Rate My Club

Team Chilly Tiger

Drew Letvin: 405382898

Pranav Maddali: 905338332

Michael Zhan: 805384414

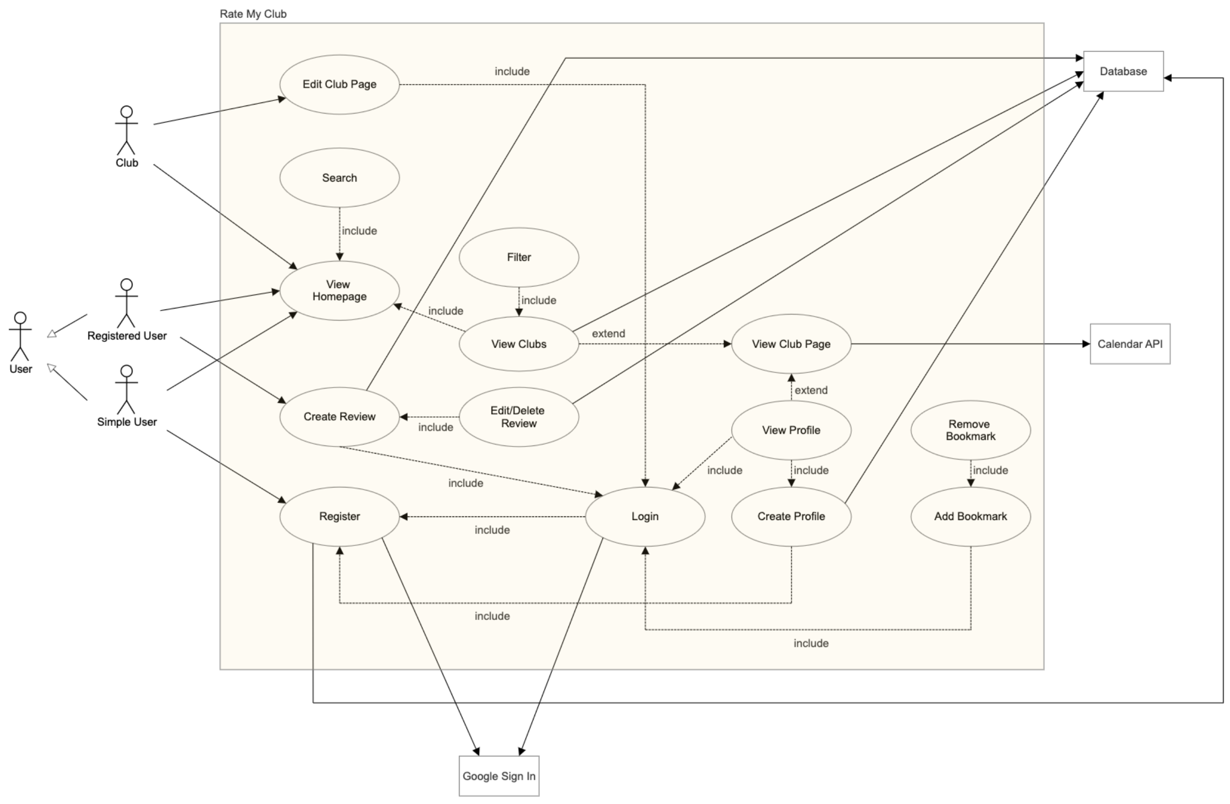
Nitya Simhadri: 105417192

Sae Tsunawaki: 105334390

GitHub Repository: https://github.com/mikezhan88/Rate-My-Club

Section 1: Analysis Description

Use Case Diagram

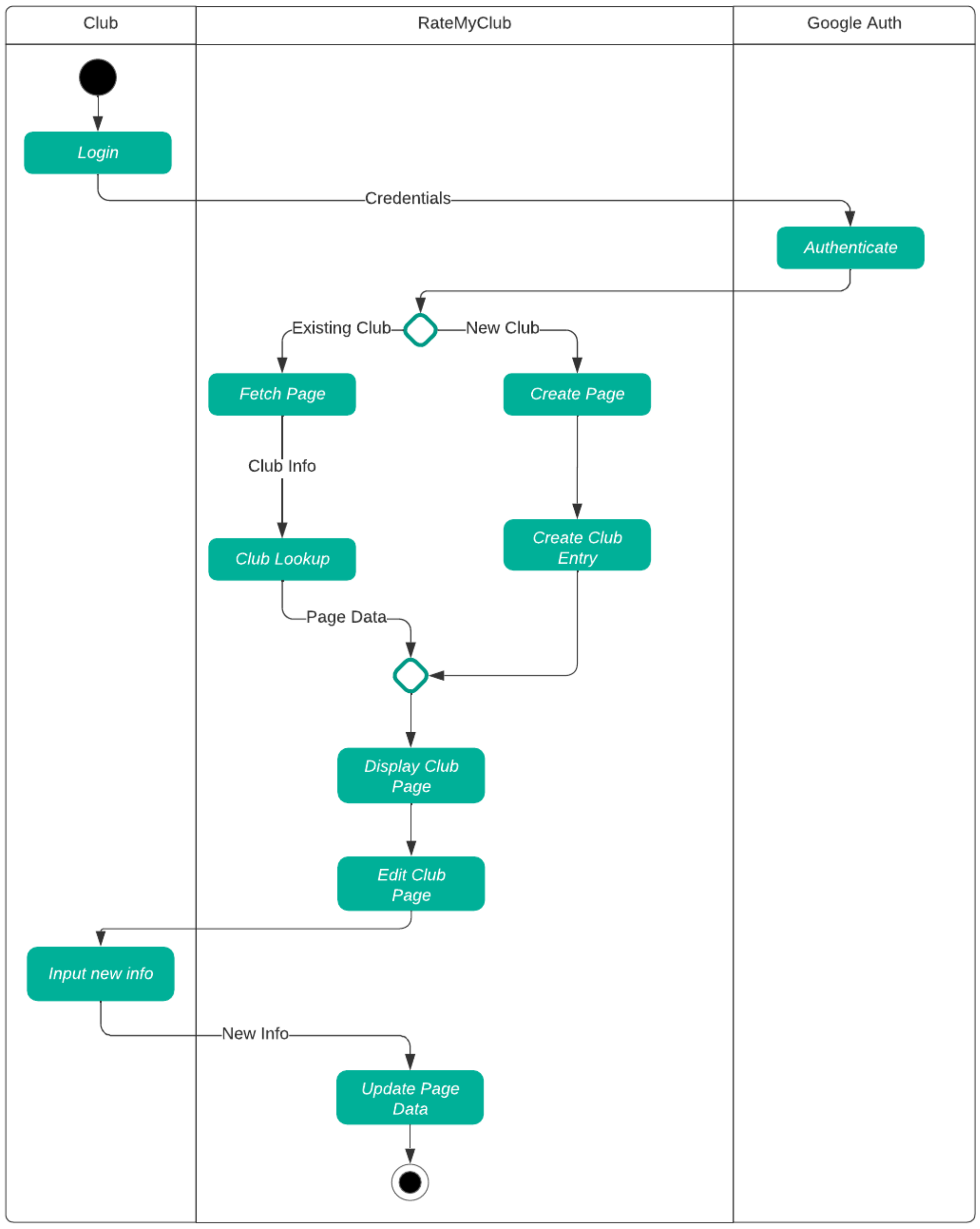


The use case diagram above highlights the high-level relationship between actors (users and clubs) and the service (Rate My Club).

