# ALL CLUBS PAGE

In the All Clubs page, we have a filtering functionality which uses the Effect hook to listen to filters clicked by the user, and handle those changes accordingly. For the filtering UI, we used the built-in <select> component to render a dropdown with various options. For the search functionality, we use regex to match up the user's inputs to the club names, and render clubs with the closest match.

The individual clubs that are rendered as a result of filtering/searching is shown using the <ClubList> component. The <ClubList> component is essentially a collection of <ClubCard> components. Given a filter/search by the user, we query the backend to retrieve all the clubs that are a match, and we append them to an array of clubs. To render these clubs on the screen, we use the .map( ) function to iterate through the array and return a <ClubCard> component.

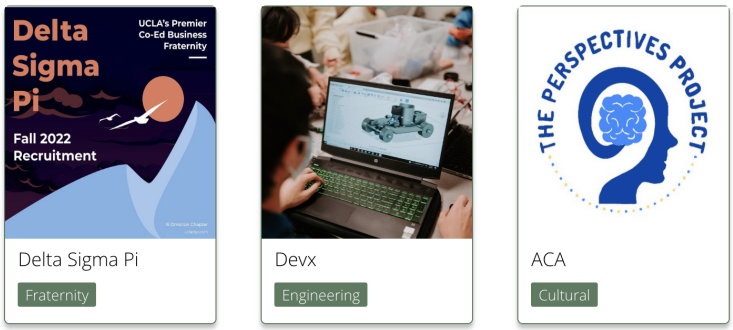


Figure 1: Example of a collection of <ClubCard> components

```
const allClubComponents = AllClubsArray.map((club) => {
    return (
        <ClubCard
            id={club["_id"]}
            profile_picture={club["profile_picture"]}
            name={club["name"]}
            tags={club["tags"]}
        />
    );
});
```

Figure 2: Iterating through the array and returning <ClubCard> components

# CLUB PAGE

When a user clicks on a <ClubCard> component, they are redirected to that specific club's detailed page. On this page, users are able to find a detailed description of the club, as well as some reviews that other users have written about the organization. These reviews are queried to the database and rendered onto the screen. Each review is a <ClubReviews> component, and the overall rating is calculated as the average number of stars given from a user. We used a .reduce( ) function to calculate the average rating, and rendered an appropriate number of filled stars from the average.

```
export const AverageRatingStars = props => {
    var average = 0;
    if(reviewCount != 0) {
        average = props.reviews.reduce((a, b) => a + b.rating, 0)/(props.reviews.length);
        average = Math.trunc(average)
    }

    return (
        <div className='stars'>
            {
                new Array(average).fill(null).map(() => (
                    <BsStarFill size={20} className='star'></BsStarFill>
                ))}
                {new Array(5-average).fill(null).map(() => (
                    <BsStar size={20} className='star'></BsStar>
                ))
            }
        </div>
    );
}
```

Figure 3: Code to calculate Average Rating and render stars

Each <ClubReviews> component includes the review rating, the review author, and the review text. These props are passed into the component, and with the use of an Effect hook, they are rendered onto the screen. When the Write a Review button is clicked, the user is redirected to another page, where they can write a review for that specific club. Page redirection is done through the React Router <Link> element. There is also a <ClubUpdates> component on this page, and it displays recent updates and upcoming events information about the club.

