SW Engineering CSC648/848 FALL 2020

DoReMeet

Team 02

Milestone 4

Nimiksha Mahajan (nmahajan@mail.sfsu.edu) - Team Lead
Meet Patel (mpatel6@mail.sfsu.edu) - Backend & Database Lead
Luke Chang-Knezevich (lchangknezevich@mail.sfsu.edu) - Frontend Lead
Mike Bolanos (mbolanos1@mail.sfsu.edu) - GitHub Master & Backend Engineer
Jungsun Eoh (jeoh@mail.sfsu.edu) - Database & Backend Engineer
Vincent Tse (vtse3@mail.sfsu.edu) - Server Admin & Frontend Engineer

History Table

Version	Date	Comments
M4V1	12/10/2020	Initial Submission
M3V2	12/05/2020	Submission updated with Feedback
M3V1	11/19/2020	Initial Submission
M2V2	12/05/2020	Submission updated with Feedback
M2V1	11/01/2020	Initial Submission
M1V2	10/11/2020	Submission updated with Feedback
M1V1	09/30/2020	Initial Submission

Table Of Contents

1. Product Summary

Product Name: DoReMeet

Our product, DoReMeet, aims to help artists build a community by providing them a platform where they can find collaborators, mentors, and/or friends. Artists, be it singers, dancers, painters, photographers or film makers, can use our product to find people that are close to their geographical location based on their personal preferences. They can choose to connect with fellow artists and collaborate on projects for work, fun or both.

The functions to be delivered in our final product are as follow:

1. Sign Up/Register - Unregistered users shall be able to sign up or register an account on DoReMeet from the landing page. To sign up, users will have to input their email and some other information like their name, date of birth, gender, art category etc.
2. Log In/LogOut - Registered users have the ability to log into their account and log out at any time using the credentials they signed up with.
3. View Potential Matches - Users will be able to see potential matches based on their preferences. They can choose to “Connect” with the user or “Pass” on them.
4. Filter Potential Matches based on Preferences - Users will be able to set their personal preferences for the art category, meeting in person or online, skill level, age and gender. The potential matches they see will be filtered based on these preferences, thus giving them more targeted search results.
5. Expand and Reduce Search Radius - If the user chooses to meet in person, they have the ability to set a radius limit for their potential matches. They can choose to expand or reduce their search radius, and will be shown potential matches only within the specified distance.
6. Matches - Users will be matched with people who they choose to connect with, if the other person also wants to connect with them. Once connected, the user has the ability to chat with the match and collaborate as they deem fit.
7. Chat - Users will be able to send and receive messages to/from the other users they’ve matched with.
8. Update Preferences - The user has the ability to update their preferences for art category, meeting (in person or online), minimum and maximum age, gender and

skill level. Updating their preferences will also update their potential match results to meet their new search criteria.

9. Search for Posts - Users also have the ability to search for posts on the Community page based on the Post Title and the Art Category

Unique Features:

1. Simple Connect / Pass Match Page - A simplified, not overwhelming space for users to pick and choose from artists nearby and around the globe. Unlike our competitors, we don't show users a bulk of fellow artists to pick from; instead we show that one person at a time and users get to directly see that individuals work. They can view the profile, and if interested choose to "Connect" with the user or they can simply "Pass" and will not be matched with the user.
2. Community Page - People will be able to see the Community page, with the recent posts made by the platform users and the Highlights of the month. Only users who have a registered account and are logged in will be able to post to the Community page to share their collaborations.
3. Community highlights - The community page will have a highlights section to show most liked posts for the month, based on user votes. Only registered users will be able to vote on the posts.

Product URL: http://ec2-13-52-247-220.us-west-1.compute.amazonaws.com/

2. Usability Test Plan

Test Objective

DoReMeet aims to provide users a platform to find and connect with local (or global) artists to collaborate, by offering them a list of potential matches based on their personal preferences(based on art category, distance, skill level, age, gender). With our product we are extending the practice of instinctual decision-making via a quick, simple yes or no to the art community, and help them find other artists near them. Given this mission, usability testing is crucial for us, because we want to ensure that we are in fact giving the users a simple, not overwhelming, easy to navigate platform. We need some core functionalities implemented in our product, and need to ensure that they serve our product's purpose; the main feature we need usability testing for are as follows:

1. ***Match page*** *shows potential matches to connect or pass on.*
   The very idea of our product is to offer a list of artists as potential matches to the user, based on their preferences. Therefore it is crucial to test the Match page to see that the users understand the idea of the page, and can instinctively navigate through it. We want to test that the users can view the potential matches' profile, see their work and then choose to "Connect" or "Pass" based on their interest.

