Bilkent University

Department of Computer Engineering

CS319 Term Project

Group 1D - CluBil

Project Design Report

İdil YILMAZ - 21703556
Doğukan Ertunga KURNAZ - 21702331
Turgut Alp EDIS - 21702587
Taha Batur ŞENLI- 21901857
Argun MURADOV - 21901305
Cemhan Kaan ÖZALTAN - 21902695

Instructor: Eray Tüzün
Teaching Assistant(s): Elgun Jabrayilzade, Emre Sülün, Erdem Tuna, Muhammad Umair Ahmed and Cevat Aykan Sevinç

Dec 18, 2021

Contents

1 Introduction

1.1 Purpose of the System

The program is a web-based student club system which aims to help manage student clubs and handle various interactions between students and clubs. It allows student clubs to easily organize and promote events while also ensuring a clear communication between the Student Activities Center (SAC) and student clubs. The users in the program are the students, student clubs and the SAC Admin.

In terms of features, the program allows students to see all the activities organized by clubs, receive notifications about specific activities and customize their profile. Additionally, students can join or leave student clubs and activities. Student clubs, on the other hand, can organize or delete activities, manage their expenses, notify their members, customize their profile and manage their budget which is assigned based on how many members they have. Finally, the SAC Admin can add or delete clubs and students, change club budgets and approve or refuse events.

1.2 Design Goals

The design goals were determined using the requirement report's non-functional requirements. The program needs to be easy to use, because it is designed to be a straightforward alternative to club management. Additionally, due to the fluctuating nature of Bilkent University student clubs, the system needs to be easily maintained by academic personnel.

1.2.1 Usability

The system will be potentially used by all Bilkent University students, so the software needs to provide ease to use. Besides, the user can learn the usage of the system easily. The usage of the system is kept on it's the simplest level. By not adding very complex functions and operations for the users who possibly do not understand the websites much to the software, it is possible to provide users a better experience.

1.2.2 Maintainability

When the users of the program increase, the need for maintenance will increase. Besides, since the program will be used in real life, the need for maintenance is important. Therefore, the maintenance becomes a design concern for this project and the aim is to keep the maintenance effort and cost at the lowest possible level. This will be done by using object-oriented programming in addition to two design patterns. These are, Singleton and Facade. By doing this, CluBil's implementation will have a higher flexibility and allow change more easily when needed. Moreover, it enables us to manage our backbone code with a special layer between the user and the backend.

2 High-level Software Architecture

2.1 Subsystem Decomposition

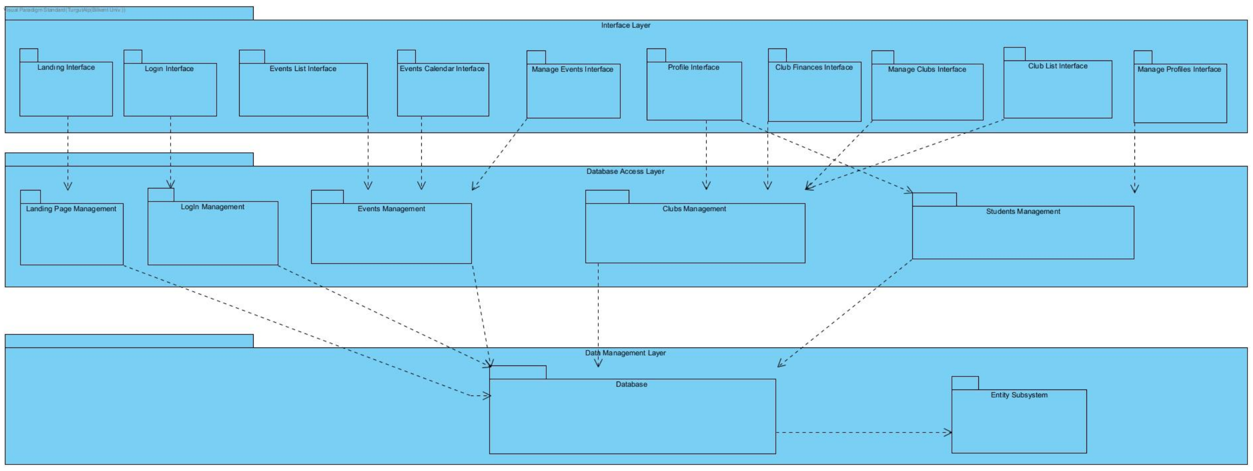

Figure 1. Subsystem Decomposition

The architecture of our system consists of 3 layers.

The first layer, the *Interface Layer*, contains the pages of our system. Every page has a controller in the database access layer to control the specific part of the system.

The *Database Access Layer* contains the front-end. It contains all important interfaces to control the *Interface Layer* and it still runs in the front-end code and is used to fetch and post relevant data from and to our Database. It is separated into packages in order to avoid a single package that has access to the whole database. Instead, every package has a specific purpose and accesses the relevant part of the Database.
Lastly, the *Data Management Layer* consists of the Database that is controlled by the *Database Access Layer*. Subsystems are explained below.

2.1.1 Interface Layer

Landing Interface: Landing interface is a subsystem that contains all the UI elements and all the user interactions required by the landing page.

Login Interface: Login interface is a subsystem that contains all the UI elements and all the user interactions required by the login page.

Event List Interface: Event list interface is a subsystem that contains all the UI elements and all the user interactions required by the event list page.

Events Calendar Interface: Events calendar interface is a subsystem that contains all the UI elements and all the user interactions required by the calendar page.

Manage Events Interface: Manage events interface is a subsystem that contains all the UI elements and all the user interactions required by the manage events page.

Profile Interface: Profile interface is a subsystem that contains all the UI elements and all the user interactions required by the profile page.

Club Finances Interface: Club finances interface is a subsystem that contains all the UI elements and all the user interactions required by the finance page.

Manage Club Interface: Manage club interface is a subsystem that contains all the UI elements and all the user interactions required by the manage club page.

Club List Interface: Club list interface is a subsystem that contains all the UI elements and all the user interactions required by the club list page.

Manage Profiles Interface: Manage profiles interface is a subsystem that contains all the UI elements and all the user interactions required by the manage profiles page.

2.1.2 Data Management Layer

Landing Page Management: Landing page management is a layer between our UI elements and the database. It enables us to fetch, store, update, and delete the necessary data from our database for landing page's front-end components.

Login Management: Login management is a layer between our UI elements and the database. It enables us to fetch, store, update, and delete the necessary data from our database for login page's front-end components.

Events Management: Events management is a layer between our UI elements and the database. It enables us to fetch, store, update, and delete the necessary data from our database for Event List page's front-end components.

