# Club Management System Report

## 1. Project Overview

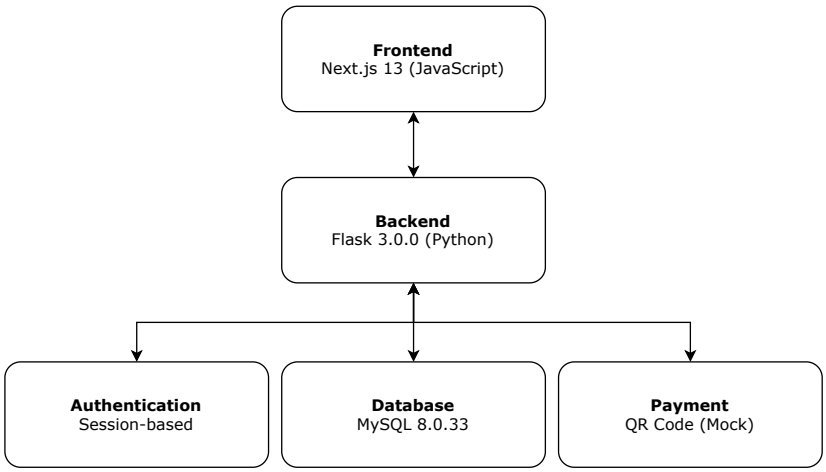
### Project Description

As clubs grow, managing members, activities, and payments becomes increasingly complex. Traditional methods, like spreadsheets or paper records, often lead to inefficiencies, errors, and missed opportunities for member engagement. This Club Management System addresses these challenges by centralizing essential functions, including membership tracking, activity registration, fee management, and internal communication. With role-based access for members, club admins, and system admins, the system streamlines operations, reduces administrative workload, and enhances the overall club experience for everyone involved.

This report outlines the Club Management System, a comprehensive platform designed for streamlined club operations. Key functionalities include:

- Membership management with different status levels (REGULAR, OFFICIAL)
- Activity organization and registration
- Fee management with QR code payment example
- Internal messaging system with reply functionality
- Role-based access control (Member, Club Admin, System Admin)

### System Architecture




### Technology Stack

- Frontend: Next.js 13 (App Router), JavaScript, Tailwind CSS, shadcn/ui components
- Backend: Flask 3.0.0 (Python), Raw SQL queries, MySQL Connector, Python libraries (qrcode, Pillow)
- Database: MySQL 8.0.33

## 2. Requirements Analysis

### Project objectives

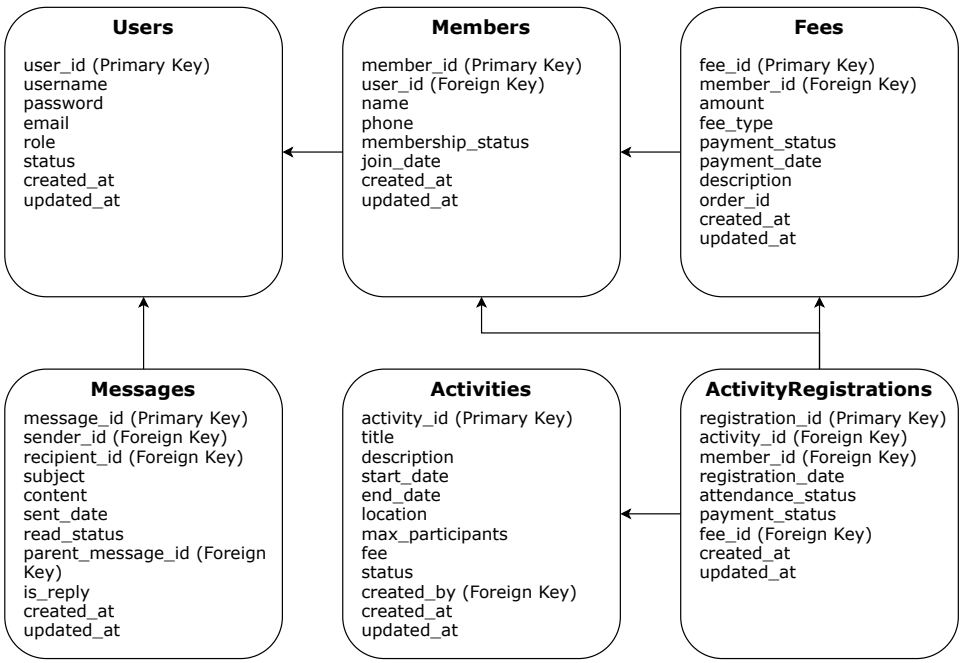
This project aims to develop a club management system that efficiently manages memberships, tracks activities, processes payments, and enables communication between members and administrators. Additionally, it will implement role-based access control for secure data access, ensuring that features are available based on user roles to maintain a clear administrative hierarchy.