2. *The* ***Community page*** *shows Recent Posts made by DoReMeet users, and one can search for posts based on the Post Title and Art Category*
   Our product aims to provide a space for artists to build a collaborative community and the Community Page promotes that idea. We want to see that the users (both registered and unregistered) are able to see the Recent Posts, and see the different collaborations made by DoReMeet users, thus encouraging them to also use the platform and submit their own collaborations. Also, we are offering a search functionality on the Community page so that it is easy to sift through all the posts made. Via testing, we want to make sure users understand that they can view posts of a certain art category or look for a particular post by its title.

3. *Users can send and receive messages to/from their matches on the* ***Chat page***.
   The only way for people to actually connect with other artists and start collaborating is by interacting with them, and the Chat page provides that functionality. We want to check that the users understand they can only chat with people they match with. Also, the chat page needs to be tested to ensure that users are able to both send and receive messages to and from their matches.

4. *Users can update their **Profile** (bio, media, tags, preferences)*
   DoReMeet's mission is to provide a customized experience for all its users, so we want the users to be able to update their own profile, whenever they want. Testing this page is important because users need to see that they can easily add/edit their bio, add/remove media files and the tags they choose and their preferences for the potential matches. Testing this feature is important because users should be able to easily navigate updating their profile/preferences, and should be able to do so whenever they want.

5. *Users will be able to get potential matches based on their **location preference** (expand and reduce the search radius)*
   One of the key features of DoReMeet is that users can find people based on their location preferences. Users are able to adjust the distance for their potential matches. With testing, we want to make sure that the users know that they have the option to share their location with our product and set a minimum or maximum distance for their potential matches' preference. Also we need to make sure that the user knows they can adjust the radius anytime, or just stop sharing their location altogether if they like.

Test Description

Usability Metric - Satisfaction

System Setup: DoReMeet is compatible with Safari, Chrome, Mozilla Firefox and Edge. For usability testing purposes, we are getting the product tested by one of our teammates and by another person who was not involved in the development of the product, to ensure that our product in fact meets the satisfaction requirements.
The tester will be given the product URL, and we will let them create an account, so that they can access all the features of our product. We'll see if they are able to update their profile, and set their preferences and location settings to get potential matches based on their interest. We'll observe how they use the Match page and if they are able to go through a list of potential matches by choosing to "connect" with or "pass" on them. We'll also check if they can navigate through the Community page, search for posts, and use the Chat page (send messages to anyone they match with).

Starting Point: The starting would be the landing page for DoReMeet (http://ec2-13-52-247-220.us-west-1.compute.amazonaws.com/) where the tester can start by creating an account, or they can go to the Community page to look at posts made by the users.

Intended Users: The intended users for this product are artists who are looking for fellow artists, in the Art, Music, Dance, Film or Photography category, to collaborate with. The users can be of any age over 18 and be of part of any gender, caste, race demographic. Also, the user can be a novice, amateur, expert or professional artist in their category, and can be looking for collaborators for work or for hobby/passion projects.

URL: http://ec2-13-52-247-220.us-west-1.compute.amazonaws.com/

What is to be measured: The test is supposed to measure how a user feels about our website, how they like the look of it and how easily they can navigate through the different pages. We want to measure on Likert scale how intuitive they find the Match page, navigating through potential matches, the Community pages along with all the other pages. We'll see if a user can recognize all the given functionalities and find out if they are satisfied with their experience, or if anything overwhelmed or confused them. Overall website friendliness and navigation will also be measured.