Clubs Management: Clubs management is a layer between our UI elements and the database. It enables us to fetch, store, update, and delete the necessary data from our database for Club List page's front-end components.

Students Management: Students management is a layer between our UI elements and the database. It enables us to fetch, store, update, and delete the necessary data from our database for Profile page's front-end components.

2.1.3 Database Layer

Database: Database is a part of Data Management Layer. The layers in the database access layer can access the database through this layer. Database layer is where our data will be stored.

Entity Subsystem: The entity subsystem package will include the backend entity objects that will be used jointly with the data fetched from our database.

2.2 Hardware/Software Mapping

The program doesn't have any specific hardware requirement to function effectively. However, the primary features of a computer hardware, namely a keyboard, screen, mouse and a system sufficiently strong enough to run web browsers is needed. The minimum requirements are as follows [2]:

- 1.9 gigahertz (GHz) x86 or x64 bit dual core processor
- 2GB RAM
- Super VGA with a resolution of 1024 x 768
- Bandwidth greater than 50 KBps (400 kbps)
- Latency under 150 ms

Additionally, the program can be used on tablets and mobile phones. For software, the program will be developed using Javascript ECMAScript 2015 and React Library version 17.0.2. As such, the following web browser versions and above are compatible with the program:

- Internet Explorer 10 and above

- Microsoft Edge 12 and above
- Mozilla Firefox 21 and above
- Google Chrome 23 and above
- Safari 6 and above
- Opera 15 and above
- Safari (iOS version) 6 and above
- Android browser version 4.4 and above
- Opera mobile version 64 and above
- Chrome for Android version 96 and above
- Firefox for Android version 94 and above

2.3 Persistent Data Management

To ensure the constant data flow within our app, we will use Firebase as our data management system. Firebase Realtime Databases provide JSON-formatted data that we will use to record our events, clubs and student profiles under different JSON trees. The user will be able to push and fetch the necessary information from an to the database while using CluBil. This data will include event names, descriptions, dates, start and end times, budget amounts, user IDs and further information to be displayed for the user. These data will then be placed inside their respective entity objects such as Student and Event. The reason Firebase was chosen for this project is it’s great compatibility with Javascript libraries, fast request times, easy-to-implement functionalities and encrypted secure data transfer between our UI and databases. Another reason is that our members had the most experience with this system. Firebase Authentication will be used to ensure secure, hashed LogIn. Also, instead of keeping passwords as plain-text, which is a really bad idea for security, Firebase Authentication makes sure even if someone can somehow fetch data from our database, they will not be able to view critical information such as passwords since they will be stored in a hashed manner.

2.4 Access Control and Security

Due to its nature as a student club manager, the program enforces certain security precautions and access control. Login credentials, provided by the team, will be used to distinguish different types of users such as students, club profiles and SAC Admins. Functionalities aside, these user types also have different permissions. Students are not permitted to edit clubs, budgets or activities, they are only permitted to edit their profile, join or leave activities and/or clubs. Clubs have permissions allowing them to create activity or budget proposals that will be sent to the SAC Admin and can cancel approved or pending activities they have organized. Additionally, Student clubs can edit their profiles and are permitted to send general notification messages to their members, which is stored by the program. Finally, SAC Admins are given the greatest amount of access. They can edit their profile, are permitted to add or delete student clubs and can manage their budgets. They are allowed to accept or delete club events and can also delete approved club events.

We also take several security precautions in order to protect the personal information of our users. The Firebase Authentication Database and its built-in security features help us ensure safety for all sorts of authentication within the system. The access control matrix below demonstrates the access privileges of different users to different functionalities.

	Student	Club Manager	SAC Administrator	Unauthorized User	Club Advisor
Login	X	X	X	X	X
Join/Leave Event	X				
Accept/Decline Invitation	X				
Edit Profile	X	X			
Change Password	X	X			

Add/Remove Event Request		X			
Accept/Decline Event Request			X		
Add/Remove Event Cost		X			
Delete Own Event		X			
Delete Any Event			X		
Add/Delete User			X		
Set Budget			X		
See Finance of Own Club		X			X
Give Request Feedback					X
See Event List	X	X	X		X
See Public Events				X	
See Club List	X	X	X		X
See Event Calendar	X	X			

2.5 Boundary Conditions

2.5.1 Initialization

The program is a web based application and therefore does not require installation. Most browsers can be used to access this site with an internet connection. To deploy our project, we will use a npm package called "*gh-pages*". After installation of this package, we should edit the "*package.json*" file to set our "*homepage*" url. After adding the homepage url in the package.json file, we should add the deploy command to the "*scripts*" part of the "*package.json*" file. The command should look like `"deploy": "npm run build && gh-pages -b master -d build".` This command will push the builded version of our repo to the master branch. At this point, we suggest that you make a dev branch of your repo to not lose your development process. Then, “npm run deploy” command will start your web app in the *your_repo_name.github.io* domain.

2.5.2 Termination

The program works separately, so the termination of one subsystem does not affect the entire application. Instead, that subsystem alone is terminated while other functionalities remain accessible. However, an admin can initiate a general termination, terminating all subsystems and therefore the entire application. Moreover, the program uses Firebase to save information and, in the case of a termination, the information on the system is saved before termination occurs.

2.5.2 Failure

The program uses the Firebase system for data storage and if the system fails it reacts by restarting itself. If this restart does not fix the failure, the developers are warned via a notification. For developer-related bugs, the system warns the developer before the changes are complete.

2.6 Deployment Diagram

This is a deployment diagram of our project. The application will be deployed according to this diagram. We will use cloud storage as a hosting service and because we are using Firebase in this project, we are also using FirebaseDB as a DB solution.

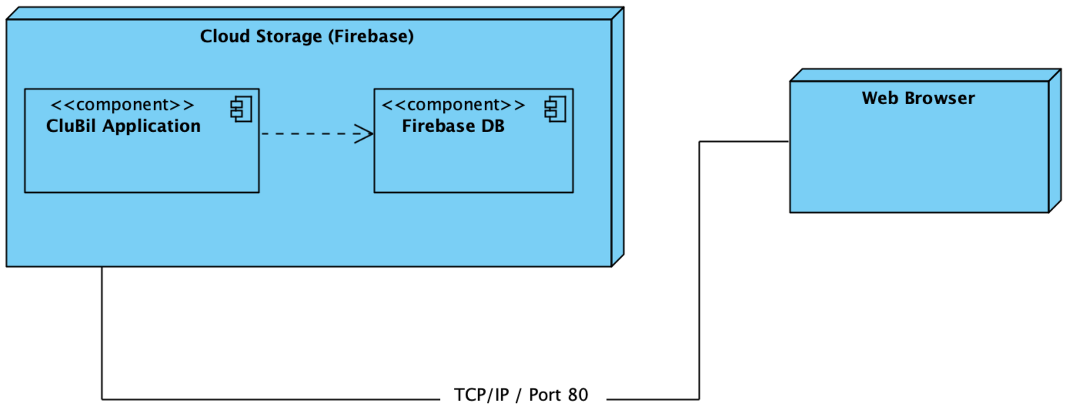

Figure 2. Deployment Diagram

3 Low-level Design

3.1 Object Design Trade-Offs

3.1.1 Usability vs Functionality:

The program's aim is to make software as simple as possible so that every Bilkent University student can use this software. Though, making this program simple causes trouble about functionality since more operations makes the software more complex, then the usability will decrease. Therefore, some possible functionalities that can be used for this software are abandoned to ensure that the software will become more usable.