A user can either be a registered user or a simple user. A registered user can login, view clubs, write/edit reviews, and bookmark clubs, while a simple user can only view clubs until they register/create a profile. When viewing clubs, users can filter their searches, and clicking on a club takes a user to the club page, which displays further information about the club. Clubs will also make a profile, and they can edit their club page to make updates and announcements about meetings/events.

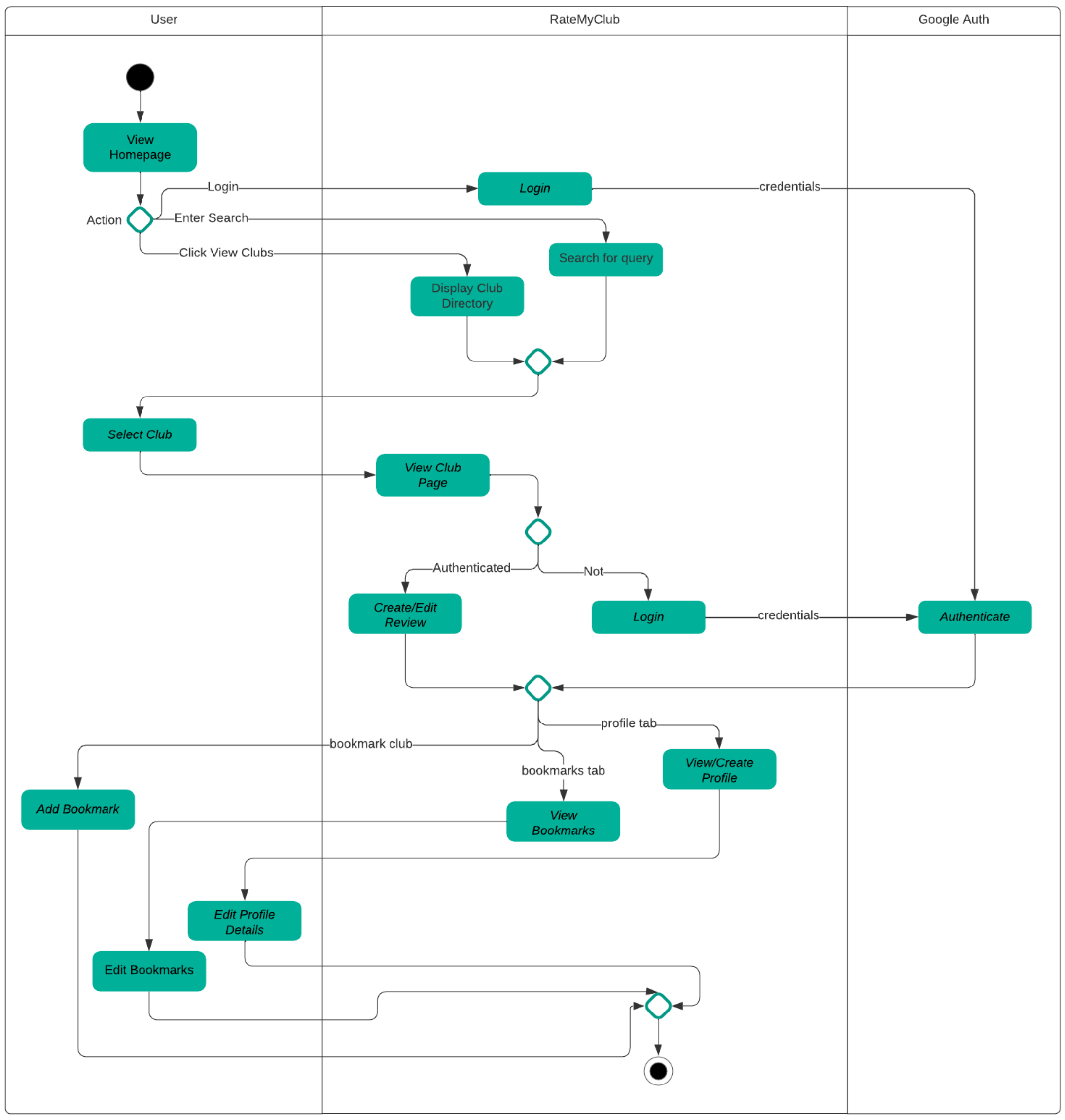
Registering and logging in will utilize Google Sign-in, and each club can post information about club meetings/events through Google Calendar API. Information about user/club login credentials, list of clubs and their reviews will be stored in the Rate My Club Database.
Activity Diagram