Usability Task Description

Open the website http://ec2-13-52-247-220.us-west-1.compute.amazonaws.com/ in any preferred browser that is supported by the product (Chrome, Safari, Mozilla Firefox or Edge) and create an account by signing up. Once you're logged in, test the following:

Task	Description
Matches	Navigate the match page, view the potential match's profile and if you like it, connect with them but if you don't pass on them
Community	Go to the Community Page and check out the Recent posts. Also, search for posts based on any art category. Post to the Community page under any of your preferred art categories.
Chat	Send a message to anyone you've matched with
Profile/Preferences	Update the profile bio, add a media file and update the art category preference. Also, update the skill level specified for the account.
Location (expand/reduce search radius)	Allow the product to use your location, and set the search radius for potential matches to be 10 miles. Check if the results displayed on the Match page are in fact within 10 miles

Usability Test Table

Test / Use Case	% Completed	Errors	Comments
Matches	20%	Cannot see potential matches or connect/pass them	No matches based on the set preferences visible
Community	95%	Can't see all images	Only certain images are visible; not all uploaded images can be seen
Chat	50%	Can't receive messages from matches	If there are no matches, there is no message prompt as to why we can't see messages. Don't see any messages or directions
Profile/ Preferences	80%	Media files and users' Community posts not linked	Can't see the posts that the user made on the community page.
Location (expand/reduce search radius)	60%	Cannot expand or reduce search radius.	Can't adjust the desired match radius and there is no way to see how far a potential match is

Questionnaire

Statements	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Match page shows potential match results based on your set preferences.	1	2	3	4	5
It is easy to either connect with or pass on a potential match.	1	2	3	4	5
It is easy to navigate through the community page and look for posts.	1	2	3	4	5

In this questionnaire, there will also be an optional *Comments section*, where the testers will have the ability to share any comments/suggestions they have.

We will use the results from usability testing to gauge the overall satisfaction with our product and get information about what we can potentially do or not do to enhance the user experience. This will help us not only make our product better, but also help with future marketing by giving us better insights on what the user needs.

3. QA Test Plan

Test objectives:

For our QA testing we are testing some of the core functionalities of DoReMeet that are required for a smooth user experience. With this testing, we'll ensure that the user is able to access DoReMeet and navigate through at least the basic features that the website provides. We will be testing the functionality and effectiveness of some key non-functional requirements of the website, and make sure that these features work properly, as required for all users on all devices and browsers.

The first thing we are testing is browser compatibility, to ensure that the user is able to use DoReMeet on all major web browsers, and their most commonly used versions.

The next feature we are testing is that only eligible users are able to create an account on DoReMeet, in order to be in compliance with our legal terms and also to ensure that we are not exposing minors to any content that is not approved by their parents/guardians.

The third feature we are testing is our login functionality to make sure that registered users are able to access their account and enjoy the features offered by DoReMeet.

Also to ensure that we are upkeep transparency and giving users a chance to make informed decisions, we'll be testing to make sure every time a user leaves the site by clicking on a third-party link, we give them an alert and let them know that they are leaving the DoReMeet space. Our website is not responsible for any cookies/cache collected by the third party and we want our users to be aware of that.

Lastly for the QA testing we'll also check to make sure that the website asks the users every time if they want to share their location, and check that no user information is being collected with the users explicit approval.

HW and SW setup:

Our product is accessible on any machine (MacOS, Windows or Linux) that is connected to the internet. It is also available on iPhones and Android devices.
The website is compatible with

- Chrome, version 80.0 - 85.0
- Safari, version 10.0 - 14.0
- Mozilla Firefox, version 78 - 81
- Edge, version 85

No additional software/libraries need to be running on the client side.

Feature to be tested:

To ensure that we are providing a service that is accessible, secure and functional, we are choosing to test the following critical functionalities.

1. The site shall be compatible with Safari (version 10.0 - 14.0), Chrome (version 80 - 85.0) and Mozilla Firefox (version 78 - 81) and Edge (version 85).
2. Users should be minimum 18 years of age to create an account.
3. The website shall warn users when clicking on links that take them to another website.
4. The user shall be required to enter both password and username to log in.
5. Users' geolocation shall only be known by the website if they allow it.