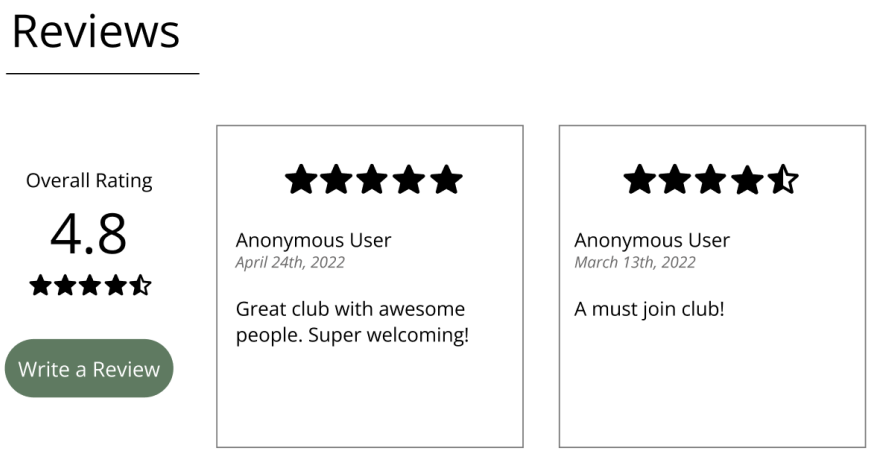


Figure 4: Example of a collection of <ClubReviews> components

# LOGIN

We use google authentication API in order to handle our actual login functionality. To login, the user can click on the login button at the top right of the landing page which will redirect them to our login page where they can click on the google login button to be redirected to logging in through the google API. After logging in through the google login API, you will be redirected to your profile page where you can view your bookmarked clubs and review that you have written, and the top bar will also change to indicate that you are currently logged in.

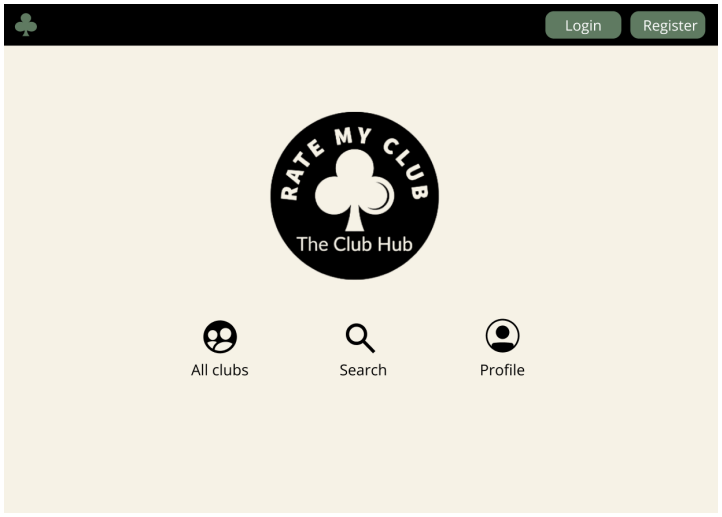


Figure 5: Landing Page with Nav Bar

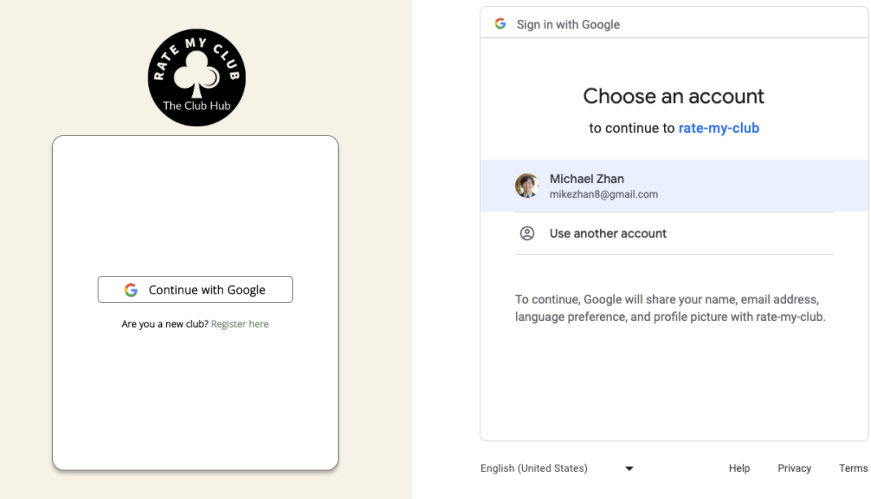


Figure 6 & 7: Logging in through Google Authentication API

The login button showing up on the initial landing page is due to the code below which populates the navigation bar with the login button when the isLoggedIn state is false. It utilizes the State hook. The navigation bar is populated differently when the isLoggedIn state is true as can be seen above. All of this logic is handled inside the <NavBar> component.

```
</Link>
{!isLoggedIn ?
<span className='navBar-buttons'>

  <Link to='/login'>
    <button className='navBar-button'>Login</button>
  </Link>
```