### User Roles and Needs

1. System Administrator
    - Role: Highest level administrator responsible for system management and club admin oversight.
    - Functions:
        - Manage Club Administrators: Create new admin accounts, view all admins, change admin status
        - Sending messages to all members
2. Club Admin
    - Role: Manages day-to-day club operations, members, and activities.
    - Functions:
        - Manage members: View all members, change member status, export member email list
        - Activity management: Create new activities, monitor registrations, track attendance, update activity status, view participant lists
        - Fee management: View all fees, process payments, issue refunds, track payment status
        - Sending messages to all members
3. Members
    - Role: Regular club members who participate in activities and use club services.
    - Functions:
        - Membership management: Register account, view membership status, pay membership fees, update profile information
        - Activity participation: View available activities, register for activities, pay activity fees, view registration status, receive activity updates
        - Communication: Send/receive messages, view announcements, participate in discussions

# 3. Database and SQL Design

## Database Schema




Primary keys and Foreign keys are labeled in the diagram. Relationship between tables are labeled with arrows.

For each table, they are designed for the following purposes:

- Users Table: User authentication, role management, status tracking
- Members Table: Member information, membership status, relationship with Users
- Activities Table: Activity details, capacity management, status tracking
- ActivityRegistrations Table: Registration records, payment status, attendance tracking, relationship with Users and Activities
- Fees Table: Fee records, payment tracking, order management
- Messages Table: Message content, threading support, read status

### Key SQL Queries

One of the most important queries is to get all available activities with total participants and registration status. This is used in the frontend to display the user's activities and registration status.

```
SELECT
    a.*,
    ar_all.current_participants,
    ar.attendance_status as registration_status,
    ar.payment_status
FROM Activities a
LEFT JOIN ActivityRegistrations ar ON a.activity_id = ar.activity_id
    AND ar.member_id = %s
    AND ar.attendance_status != 'CANCELLED'
LEFT JOIN (
    SELECT COUNT(registration_id) as current_participants, activity_id
    FROM ActivityRegistrations ar_all
    WHERE ar_all.attendance_status != 'CANCELLED'
    GROUP BY activity_id
) ar_all ON a.activity_id = ar_all.activity_id
ORDER BY a.start_date ASC
```

Here, we select all activites info, and join with ActivityRegistrations to count total participants, and left join to get registration status and payment status for current user. Finally, we order by activity start date.

## 4. Implementation Details

For backend, Flask's RESTful API is used to handle requests and responses. Each route is defined in a separate file, and they are located in the `backend/routes` directory. For frontend, Next.js is used to handle the UI. Each page is a separate component, and they are located in the `frontend/app` directory. The directory structure is as follows:

```
club-manage-system
├── backend
│   └── routes (RESTful API)
├── docker
├── docs
└── frontend
    ├── app
    │   ├── activities
    │   ├── admin
    │   ├── fees
    │   ├── fonts
    │   ├── login
    │   ├── member
    │   ├── messages
    │   ├── register
    │   └── sysadmin
    ├── components (UI components)
    │   └── ui (shadcn/ui components)
    └── lib
```

### Function Implementation Table

In this project, implemented functions counts are as follows:

- Frontend: 21 (By searching `export default function` in the codebase)
- Backend: 50 (By searching `def` in the codebase)
- SQL: 62 (By searching `cursor.execute("""` in the codebase)

For primary functions are listed below, the format for the implementation is `<path/to/file>:<function_name>`.