Club Diagram



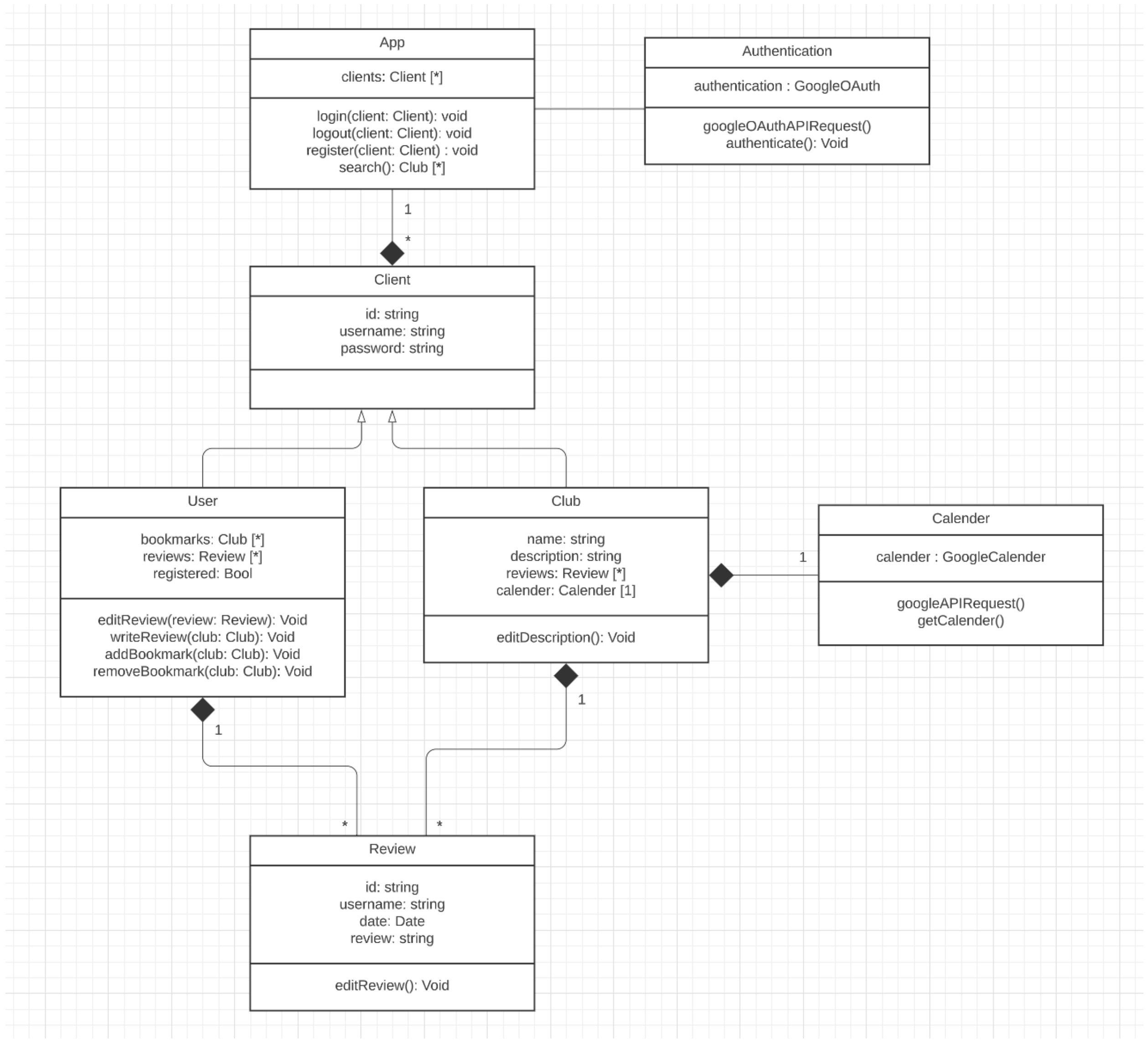
The above diagram illustrates the most common use case for a club user. The user starts by logging in, as is required by those acting as a club. This is done by sending over credentials to the Google authentication API which uses a gmail account to log into our application. If the page exists it will be fetched from our database. If this is the club's first time logging in then a blank template page will be created for them and stashed in our database until edits are made. This page is then displayed and the user is given the option to edit. Once edits are made the new info is sent to the database and the page is updated to reflect the changes.

User Diagram



This next diagram shows the flow for our primary user, a student searching for clubs on campus. Upon viewing the homepage the user will see three main avenues for continuing: loggin in, searching for a club, or accessing the directory of all clubs registered to the application. By either searching for or selecting a club the user will then end up on a club's page where that club's info and reviews can be seen. If log in was not already completed then here they can log in and add a review to this club's page themselves. Now that they are logged in they also have access to their other profile features and can view/edit their bookmarks and profile.