Figure 8: Utilizing State hook for the <NavBar> component

Below we can see the actual implementation of the login page and use of the Google authentication API inside of the LoginPage.js file. We use the react fragment at the button in order to actually populate the page with the site logo and button for logging in through Google as you can see in the button tag. Using the import Google login hooks, we call the login function on click of the login button which sets the user as the response sent back by the google API. Detecting a change in the user state, our top use effect is triggered which uses the axios module in order to make a get request to the Google API using the access token stored in user to set our profile state to an

object with the information of the user including their name, email, profile picture, and some other information. Since a change in the profile state occurred, the second use effect is then triggered where we first try to make a get request to our backend in order to see if the user is an existing user, from which we call pull the username to use to populate the profile page when we redirect to it later. If the get request fails, then we know the user is not registered, thus we use a post request to our database to add the user to our database and then redirect to their profile page.

```
export default function LoginPage() {

    let navigate = useNavigate();

    console.log(isLoggedIn)
    const [ user, setUser ] = useState([]);
    const [ profile, setProfile ] = useState(new Array());
    if (user.length === 0) {
        isLoggedIn = false
    }

    const login = useGoogleLogin({
        onSuccess: (codeResponse) => setUser(codeResponse),
        onError: (error) => console.log('Login Failed:', error)


    });
```

Figure 10: LoginPage() functionality

```
useEffect(
    () => {
        if (user) {
            axios
                .get(`https://www.googleapis.com/oauth2/v1/userinfo?access_token=${user.access_token}`, {
                    headers: {
                        Authorization: `Bearer ${user.access_token}`,
                        Accept: 'application/json'
                    }
                })
                .then((res) => {
                    setProfile(res.data);
                })
                .catch((err) => console.log(err));

            if (user.length != 0){
                isLoggedIn = true;
                console.log(isLoggedIn)
            }

        }

    },
    [ user ]
);
```

Figure 11: Using Effect hooks for Google Auth API

# USER PROFILE PAGE

The user profile page is displayed for every logged in user. It displays the user's bookmarked clubs and the reviews that they have written in the past. There are two main components in this page: <UserBookmark> and <UserReview>. The <UserBookmark> component utilizes the same <ClubCard> component used in the All Clubs page. As for <UserReview>, it is essentially identical to the <ClubReviews> component also used in the detailed club page, but these have an "Edit Review" and "Delete Review" button for the logged in user. The reviews that the user has written and the clubs that they have bookmarked are all stored in the backend, so we query it first to then render it onto the screen through the aforementioned components. Given the list of reviews that the user has written, we used the .map( ) function again to iterate through the list, and create a <ClubReview> component for each review.

```
const userReviews = props.reviews.map((e) => {
  return (
      <ClubReview
          rating={e.rating}
          clubName={e.clubName}
          date={e.date}
          content={e.text}
      />
  );
```

Figure 12: Code used to create <ClubReview> components

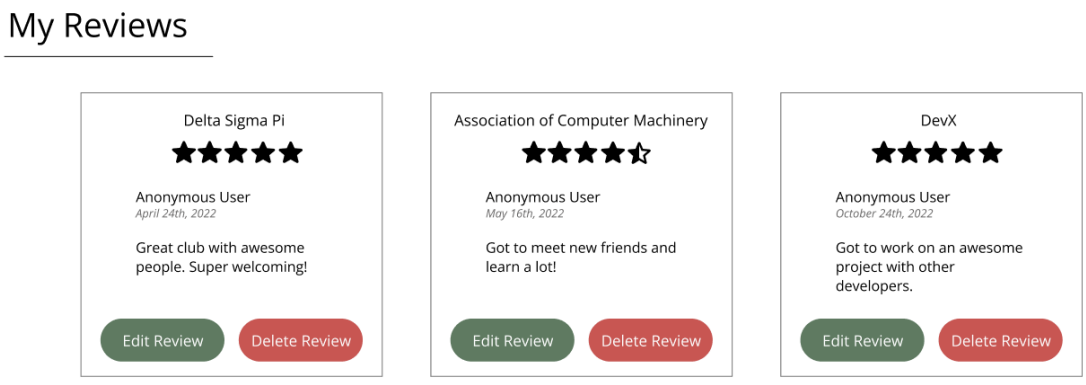


Figure 13: Example of a User profile page populated with Reviews they have written