3.1.2 Usability vs Security:

The program offers some security properties to protect the data of the students, admin and clubs. However, the data security is limited at some point because making more secure software results in more complex software, so there is a conflict on the design goal. Since security is not included in the top design goals of the program, the idea of providing more security was given up to make the software usable.

3.1.3 Maintainability vs Performance:

One of the top design goals of the program is maintainability. The number of layers is enough to cover all parts of the software. It causes the loss of the performance. However, it grants great advantage to the maintainability of the program. Besides, though increasing maintainability causes performance loss, the performance can be abandoned to keep maintainability at its peak level.

3.2 Final Object Design

The final object design is given below.

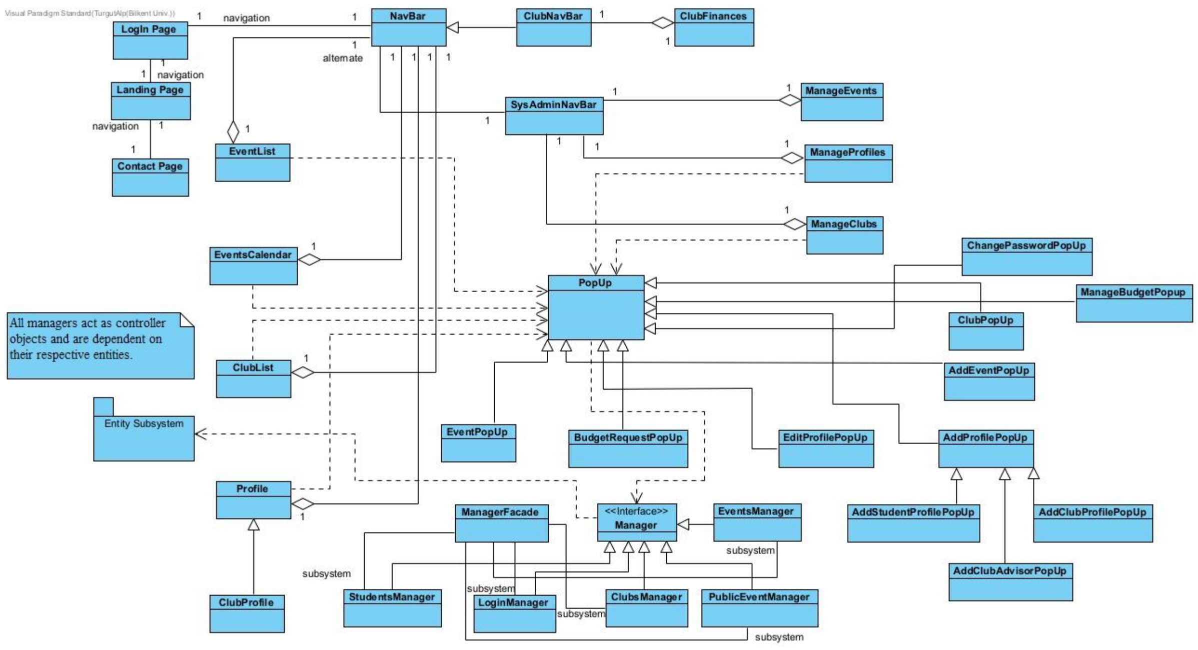

