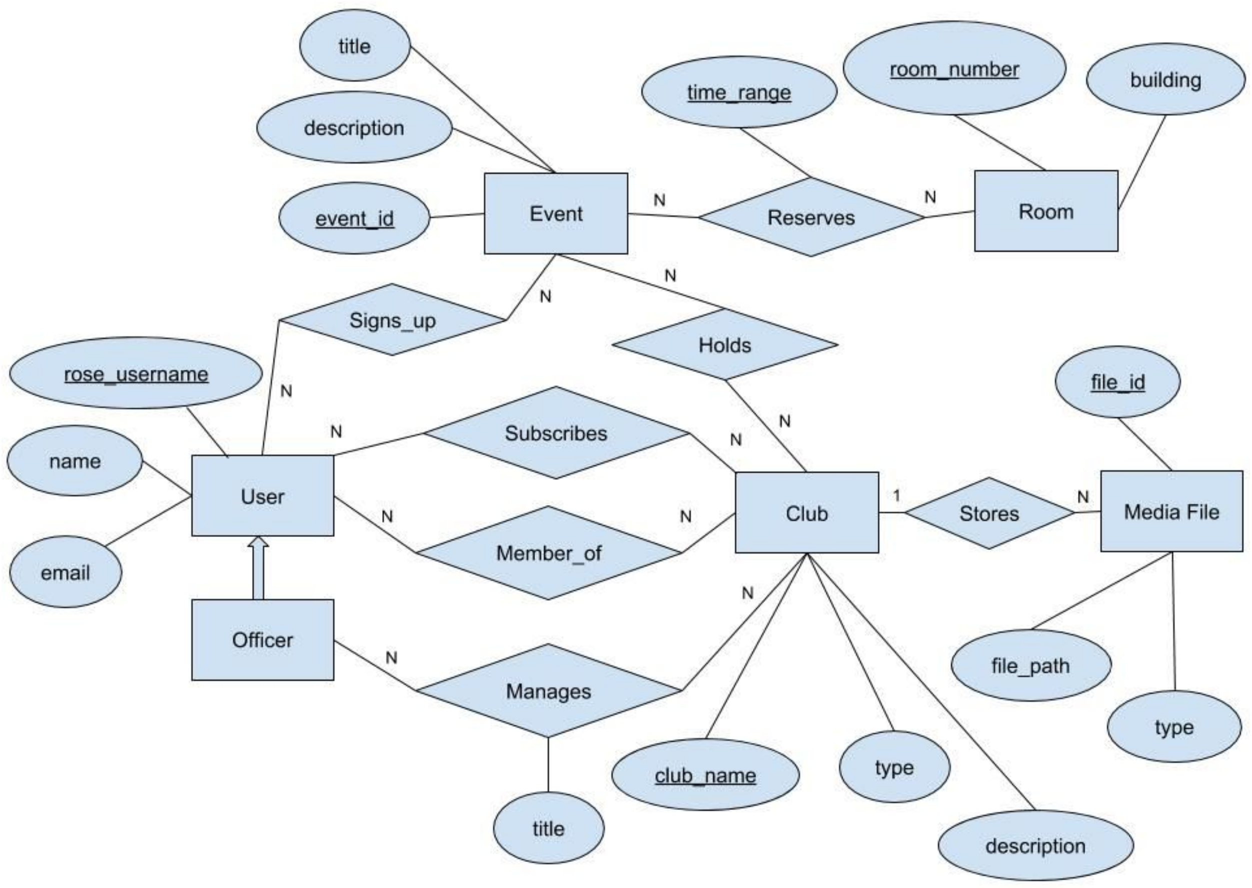
- The views are very rarely used and thus are wasting space in the database. This could be fixed by removing the views and changing the stored procedures that use the views to use tables instead.
- A few tables we initially planned on using we ended up not using, which, similar to the views, are wasting space in the database. This again can be fixed by removing those unnecessary tables.
- Naming of stored procedures could be improved

# Appendix A

## Relational Schema


User
rose_username | name | email
Member_Of
rose_username (FK) | club_name (FK)
Sign_Up
rose_username (FK) | event_id (FK)
Subscribe
rose_username (FK) | club_name (FK)
Club
club_name | club_type | description
Manage
rose_username (FK) | club_name (FK) | title
Event
event_id | title | description
Media File
file_id | file_path | file_type | club_name (FK)
Hold
club_name (FK) | event_id (FK)
Room
room_number | building | event_id (FK)
Reserve
time_range | room_number (FK) | event_id (FK)

## Entity Relational Schema



## Explanation of ER Diagram

The database is centered around Users. Users can be a member or subscriber of a club they are interested in. Users can also sign up for events they want to attend. An officer is a user that is in charge of a club. The database is also built around Clubs. A club can hold an event and stores media files that appear on the app. Finally the database is built around Events. Each event can reserve a room for a specific time to host the event and let others know when and where the event will be held.

## References

1. Android version market share distribution among smartphone owners as of September 2016: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/
2. Rose-Hulman club information: http://www.rose-hulman.edu/offices-and-services/student-life/student-activities/student-organizations.aspx
3. http://tediousjs.github.io/tedious/index.html