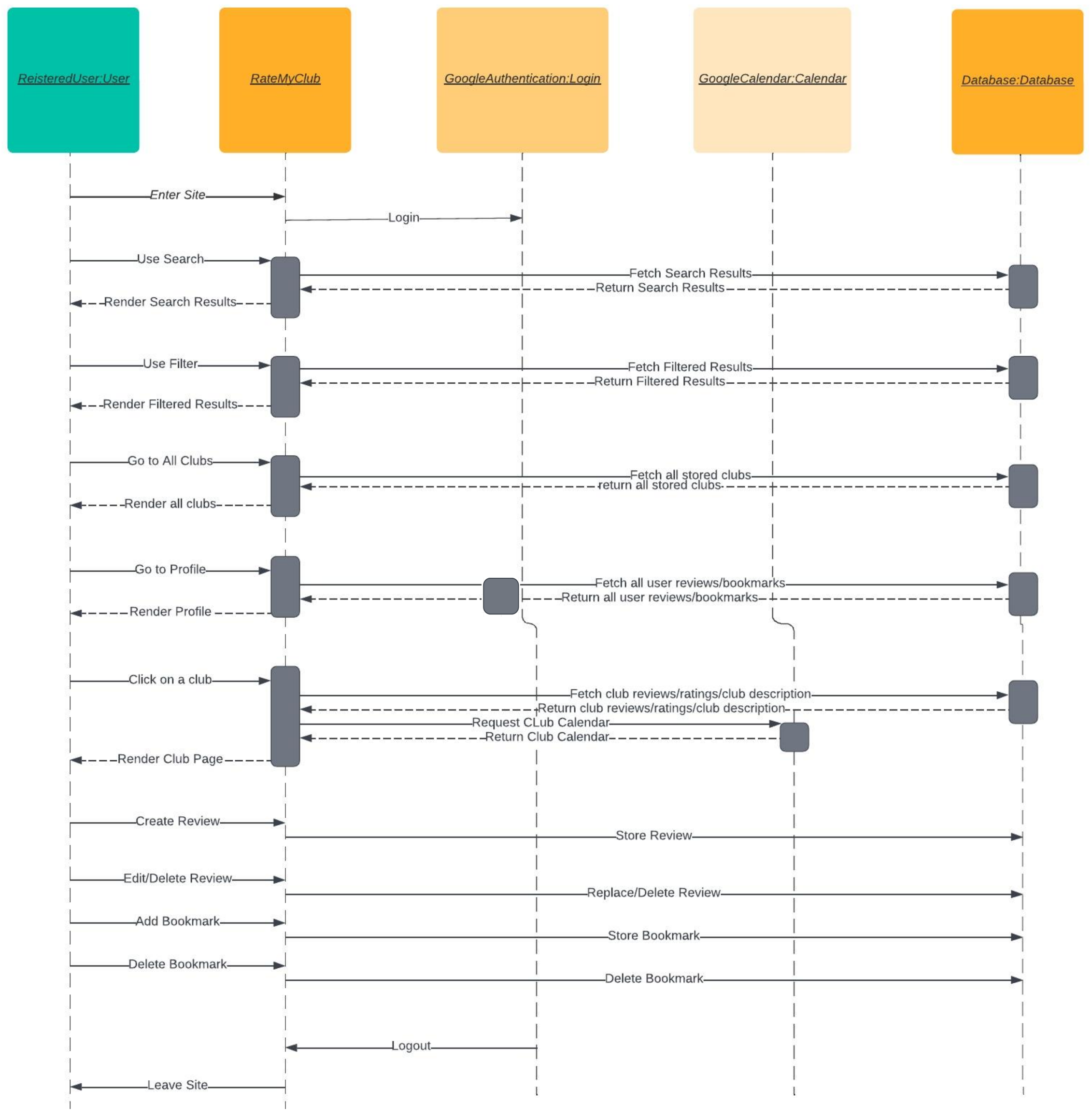
Class Diagram

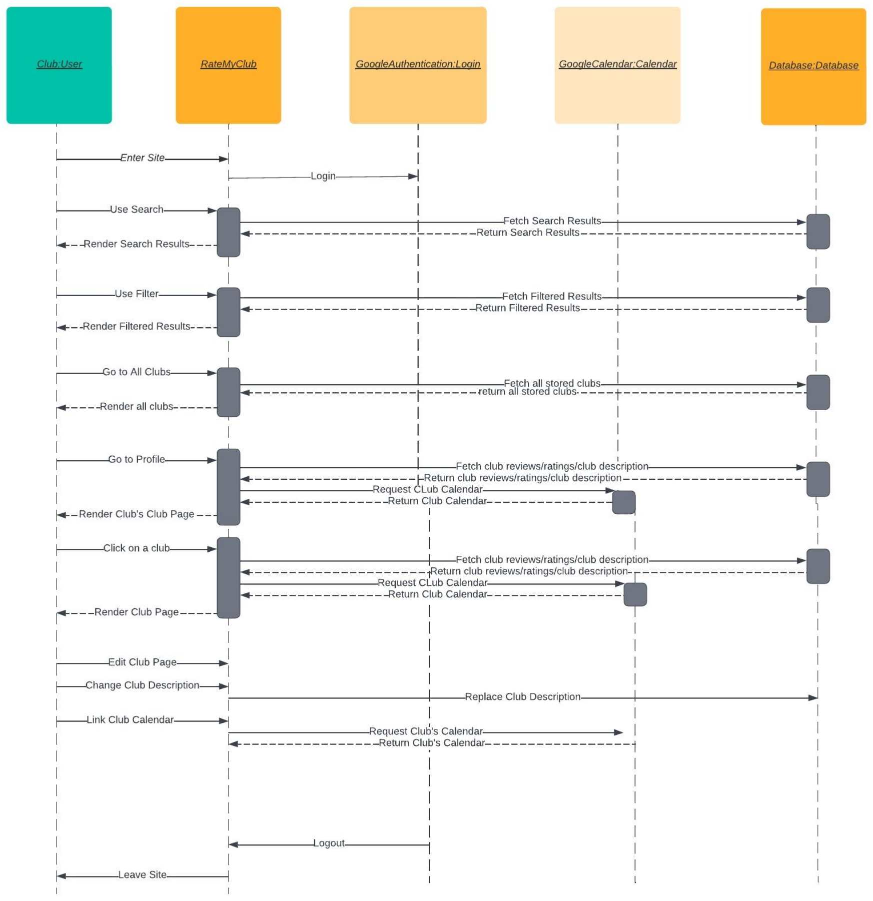


The class diagram shows the main classes in the application, and how they will interact with one another. We will have a Client class which stores its unique id, a username, and password. The User class and the Club class will inherit from the Client class. A User has a list of bookmarked Clubs, and a list of Reviews. We will keep a boolean value to check if a User is registered or not.

A Club will have a description attribute that they are able to edit, as well as a list of Reviews and a Calendar object. A Review class holds a unique id, the username of the author, a date, and a method to edit a review (if the User is the author). Calendar and Authentication classes will use the Google API.