| Requirements | Implementation | Function Introduction |
|---|---|---|
| **User Authentication** | | |
| Login | frontend/app/login/page.tsx: `LoginPage` | Provides login form UI and handles user input |
| | backend/app.py: `login` | Validates credentials and creates user session |
| Register | frontend/app/register/page.tsx: `RegisterPage` | Handles new user registration form and input validation |
| | backend/app.py: `register` | Creates new user account and initial member record |
| **Member Management** | | |
| View Members | frontend/app/admin/members/page.tsx: `AdminMembersPage` | Displays member list with status and actions for admins |
| | backend/routes/admin_routes.py: `get_all_members` | Retrieves all member data with filtering and sorting |
| Change Status | frontend/app/admin/members/page.tsx: `handleStatusChange` | Provides UI for changing member status |
| | backend/routes/admin_routes.py: `update_member_status` | Updates member status in database |
| **Activity Management** | | |
| Create Activity | frontend/app/admin/activities/new/page.tsx: `NewActivityPage` | Form for creating new activities |
| | backend/routes/admin_routes.py: `create_activity` | Handles activity creation in database |
| View Activities | frontend/app/activities/page.tsx: `ActivitiesPage` | Lists all activities with filtering by status |
| | backend/routes/activity_routes.py: `get_activities` | Retrieves activities with registration status |
| Activity Details | frontend/app/activities/[activityId]/page.tsx: `ActivityDetailPage` | Shows detailed activity info and registration |
| | backend/routes/activity_routes.py: `get_activity` | Gets single activity details |
| Register Activity | frontend/activities/[activityId]/page.tsx: `handleRegister` | Handles activity registration UI |

| Requirements | Implementation | Function Introduction |
|---|---|---|
| | backend/routes/member_routes.py: `register_activity` | Processes activity registration with timeout |
| **Fee Management** | | |
| View Fees | frontend/app/fees/page.tsx: `FeesPage` | Shows user's fees and payment status |
| | backend/routes/member_routes.py: `get_member_fees` | Retrieves user's fee records |
| Payment Processing | frontend/app/fees/page.tsx: `initiatePayment` | Handles payment initiation and QR code display |
| | backend/routes/fee_routes.py: `initiate_payment` | Generates payment QR code and order |
| Admin Fee Management | frontend/app/admin/fees/page.tsx: `AdminFeesPage` | Admin interface for fee management |
| | backend/routes/fee_routes.py: `get_all_fees` | Retrieves all fee records for admin |
| **Messaging System** | | |
| Inbox | frontend/app/messages/inbox/page.tsx: `InboxPage` | Shows user's received messages |
| | backend/routes/message_routes.py: `get_inbox` | Retrieves user's messages |
| Send Message | frontend/app/messages/message/[messageId]/page.tsx: `MessageDetailPage` | Interface for viewing and replying to messages |
| | backend/routes/message_routes.py: `send_message` | Handles message sending and threading |
| **System Administration** | | |
| Admin Management | frontend/app/sysadmin/dashboard/page.tsx: `SystemAdminDashboardPage` | System admin dashboard and controls |
| | backend/routes/admin_routes.py: `get_system_stats` | Retrieves system-wide statistics |
| Admin Status | frontend/app/sysadmin/admin/manage/page.tsx: `handleStatusChange` | UI for managing admin accounts |
| | backend/routes/admin_routes.py: `update_admin_status` | Updates admin account status |

## 5. System Usage Guide

### Installation Instructions

1. Backend Setup

```
cd backend
vim db_connection.py # Configure database connection, remember to import backend/init.sql into MySQL
pip install -r requirements.txt
flask run --debug
```

2. Frontend Setup

```
cd frontend
npm install
npm run dev
```

### User Guide

1. Member Functions

   - Registration process: Enter system, click register, fill in info, and pay fees.
   - Activity participation: View activities, register for activities, pay fees, and view registration status.
   - Message management: Send/receive messages, view announcements.

2. Admin Functions

   - Member management: View all members, change member status.
   - Activity management: Create new activities, monitor registrations, track attendance, update activity status, view participant lists.
   - Payment handling: View all fees, process payments, issue refunds, track payment status.
   - Fee report generation: Export fee report.

3. System Admin Functions

   - Admin account management: Create new admin accounts, view all admins, change admin status.
   - System monitoring: View system stats, including total members, total activities, total registrations, active admins, database status, and system uptime.

## Test Account

```
System Admin: system_admin
Password: password123

Club Admin - Active: sarah_admin
Password: password123

Club Admin - Suspended: jane_admin
Password: password123

Member - Official: peter_member
Password: password123

Member - Regular: mary_member
Password: password123

Member - Expired: alice_member
Password: password123
```