QA Test Plan:

Test #	Test Title	Test Description	Test Input	Expected Output	Test Results
1.	Sign up - Must be at least 18 years of age	Any user trying to create an account must be at least 18 years old. If they are under 18, they should get a warning and not be able to create an account	For Date of Birth. enter a date in 2003 or after	Error message letting the user they do not meet DoReMeet's age requirement	PASS
2.	Leaving website message	The user should get a warning message if they click on a link that takes them to a third party website	Click on the Spotify link in a potential matches' profile	An alert message saying that the user is leaving DoReMeet and visiting a third party site	PASS
3.	Allow location use	The website shall explicitly ask user if they "Allow" the site to user their location, and do so only id the user agrees	Log into your DoReMeet account	Get an alert message asking if the user is Allow access to their location to get potential matches. Users can choose yes or no.	FAIL *not yet implemented on the actual product website

QA Test Browser Compatibility:

Test #	Browser	Compatibility (Yes/No)
1.	Chrome, version 80.0 - 85.0	Yes
2.	Safari, version 10.0 - 14.0	Partly. Some components do not display correctly
3	Mozilla Firefox, version 78 - 81	Yes
4.	Edge, version 85	Yes

4. Code Review

a. Coding Style

The coding style we've used is:

- Careful indentation and use of tabs instead of spaces
- Camel case for naming
- Descriptive names for variables and functions
- Code organized in folders categorized by their functionality (Auth, NavigationBar, Footer, card component etc.)
- /* Comments */ as header for every file
- / inline comments within the code
- Modular functions

b. Code

For the Code Review we chose to get our Community Page code reviewed since it is a core functionality of our product. The Community page is also a unique feature that we provide so that artists can share their collaborations and build a community with other DoReMeet users.

Code Review request email:

CSC648 Team 02 Code Review Inbox ×

Luke Chang-Knezevich
to Nimiksha

Hi Nimiksha,

Here's the code from our CommunityPage.js for the code review.

```
/*
**CSC 648 Team 02 DoReMeet
**File: CommunityPage.js
**Desc: Contains all the code needed for the community page. Calls on the backend through axios and
recieves json responses that it then displays accordingly on the page.
*/

import React, {useEffect, useState} from 'react';
import '../App.css';
import './CommunityPage.css';
import axios from 'axios';
import Navbar from '../components/Navbar/Navbar';
import Highlights from '../components/Highlights/Highlights';
import Footer from '../components/Footer/Footer';
import { BrowserRouter as Router } from 'react-router-dom';


const CommunityPage = (stateObj) => {
```