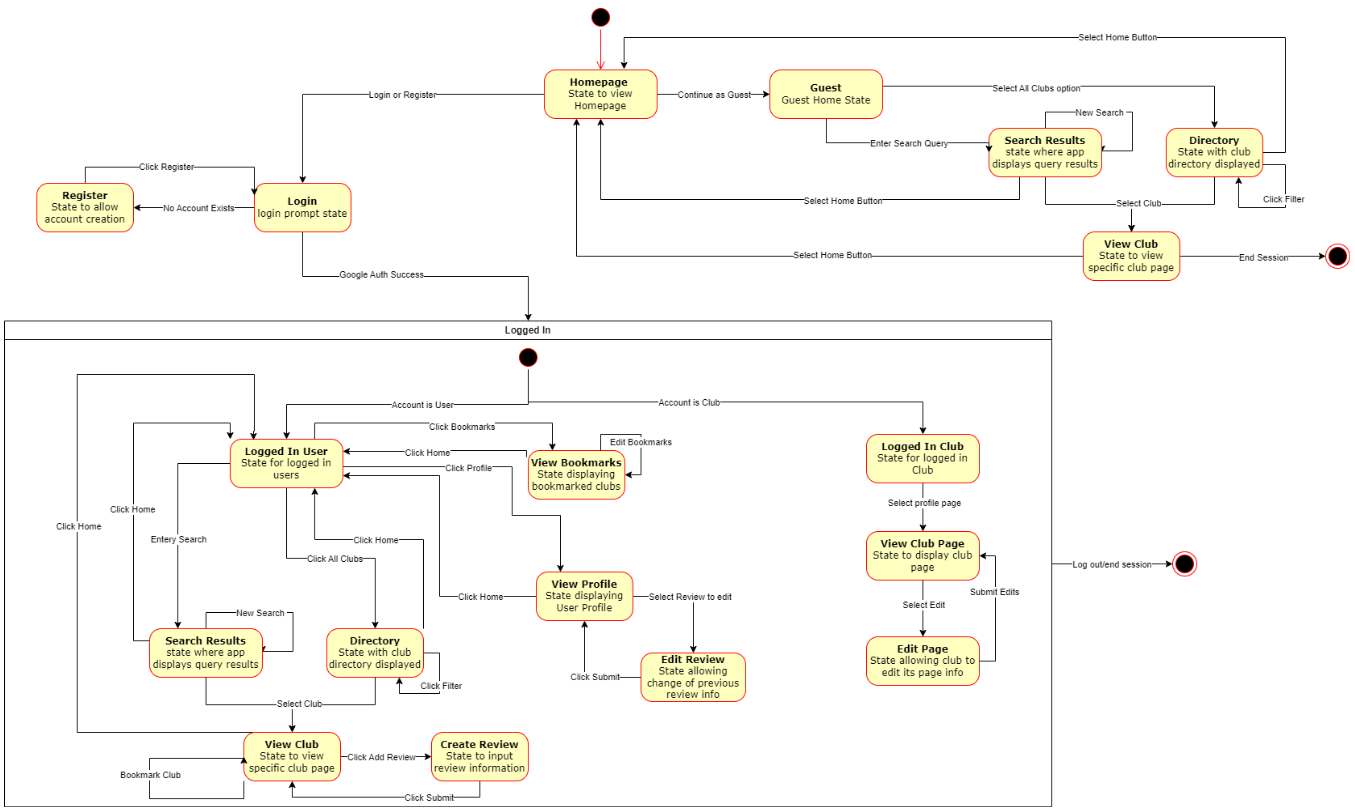
Sequence Diagram





As shown above, we have two sequence diagrams, the first one showing the flow for a registered user and the second for the flow of a registered club. We can see that there is much overlap in the functionality available for both a registered user and a club. Both are able to login and use functions such as search, filter, view the club directory, and view a clubs page. The biggest distinctions are that a registered user can view their own profile with their saved bookmarks for clubs and reviews they have created. Club users on the other hand are redirected to their club page when clicking on profile where they have the ability to edit their own club page by linking their google calendar or changing their club description. We can see that both diagrams show that certain actions by the user will have responses such as using the search bar which will render search results by fetching them from the database which also has the response of returning the search results from the database. Many of the other actions that a user can take also follow a very similar logic flow.

State Diagram



The State diagram above, while one diagram, has two distinct sections. One resides within the "Logged In" state while the other is the area not enclosed within this super-state. The area outside the "Logged In" section defines the behavior of our application for users who wish to remain guest users. The guest user flow is a simplified version of a logged in user flow. The main action that can be taken without credentials is viewing club pages and their associated reviews. The other states simply detail how a user can navigate, either by searching or combing through a directory, to the desired club. If a guest user wishes to do anything more than view they must follow the log in/register path and enter the "Logged In" section.

Within the "Logged In" section two sets of behavior are defined, a club member and a student user. Logged in student users gain the functionality of bookmarking clubs to a list that can be revisited later, adding reviews to clubs, and being able to see/edit reviews they've left in the past via their profile page. It's worth noting that "Logged In" is reached with a successful call to the

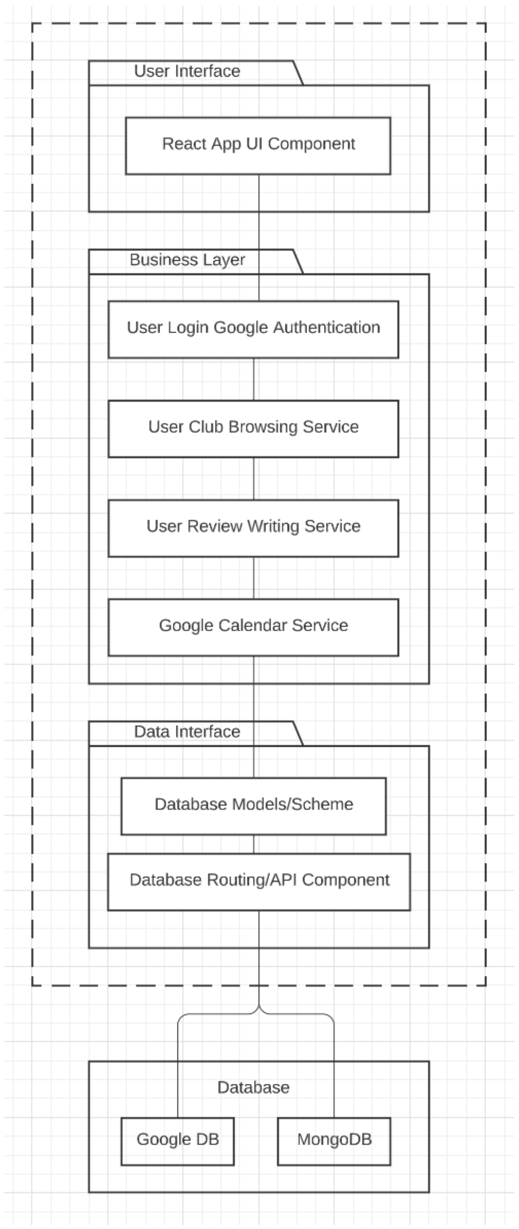
Google authentication API as users will be able to utilize their school gmail accounts as ready made credentials. As a club member you no longer will browse for other clubs but instead have a page of your own which can be customized once logged in with an associated club account. While there are endpoints in the state diagram, these mainly represent what might be common exit points to the application as nearly all states provide loops back to other pages where the user can continue interacting with our application. Exit may truly be achieved in most/all states.

We found it necessary to add a distinction between logged in and guest users as a type of safeguard to poor behavior and review spam/bombing. While it is a minimal level of verification it will provide some form of accountability. Additionally we don't want to allow anonymous creation of clubs and thus require clubs to be logged in. We feel that it may even be worthwhile to manually verify club accounts due to the limited scaling in new club creations.

Section 2: Architecture Description

Application Landscape Pattern

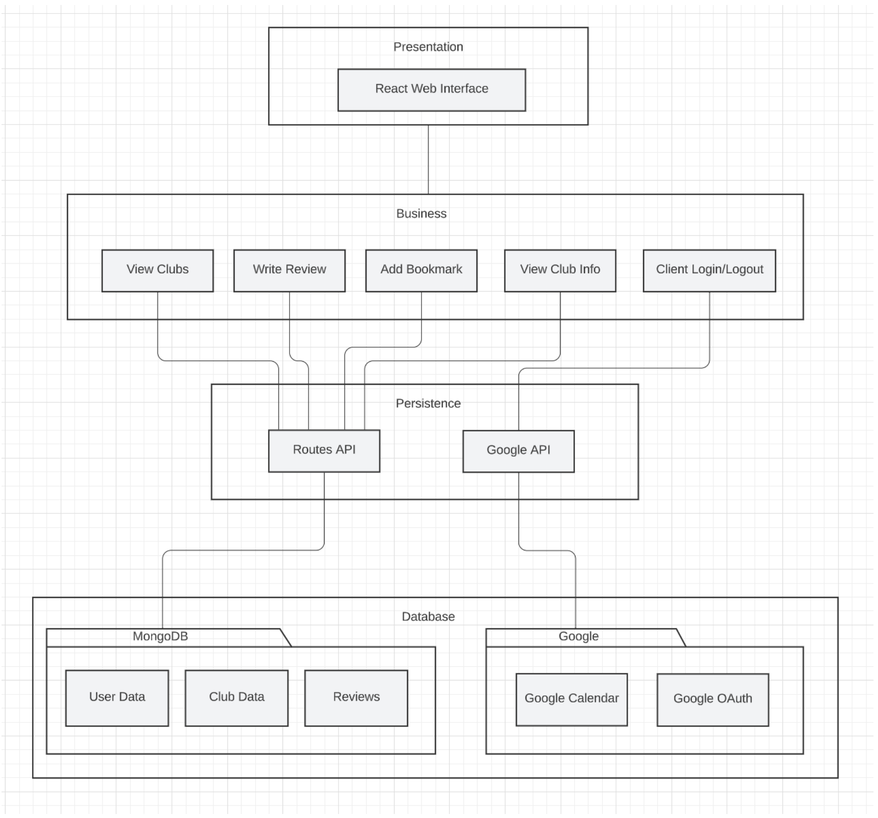
For our application landscape pattern, we decided to use a monolith architecture. Monolith makes the most sense for our application, since it is ideal for short development time and a limited scope application. In our case, we only have a few weeks to deploy our application, and with most team members having to learn new technology, we think a simple architecture like monolith would be best. Infrastructure related concerns can be omitted, and we can focus on the functionalities of our application, rather than figuring out how to deploy a complex application.