Figure 3. Final Object Design

3.3 Layers

3.3.1 Interface Layer

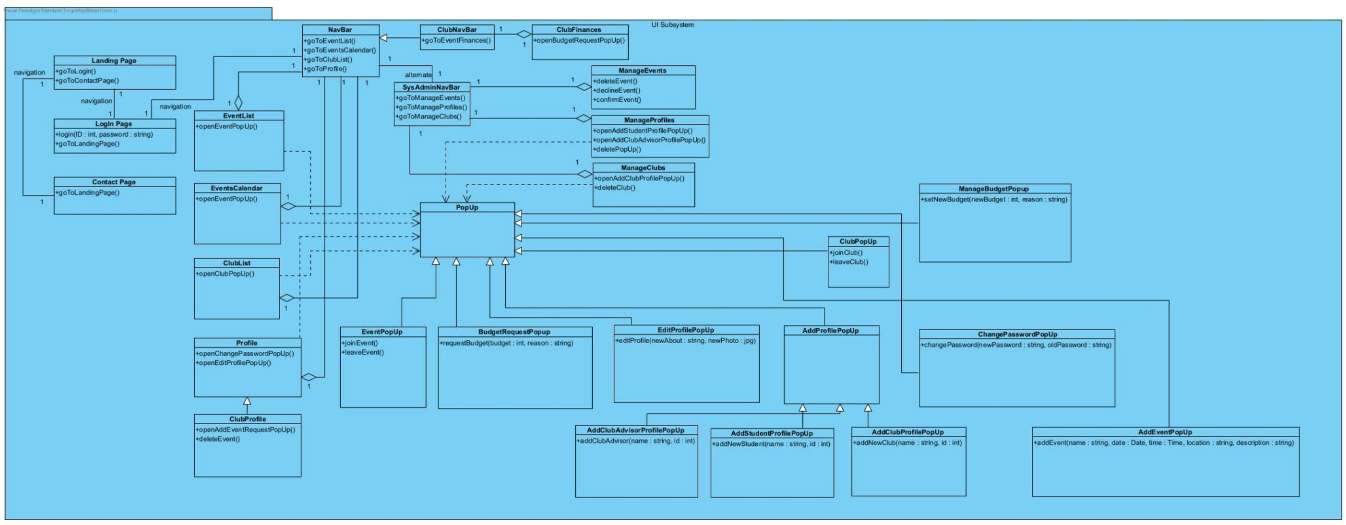

Figure 4. Interface Layer Class Diagram

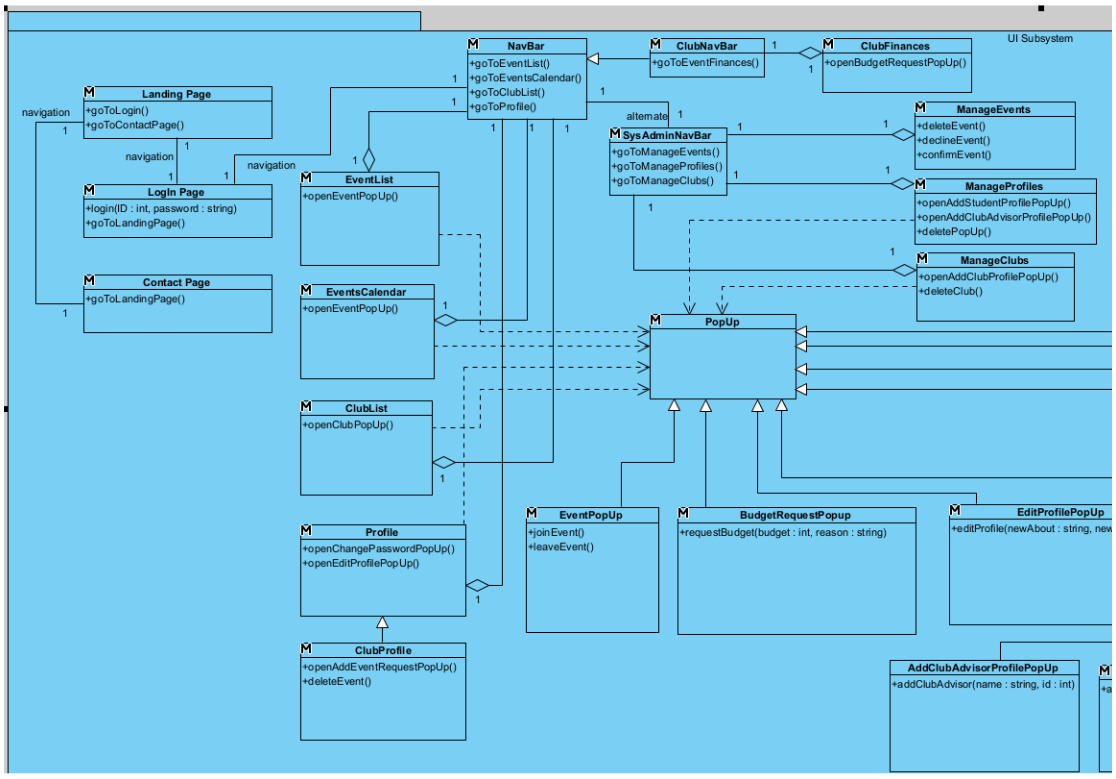

Figure 5. Left Side of the Interface Layer Class Diagram

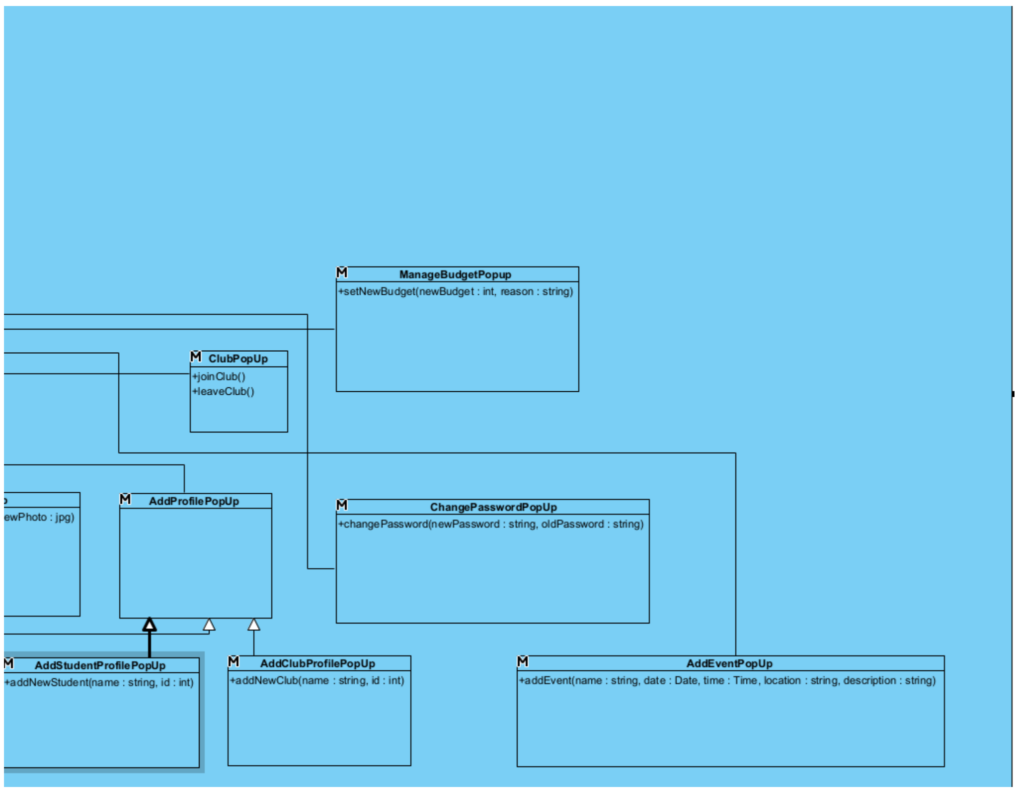
ManageBudgetPopup
+setNewBudget(newBudget : int, reason : string)
ClubPopUp
+joinClub()
+leaveClub()
ewPhoto : jpg)
AddProfilePopUp
ChangePasswordPopUp
+changePassword(newPassword : string, oldPassword : string)
AddStudentProfilePopUp
+addNewStudent(name : string, id : int)
AddClubProfilePopUp
+addNewClub(name : string, id : int)
AddEventPopUp
+addEvent(name : string, date : Date, time : Time, location : string, description : string)

Figure 6. Right Side of the Interface Layer Class Diagram

The Interface layer consists of all the front-end functionalities that users can use to communicate. Classes here are Javascript files containing React Component Objects.

3.3.2 Database Access Layer

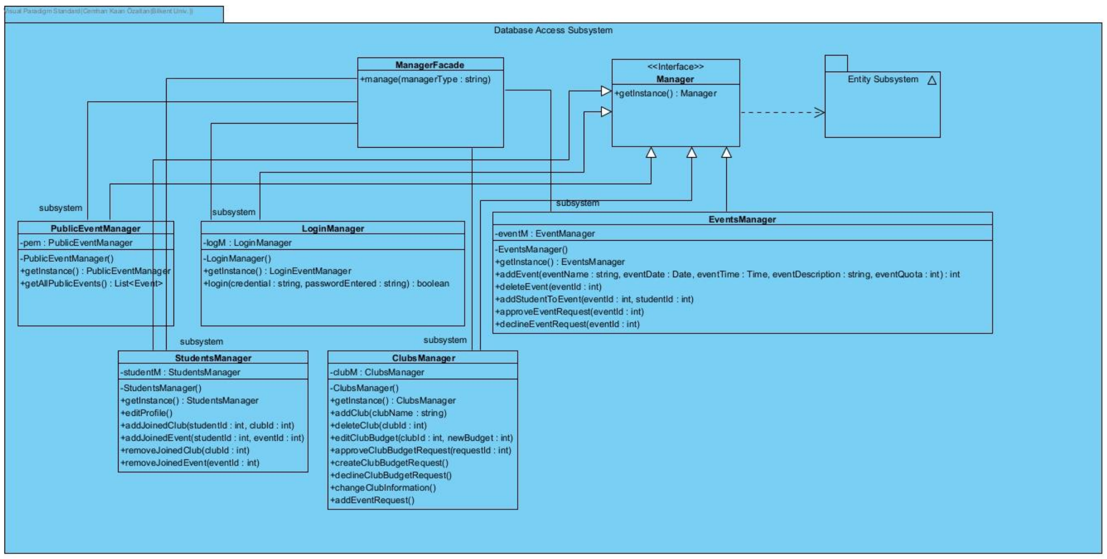
Visual Paradigm Standard(Cemhan Kaan Özaltan(Bilkent Univ.))
Database Access Subsystem
ManagerFacade
+manage(managerType : string)
<<Interface>>
Manager
+getInstance() : Manager
Entity Subsystem
subsystem
PublicEventManager
-pem : PublicEventManager
-PublicEventManager()
+getInstance() : PublicEventManager
+getAllPublicEvents() : List<Event>
subsystem
LoginManager
-logM : LoginManager
-LoginManager()
+getInstance() : LoginEventManager
+login(credential : string, passwordEntered : string) : boolean
subsystem
EventsManager
-eventM : EventManager
-EventsManager()
+getInstance() : EventsManager
+addEvent(eventName : string, eventDate : Date, eventTime : Time, eventDescription : string, eventQuota : int) : int
+deleteEvent(eventId : int)
+addStudentToEvent(eventId : int, studentId : int)
+approveEventRequest(eventId : int)
+declineEventRequest(eventId : int)
subsystem
StudentsManager
-studentM : StudentsManager
-StudentsManager()
+getInstance() : StudentsManager
+editProfile()
+addJoinedClub(studentId : int, clubId : int)
+addJoinedEvent(studentId : int, eventId : int)
+removeJoinedClub(clubId : int)
+removeJoinedEvent(eventId : int)
subsystem
ClubsManager
-clubM : ClubsManager
-ClubsManager()
+getInstance() : ClubsManager
+addClub(clubName : string)
+deleteClub(clubId : int)
+editClubBudget(clubId : int, newBudget : int)
+approveClubBudgetRequest(requestId : int)
+createClubBudgetRequest()
+declineClubBudgetRequest()
+changeClubInformation()
+addEventRequest()

Figure 7. Database Access Layer

The Database Access layer acts as a controller unit between front-end and firebase back-end. It includes the firebaseConfig file, which stores the database credentials.

3.3.3 Database Layer

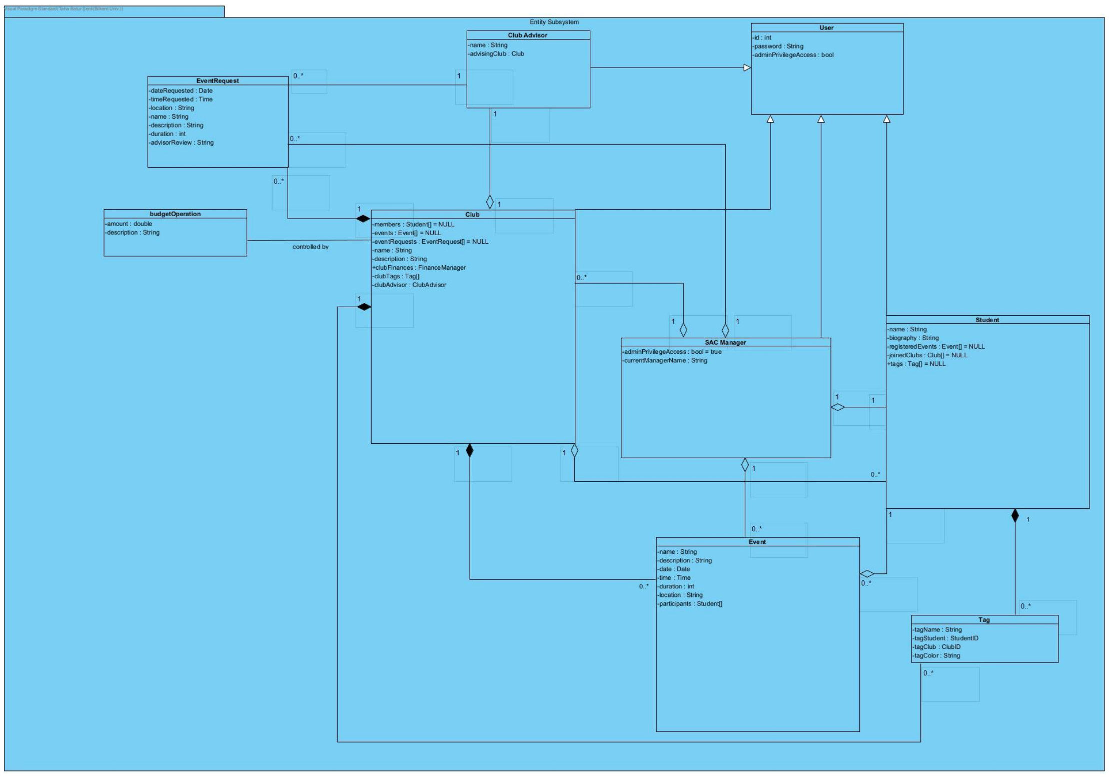

Figure 8. Entity Subsystem Diagram

3.4 Design Patterns

3.4.1 Singleton Design Pattern

The manager classes are responsible for establishing connection with the database and making data retrieval and storage more organized and usable during programming. These classes also act as controllers as they form the layer between the front-end UI elements and the backend entities and database communications. However, for each of the different manager types that deal with different categories of entities, there only needs to be one instance for establishing connection and managing data. More instances than one would not provide any more functionality and would only increase the runtime memory complexity of CluBil along with making its implementation more confusing. By using the singleton pattern and making the subclasses of the Manager interface implement a private constructor and a getInstance method, this goal is achieved.