Application Structure Pattern

For our application structure pattern, we decided to use a layered model. A layered model works well with our monolith architecture, as it is easy to recognize and it separates concerns. Modules with similar functionalities are separated and organized into horizontal layers. In our case, these layers are Presentation, Business, Persistence, and Database.

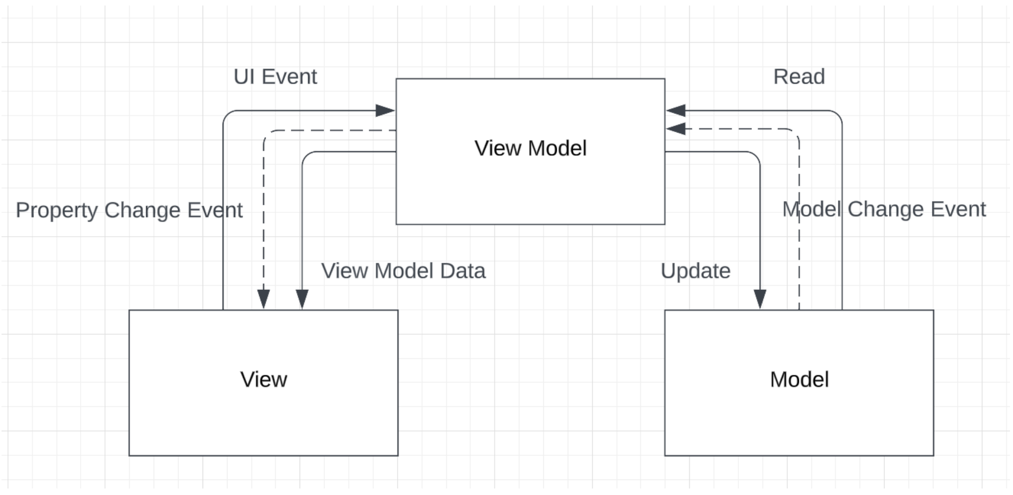
The Presentation layer holds the React Web Interface; this is where the user interacts with the application. The Business layer holds the main business logic; in our case, these would be what the business offers to our clients. Clients are able to view clubs, write reviews, add bookmarks, view detailed club information, and log in/out and register. The Persistence layer connects the Business layer and the Database layer. To access our MongoDB database, we need to use a routes API, and to handle authentication and the calendar service, we need to use a Google API. The final Database layer will hold all our data, separated between MongoDB, which will hold user/club data and all the reviews that have been written. The Google DB will hold client credentials, and Google Calendar data for each club.



User Interface Pattern

We are using React to implement all of the aspects with our frontend. React is simply a view layer, however the user interface pattern that we believe will work the best is MVVM. This works best with React because React breaks everything into separate components. Each component in the app has different functionalities and behaviors. We chose MVVM because MVVM “couples” the view and the logic pertinent to the view, by handling it in the same interface but also separates and maintains the logic as a completely separate entity of the view.

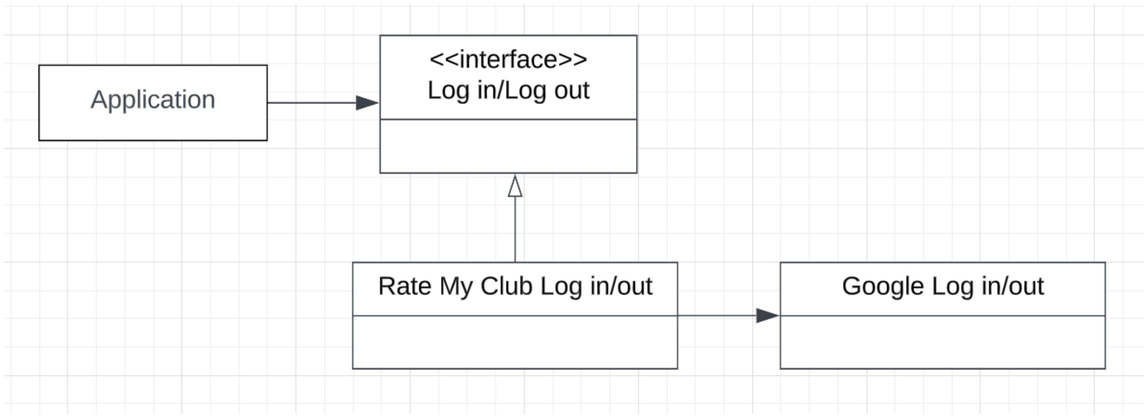
That allows us to have more control and to be able to perform more efficient and case specific testing. We can also clearly understand how the code works. Further, separating the code in the way that MVVM does, makes the UI code simpler because the only responsibility of the UI is to display the data, while the viewmodel is responsible for all the data manipulation. In our implementation, the View object will take in the ViewModel as a prop and the View Model will maintain all the logic for the view.