3.4.2 Facade Design Pattern

The UI elements all depend on the mentioned manager classes which form a subsystem of their own due to the similarities in their functioning. Different UI components use different kinds of managers which depend on their respective categories of entities and retrieve different kinds of data. Because of this, accessing the management subsystem would be very difficult without the existence of a proper interface since many different classes would have to be accessed. With the usage of the Facade design pattern, a unified interface with the purpose of accessing the required manager objects is used. By doing this, changes in these said manager classes do not affect the rest of the program’s implementation and access becomes easier since the same unified interface is used by all UI components.

3.5 Packages

3.5.1 Internal Packages

These are the packages specifically developed for this application.

3.5.1.1 LandingPageManagement

This package is responsible for the management of the public data displayed on the landing page.

3.5.1.2 Login Management

This package manages the login authorization data of the user.

3.5.1.3 Events Management

This package is responsible for management of the event data displayed on the Events List, Events Calendar and Manage Events interfaces.

3.5.1.4 Club Management

This package manages the data of Club displayed on interfaces related with the club which are Profile, Club Finances, Manage Clubs, Club List

3.5.1.5 Students Management

This package is responsible for the management of the student data displayed on the Manage Profiles and Profile interfaces.

3.5.2 External Library Packages

3.5.2.1 reactjs-popup

This library is used to help develop accurate and user-friendly customizable popups for the various uses it has within our app

3.5.2.2 react-calendar

This library will help us formulate the monthly event calendars for students and clubs.

3.5.2.3 styled-components

This React library will help us design the UI element stylings in classic CSS syntax, but with more functionalities and cleaner code.

3.6 Class Interfaces

3.6.1 Interface Layer

3.6.1.1 Landing Page

Landing Page is a class that shows public events that are open for everyone.

Operations:
public goToLogin(): Goes to the login page when the button is clicked.
public goToContactPage(): Goes to the contact page when the button is clicked.

3.6.1.2 Login Page

Login Page is a class that shows a simple username and password field and a sign in button for both students and clubs.

Operations:
public login(int ID, string password): Submits the user data to the system to check whether credentials are valid or not.
public goToLandingPage(): Goes to the landing page when the button is clicked.

3.6.1.3 Contact Page

Contact Page is a class that shows a basic text field which contains an email. If the user has any trouble, they can contact us from that email.

Operations:
public goToLandingPage(): Goes to the landing page, when the button is clicked

3.6.1.4 NavBar

NavBar is a class that allows users to navigate among the pages.

Operations:
public goToEventList(): Goes to the eventlist page, when the button is clicked.
public goToEventsCalendar(): Goes to the events calendar page, when the button is clicked.
public goToClubList(): Goes to the club list page, when the button is clicked.
public goToProfile(): Goes to the profile page, when the button is clicked.

3.6.1.5 EventList

EventList is a class that shows the list of events according to the appropriate club.

Operations:
openEventPopUp(): Goes to the pop up, when the user wants to view an event in detail.

3.6.1.6 EventsCalendar
EventsCalender is a class that allows the events determined by the club to be seen on the calendar according to the time and day.

Operations:
openEventPopUp(): Goes to the pop up, when the user wants to view an event in detail.

3.6.1.7 ClubList
ClubList is a class that provides the determination of the list of the clubs that are approved by the university and that will or want to perform the events for students.

Operations:
openClubPopUp(): Goes to the pop up, when the user wants to view a club in detail.

3.6.1.8 Profile
Profile Page shows information about the user and users can edit the profile info and change their password through this page. Also users can see their notifications on this page.

Operations:
openChangePasswordPopUp(): Opens a pop up where users can change their password through this pop up.
openEditProfilePopUp(): Opens a pop up where users can change their profile information through this pop up.

3.6.1.9 ClubProfile
ClubProfile contains the profile of the club and detailed information about the club in this profile.

Operations:
openAddEventRequestPopUp(): Opens a popup when the club manager wants to add an event belonging to that club, it sends a request for club advisor's approval.
deleteEvent(): A manager of the club can delete an added event.

3.6.1.10 ClubNavBar
ClubNavBar is a class that has a specific navigation property for the clubs.

Operations:
public goToClubFinances(): Goes to the club finances page, when the button is clicked.

3.6.1.11 SysAdminNavBar
SysAdminNavBar is a navigation bar specified for the sysadmin. It alternates from the NavBar.

Operations:
public goToManageEvents(): Goes to the manage events page, when the button is clicked. SysAdmin can manage the events through this page.
public goToManageProfiles(): Goes to the manage profiles page, when the button is clicked. SysAdmin can manage the profiles through this page.
public goToManageClubs(): Goes to the manage clubs page, when the button is clicked. SysAdmin can manage the clubs through this page.

3.6.1.12 PopUp
PopUp is the page that allows users to do different activities.

3.6.1.13 EventPopUp
EventPopUp is a PopUp that allows users to join and to leave the event.

Operations:
public joinEvent(): If the user clicks the join button in the event pop up, the user's status will be updated as joined to that specific event.
public leaveEvent(): If the user clicks the leave button in the event pop up, the user's status will be updated as not joined to that specific event.

3.6.1.14 BudgetRequestPopUp
BudgetRequestPopUp is a PopUp that allows clubs to make budget requests.

Operations:
public requestBudget(int budget, string reason): If the user clicks the request budget change button in the budget request pop up, filled information will be sent to the database.

3.6.1.15 ClubFinances
ClubFinances is a page that allows clubs to see their financial situation.

Operations:
public openBudgetRequestPopUp(): It opens BudgetRequestPopUp, when the button is clicked.

3.6.1.16 ManageEvents
ManageEvents is a page where sysadmin can manage events.

Operations:
public deleteEvent(): If the sysAdmin clicks the delete event button in the manage events page, the event will be deleted.
public declineEvent(): If the sysAdmin clicks the decline event button in the manage events page, the event will be declined.
public confirmEvent(): If the sysAdmin clicks the confirm event button in the manage events page, the event will be confirmed.

3.6.1.17 ManageProfiles
ManageProfiles is a page where sysadmin can manage profiles.

Operations:
public openAddStudentProfilePopUp(): If the sysAdmin clicks the add student button in the manage profiles page, the pop up will be opened with the necessary fields for the adding student to a system.
public openAddClubAdvisorProfilePopUp(): If the sysAdmin clicks the add club advisor button in the manage profiles page, the pop up will be opened with the necessary fields for the adding club advisor to a specific club.
public deletePopUp(): If the sysAdmin clicks the delete student button in the manage profiles page, the pop up will be opened which informs the sysAdmin to be sure to delete the profile from a system.

3.6.1.18 ManageClubs
ManageClubs is a page where sysadmin can manage clubs.

Operations:
public openAddClubProfilePopUp(): If the sysAdmin clicks the add club button in the manage clubs page, the pop up will be opened with the necessary fields for the adding a club to a system.
public deleteClub(): If the sysAdmin clicks the delete club button in the manage clubs page, the pop up will be opened which informs the sysAdmin to be sure to delete the club from a system.

3.6.1.19 EditProfilePopUp
EditProfilePopUp is a PopUp that allows users to change their profiles.

Operations:
public editProfile(string newAbout, string newPhotoUrl): If the user clicks the edit profile button in the profile page, the pop up will be opened with the necessary fields for editing their own profile information.

3.6.1.20 AddClubAdvisorProfilePopUp
AddClubAdvisorProfilePopUp is a PopUp that allows sysadmin to add a new club advisor account.

Operations:
public addClubAdvisor(string name, int id): If the user clicks the add club advisor button, the filled fields' data will be sent to the database.

3.6.1.21 AddStudentProfilePopUp

AddStudentProfilePopUp is a PopUp that allows sysadmin to add a new student account.

Operations:
public addNewStudent(string name, int id): If the user clicks the add new student button, the filled fields' data will be sent to the database.

3.6.1.22 AddProfilePopUp

AddProfilePopUp is a PopUp that allows sysadmin to add a new account.

3.6.1.23 AddClubProfilePopUp

AddClubProfilePopUp is a PopUp that allows sysadmin to add a new club account.

Operations:
public addNewClub(string name, int id): If the user clicks the add new club button, the filled fields' data will be sent to the database.

3.6.1.24 ClubPopUp

ClubPopUp is a PopUp that allows students to join or leave the club.

Operations:
public joinClub(): If the user clicks the join button in the club pop up, the user's status will be updated as joined to that specific club.
public leaveClub(): If the user clicks the join button in the club pop up, the user's status will be updated as joined to that specific club.

3.6.1.25 ManageBudgetPopUp

ManageBudgetPopUp is a PopUp that allows clubs to request a change in their budgets.

Operations:
public setNewBudget(int newBudget, string reason): If the user clicks the set new budget button in the manage budget pop up, the club's budget change request is sent to the sysAdmin to be accepted.

3.6.1.26 ChangePasswordPopUp

ChangePasswordPopUp It allows the user to replace an old password with a new one.

Operations:
public changePassword(string newPassword, string oldPassword): If the user clicks the change password button in the pop up, the user's old password will be replaced with a new one.

3.6.1.27 AddEventPopUp
AddEventPopUp opens a popup to add events according to its name, time, location and description.

Operations:
public addEvent(string name, Date date, Time time, string location, string description): If the user clicks add event button, event is added according to name, time, location, date and description fields filled in the pop up by the user.

3.6.2 Database Access Layer
3.6.2.1 PublicEventManager
PublicEventManager is a controller that gets all public events from the database.

Attributes:
private PublicEventManager pem: pem is the object name of the PublicEventManager.

Operations:
private PublicEventManager(): This is the constructor of the PublicEventManager.
public getInstance(): This operation enables us to use a singleton design pattern.
public getAllPublicEvents(): This operation enables us to fetch all the public events from the database.

3.6.2.2 LoginManager
LoginManager is a controller that.gets the login data of the user from the database and controls the credentials.

Attributes:
private LoginManager logM: logM is the object name of the LoginManager.

Operations:
private LoginManager(): This is the constructor of the LoginManager.
public getInstance(): This operation enables us to use a singleton design pattern.
public login(string credential, string passwordEntered): This operation enables users to login to the system. It’s purpose is to make a connection between the database and user interface.

3.6.2.3 StudentsManager
StudentsManager is a controller that manages the profile, clubs and events of the students and saves new properties to the database.

Attributes:
private StudentsManager studentM: studentM is the object name of the StudentsManager.

Operations:
private StudentsManager(): This is the constructor of the StudentsManager.
public getInstance(): This operation enables us to use a singleton design pattern.
public editProfile(): This operation enables users to edit their profile. It’s purpose is to make a connection between the database and user interface.
public addJoinedClub(int id, int clubId): This operation enables users to join a club. It’s purpose is to make a connection between the database and user interface.
public addJoinedEvent(int id, int eventId): This operation enables users to join an event. It’s purpose is to make a connection between the database and user interface.
public removeJoinedClub(int clubId): This operation enables users to leave from a joined club. It’s purpose is to make a connection between the database and user interface.
public removeJoinedEvent(int eventId): This operation enables users to leave from a joined event. It’s purpose is to make a connection between the database and user interface.

3.6.2.4 ClubsManager
ClubsManager is a controller that manages the profile, clubs and events of the clubs and saves new properties to the database.

Attributes:
private ClubsManager ClubM: ClubM is the object name of the ClubsManager.

Operations:
private ClubsManager(): This is the constructor of the ClubsManager.
public getInstance(): This operation enables us to use a singleton design pattern.
public addClub(string clubName): This operation enables users to add a new club. It’s purpose is to make a connection between the database and user interface.
public deleteClub(int clubId): This operation enables users to delete a new club. It’s purpose is to make a connection between the database and user interface.

public editClubBudget(int clubId, int newBudget): This operation enables users to edit existing budget by using the user's clubId. It’s purpose is to make a connection between the database and user interface.
public approveClubBudgetRequest(int requestId): This operation enables users to approve club budget request for each club existing according to its requestId. It’s purpose is to make a connection between the database and user interface.
public createClubBudgetRequest(): This operation enables users to create a new budget request to determine budget for each event. It’s purpose is to make a connection between the database and user interface.
public declineClubBudgetRequest(): This operation enables users to decline the club's budget request in case of taking precautions. It’s purpose is to make a connection between the database and user interface.
public changeClubInformation(): This operation enables users to change the club information necessary for the students to be informed. It’s purpose is to make a connection between the database and user interface.
public addEventRequest(): This operation enables users to enable the creation of a request to be approved by the club advisor so that the events to be held by the club can be accepted. It’s purpose is to make a connection between the database and user interface.

3.6.2.5 EventsManager

EventsManager is a controller that gets and manipulates the data of the event according to the activities of the user.