Section 3: Design Description

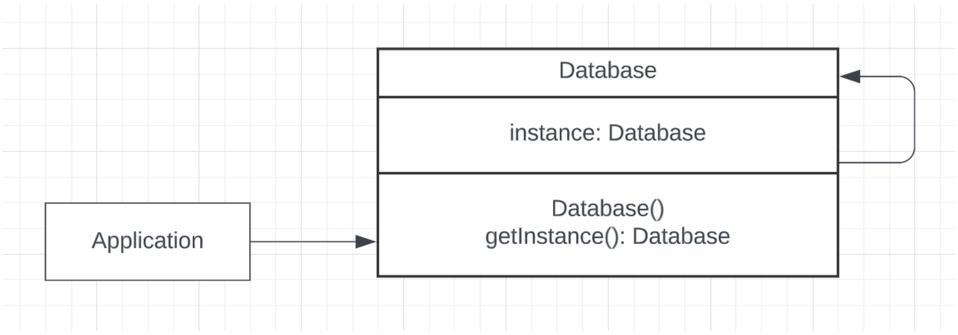
Authorization (Adapter Pattern)

For our login and logout functionalities, we decided to use an adapter pattern. As shown in the diagram below, our Rate My Club login/logout acts as an adapter to the Google Authentication. Adapter pattern works well here, since there is a preexisting Google Authentication login/logout component that we wanted to leverage, but we cannot integrate it smoothly into our application. By creating a middle layer between our application and Google Authentication, we essentially have an adapter that acts as a wrapper to the 3rd party login component, which communicates information back to our application component.



Database (Singleton Pattern)

Our database should follow the singleton pattern. In our application, we must have a single database object which is shared by different parts of the program. Since a singleton pattern disables all other means of creating a database object, we can be sure that every object in our application accesses the same database.



UI (Mediator Pattern)

We decided to use the mediator pattern for the UI. The App class that we will have will act as the mediator object for all other components that will be on the screen. There are many-to-many dependencies between the components that can be handled by the central App class. The mediator pattern also reduces coupling between the several components. Examples of components include Landing Page, Navigation Bar, Login/Register Page, and Writing a Review Page. These components will have a render method that overrides the interface method.

Section 4: Implementation Status

Epic Status

We are currently in the middle of Sprint 1. We have dedicated Sprint 1 to ensure that we have the proper infrastructure at the beginning of our process to ensure more efficient coding and less design related problems down the line. As a part of Sprint 1, we have updated our epics and user stories to more precisely and concretely layout our epics, and created a figma to layout the frontend interface. We also have spent our sprint determining what database (MongoDB) and user interface we want to use (React). At this point, our github holds all of our diagrams to aid us with code architecture. We will be finished with this Sprint on Sunday, Feb 19th and at that point our Github will hold our fully updated epics, diagrams and the barebones for the React landing page and we will have implemented a Heroku pipeline for code structure. At this point, we are 70% completed with Sprint 1- diagrams are completed, epics and stories are documented and some still need to be fully fleshed out. As discussed with Akshay, we are behind with our progress but have set up proper infrastructure and properly communicated as a team to ensure more accelerated progress.

Successes/Challenges

Our biggest challenge has been being able to dedicate cohesive time to spend on Sprint 1- due to the height of midterm season. We cannot speak for any technical successes or challenges, however we have found success in finding tools to help us organize our code like lucidcharts for diagramming and figma to display the frontend interface. We also have implemented weekly schedules and a more defined calendar to ensure that we will solve our time management challenge.

Libraries/Frameworks/Components

We are using React UI framework along with MongoDB as a database to implement our frontend and to bridge the frontend with the backend. We are also implementing two Google APIs- Google Calendar API and and Google Login API (OAuth) and will utilize them within React to display the club calendar and in our backend to authenticate users. So far, we haven't found any tradeoffs as we haven't explored many options yet, but we do see using the Google APIs as a benefit because that would allow us to standardize the login for all users and would lessen the task of creating and storing user profiles.

Special Techniques

As mentioned, we have not uncovered many special techniques yet, however we intend to implement Heroku for another layer of code management. We also have been using figma to visualize the user interface and also determine what functions are needed and how they relate to others.

UI Mockup

Figure 1

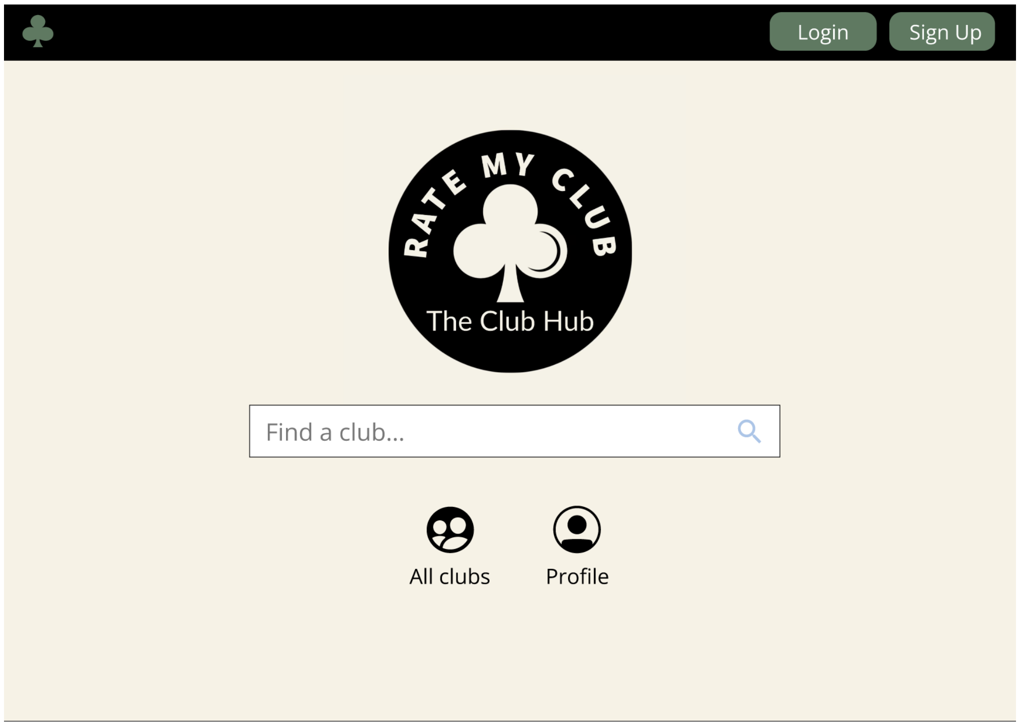


Figure 1 displays the landing/home page. Any user can search for a club, and registered users can visit their profile. Note: for all the pages if a user is not signed in, we will display the option to login or register an account. If a user is registered, we will display a profile icon that leads to their profile page as seen in the mockups below.