Attributes:
private EventManager eventM: eventM is the object name of the EventManager.

Operations:
private EventsManager(): This is the constructor of the EventsManager.
public getInstance(): This operation enables us to use a singleton design pattern.
public addEvent(string eventName, Date eventDate, Time eventTime, string eventDescription, int eventQuota): This operation enables users to add an event to the system. It’s purpose is to make a connection between the database and user interface.
public deleteEvent(int eventId): This operation enables users to delete an event. It’s purpose is to make a connection between the database and user interface.
public addStudentToEvent(int eventId, int studentId): This operation enables users to add another user to a specific event. It’s purpose is to make a connection between the database and user interface.
public approveEventRequest(int eventId): This operation enables users to approve an event request which comes from another user. It’s purpose is to make a connection between the database and user interface.

public declineEventRequest(int eventId): This operation enables users to decline an event request which comes from another user. It's purpose is to make a connection between the database and user interface.

3.6.2.6 ManagerFacade

ManagerFacade is a unified interface class that allows the UI elements to interact with the database layer and the entity objects using the facade design pattern. This class uses all of the available management subclasses in order to provide the needed data to different UI components.

Operations:
public manage(string managerType): This operation enables the system to select which type of management mode is wanted from the user.

3.6.2.7 Manager

Manager interface's method, which is necessary for the usage of the singleton design pattern, is inherited by all manager classes under it. Using this method, either an instance of these classes can be created or the existing one can be retrieved. All of the manager classes also have private constructors for the implementation of the singleton design pattern.

Operations:
public getInstance(): This operation enables us to use a singleton design pattern.

3.6.3 Database Layer

3.6.3.1 EventRequest

EventRequest is a class that stores the contents of the event request object that will be stored in the database to be added or edited.

Attributes:
private Date dateRequested: The date of the Event Request in dd/mm/yyyy format.
private Time timeRequested: The time of the Event Request in hh/mm/ss format.
private String location: The Event Request's location (i.e. Mithat Çoruh Amfi).
private String name: Event Request's name.
private String description: A brief description of the Event Request.
private int duration: The length of the Event Request.
private String advisorReview: A review of the Event Request by the Club Advisor.

3.6.3.2 BudgetOperation

With BudgetOperation, the amount of the budget determined for the club and the event is arranged and an explanation is written for the amount to be edited/added.

Attributes:
private double amount: The amount of budget (request or approved) for each club.
private String description: The detailed description of the reason why the club requested a budget change.

3.6.3.3 ClubAdvisor
ClubAdvisor is an entity that stores the name and club property of the club advisor in the database. It can be edited or added.

Attributes:
private String name : The name of the Club Advisor.
private Club advisingClub: the Club object that the Club Advisor is advising.

3.6.3.4 Club
Club is used to store the properties of the club object to be edited or added in the database.

Attributes:
private Student[] members: The list of the Club's members.
private Event[] events: The list of the Club's approved & approaching events.
private EventRequest[] eventRequests: The list of the Club's event requests that are pending for approval by the SAC.
private String name: The Club's name.
private String description: A brief description of the Club, it's activities, focus etc.
public FinanceManager clubFinances: The finance management tab of the Club, which can handle functions such as altering the budget or allocating budget for events.
private Tag[] clubTags: The tags of the Club's regular members and board members.
private ClubAdvisor clubAdvisor: The Club's advisor is kept as a private ClubAdvisor object.

3.6.3.5 User
User is used to store the properties of the user object to be edited or added.

Attributes:
private int id: The User's Bilkent ID.
private String password: The User's password.
private bool adminPrivilegeAccess: A boolean value to check whether or not the user has access to admin privileges.

3.6.3.6 SAC Manager
Login Page shows a simple username and password field and a sign in button for both students and clubs

Attributes:
private bool adminPrivilegeAccess = true: The status of the SAC Manager's access to administrative privileges, which is set to true.
private String currentManagerName: The name of the current SAC Administrator.

3.6.3.7 Event
Event is used to store the properties of the approved and approaching Event object to be added or edited in the database.

Attributes:
private String name: The Event's name.
private String description: A brief description of the Event.
private Date date: The date of the Event in dd/mm/yyyy format.
private Time time: The time of the Event in hh/mm/ss format.
private int duration: The duration of the Event (i.e. 69 minutes).
private String location: The location of the Event (i.e. Mithat Çoruh Amfi).
private Student[] participants: The list of participants for the given Event.

3.6.3.8 Student
Student is used to store the properties of the student object to be added or edited in the database.

Attributes:
private String name: The name of the Student.
private String biography: The biography of the Student.
private Event[] registeredEvents: The events that the Student has registered to.
private Club[] joinedClubs: The clubs that the Student has joined.
private Tag[] tags: The tags of the Student which indicate their position in the club. (i.e. ACM Active Member) as well as their name, ID and name color.

3.6.3.9 Tag
Tag is used to store the properties of the tags in the database to be added or edited.

Attributes:
private String tagName: The name of the Tag.
private StudentID tagStudent: The student who is the owner of the tag.
private ClubID tagClub: The tag of the student within a club (i.e. Active Member).
private String tagColor: The color of the name in the Tag.

4 Improvements and Summary

4.1 General

- The font of the report has been changed to Times New Roman and puntos have been adjusted accordingly.
- The explanation of all classes is added.

4.2 Design Goals

- We kept our original design goals of usability and maintainability, but expanded our reasoning for choosing them and how we implemented them.

4.3 Boundary Conditions - Initialization

- Initialization process has been refined. Added needed commands and steps to deploy the application from scratch.

4.4 Final Object Design

- Added a new actor called “Club Advisor” and reflected the necessary changes on the Final Object Design.
- The associations and multiplicities between objects were adjusted and explained.

4.5 Interface Layer

- Added new objects associated with the new actor “Club Advisor”.
- The associations and multiplicities between objects were adjusted and explained.

4.6 Database Access Layer

- Both the singleton and facade design patterns were implemented in certain classes contained in this layer.
- ManagerFacade class, which is a unified interface for the implementation of the facade design pattern was added.

- Manager class was turned into an interface for better design.
- Singleton design pattern was implemented for the manager subclasses using private constructors and a getInstance method inherited from their parent interface.
- Manager classes' relation with the Entity Subsystem was added here.

4.7 Database Layer

- A diagram for the Entity Subsystem located in the Database Layer was added.
- In the Entity Subsystem, the entity classes used for the backend of the CluBil system are given.

5 Glossary and References

SAC - Student Activities Center

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2] https://docs.microsoft.com/en-us/power-platform/admin/web-application-requirement s