Figure 2

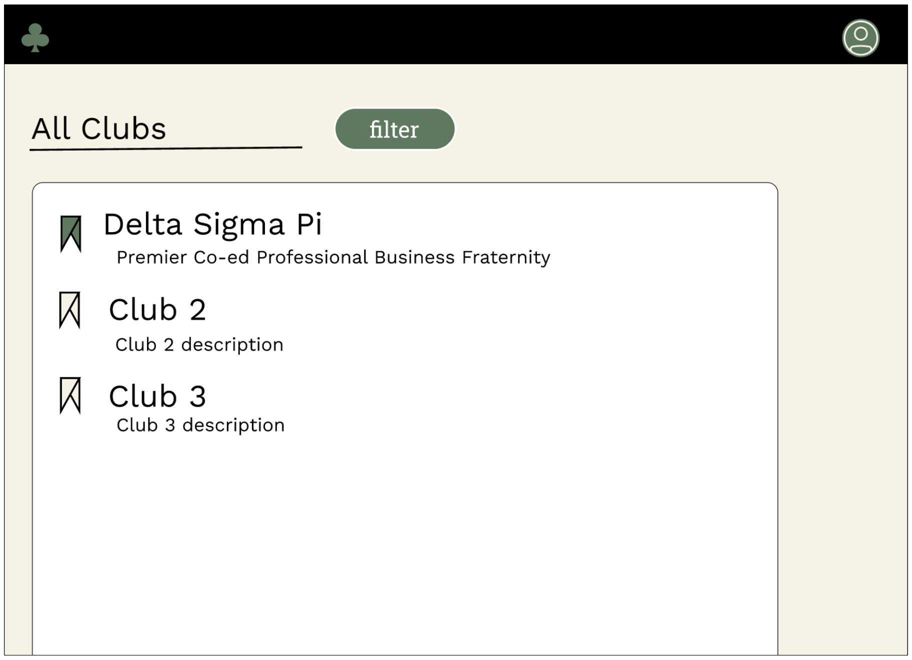
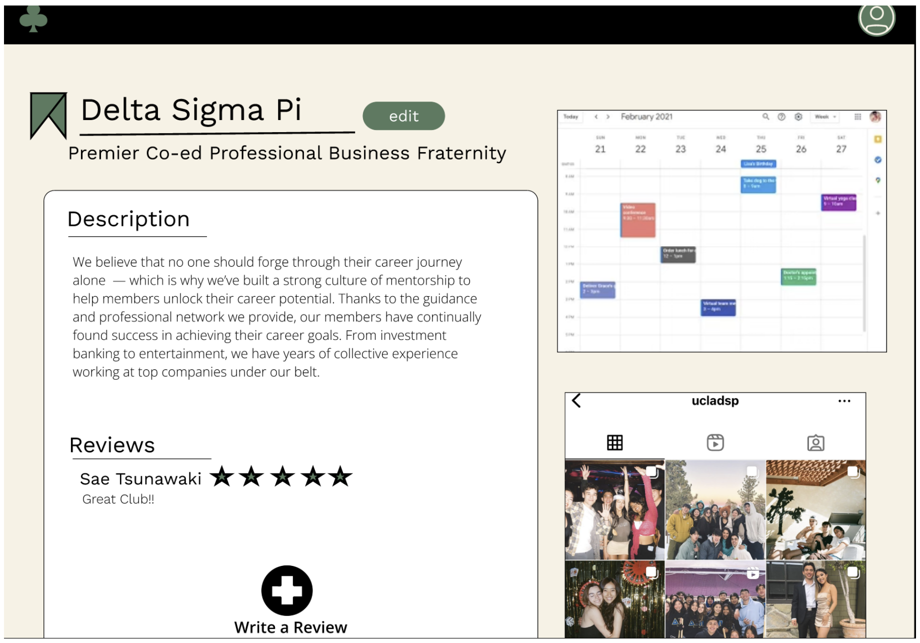


Figure 2 displays the club directory. Users can filter their club search and click on a given club to get to their club page.

Figure 3



We believe that no one should forge through their career journey alone — which is why we've built a strong culture of mentorship to help members unlock their career potential. Thanks to the guidance and professional network we provide, our members have continually found success in achieving their career goals. From investment banking to entertainment, we have years of collective experience working at top companies under our belt.

Figure 3 displays a given club page. Users can get club info and bookmark the club. Note: only clubs will have the edit option.

Figure 4

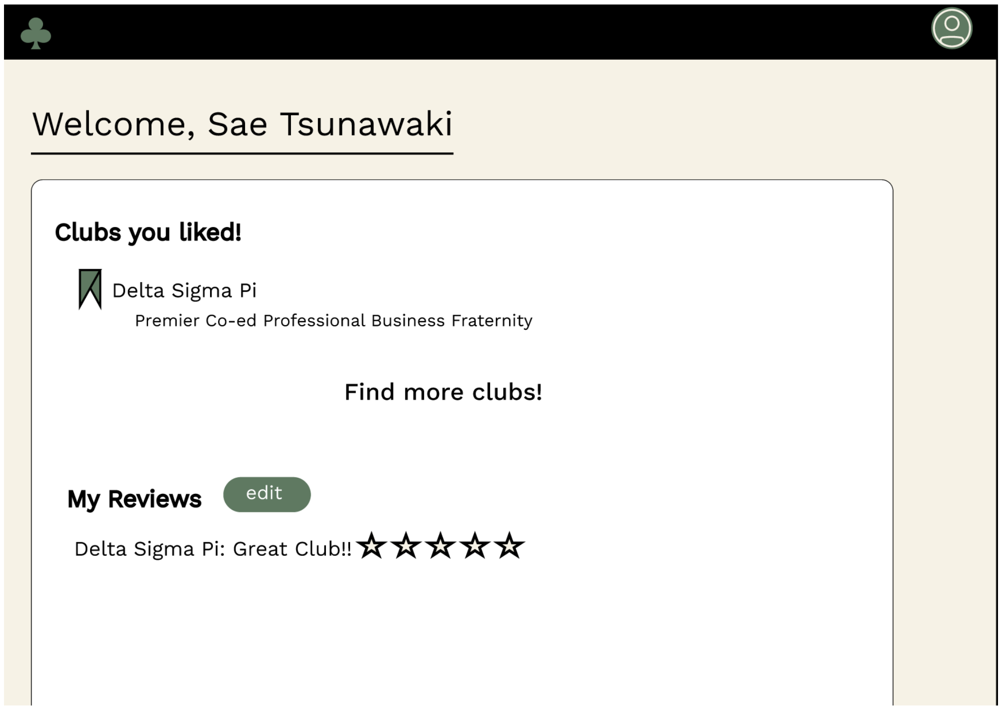


Figure 4 displays a registered user's profile page. Here they can see all their bookmarked clubs and reviews. This page will only exist for registered users.