Code to be reviewed:

```
/*
**CSC 648 Team 02 DoReMeet
**File: CommunityPage.js
**Desc: Contains all the code needed for the community page. Calls on the
backend through axios and recieves json responses that it then displays
accordingly on the page.
*/

import React, {useEffect, useState} from 'react';
import '../App.css';
import './CommunityPage.css';
import axios from 'axios';
import Navbar from '../components/Navbar/Navbar';
import Highlights from '../components/Highlights/Highlights';
import Footer from '../components/Footer/Footer';
import { BrowserRouter as Router } from 'react-router-dom';

const CommunityPage = (stateObj) => {

  const onChange = e => {
    e.preventDefault();
    stateObj.setFile(e.target.files[0]);
    stateObj.setFileName(e.target.files[0].name);
  };

  //Handles the posting of projects to the community page
  const postHandler = async e => {
    e.preventDefault();
    const formData = new FormData();
    stateObj.setPostFile(stateObj.file);
    formData.append('file', stateObj.file);
    formData.append('post_title', stateObj.postName);
    formData.append('post_category', stateObj.postCategory);
    closePost();
    await axios.post('/makePost', formData, { headers: { 'Content-Type':
'multipart/form-data' } });
  };
```

```
  //Sends the search terms to the backend and gets the response
  const submitHandler = (event) => {
    event.preventDefault();
    alert("You are searching for " + stateObj.searchTitle + " " +
stateObj.searchCategory);

    axios.get('/searchPost', { params: { post_title: stateObj.searchTitle,
post_category: stateObj.searchCategory } }).then(response => {
      if(response.data.length > 0){
        console.log(response.data.length);
        let _html = "";
        _html += `<h1>Search Results</h1>`
        _html += `<div class="RecentPostsFormat">
                  <img class="PostImage"
src="assets/postImages/${response.data[0].post_file}" alt="Post Image"/>
                  <h2 class="PostTitle">${response.data[0].post_title}</h2>
                  <h3
class="PostCategory">${response.data[0].post_category}</h3>
                  <h4 class="PostVotes">${response.data[0].post_votes}</h4>
                  <p class="PostDescription">Post Description</p>
                  <p hidden class="PostID">${response.data[0].comm_pg_id}</p>
                  <button id="PlusButton" type="button">+</button>
                  <button id="MinusButton" type="button">-</button>
                  </div>`;
        document.getElementById("search-post").innerHTML = _html;
        document.getElementById("recent-posts").innerHTML = '';

        document.getElementById("PlusButton").addEventListener("click",
voteplus);
        document.getElementById("MinusButton").addEventListener("click",
voteminus);
      }
else{
        let _html = "";
        _html += `<h1>Search Results</h1>`
        _html += `<div class="NoResult">
                  <p>Sorry, we couldn't find anything</p>
                  </div>`;
        document.getElementById("search-post").innerHTML = _html;
        document.getElementById("recent-posts").innerHTML = '';
```

```
      }

    }).catch(function (error) {
      stateObj.setResultTitle("Not Found");
      stateObj.setResultCategory("Not Found");
      stateObj.setResultFile("");
    });
  };

  const voteplus = () => {
    axios.post('/voteplus');
    console.log("vote test");
  };
  const voteminus = () => {
    axios.post('/voteminus');
    console.log("vote test");
  };

  //Simply opens up the post box
  const openPost = () => {
    document.getElementById("postform").style.display = "block";
  }
  //Closes the post box
  const closePost = () => {
    document.getElementById("postform").style.display = "none";
  }
    //Upon entering the page the most recent 5 posts are displayed
     useEffect(() => {
       axios.get('/recent5').then(response => {
         console.log(response.data[0]);
         let _html = "";
         _html += `<h1>Recent Posts</h1>`;
         response.data.forEach(post => {_html += `<div
class="RecentPostsFormat">
                  <img class="PostImage"
src="assets/postImages/${post.post_file}" alt="Post Image">
                  <h2 class="PostTitle">${post.post_title}</h2>
                  <h3 class="PostCategory">${post.post_category}</h3>
```

```
                    <p class="PostDescription">Post DescriptionPost
DescriptionPost DescriptionPost DescriptionPost DescriptionPost DescriptionPost
DescriptionPost
                    DescriptionPost Description</p>
                    <button id="PlusButton" type="button">+</button>
                    <button id="MinusButton" type="button">-</button>
                    </div>`;})
          document.getElementById("recent-posts").innerHTML = _html;
          document.getElementById("search-post").innerHTML = '';
          document.getElementById("PlusButton").addEventListener("click",
voteplus);
          document.getElementById("MinusButton").addEventListener("click",
voteminus);
        }).catch(function (error) {
          console.log('fail')
        });
      });

  return (
    <>
      <Router>
        <div className="App">
          <Navbar />
          <header className="App-header">
            <div class="description">
              <h2 align='center' top='30%'> Community Page </h2>
              <p style={{fontSize: 22, marginLeft: 40, marginRight: 40 }}
align='center'>See how other people are using DoReMeet to bring their artistic
dreams to life, or post some of your own amazing collaborations with fellow
DoReMeet users
              <br />
              <br />
              Check out our <a href="#highlight-section"> Highlights section
</a> to see the most liked posts of the month.</p>
            </div>
            <div class="PageContainer">
              <div class="SearchContainer">
                <form class="search" onSubmit={submitHandler}>
                  <input class="searchBar" onChange={e =>
stateObj.setSearchTitle(e.target.value)} type="text" placeholder="Search" />
```

```
                  <select class="searchButtons" onChange={e => {
stateObj.setSearchCategory(e.target.value); }}>
                    <option value={"Music"}>Music</option>
                    <option value={"Dance"}>Dance</option>
                    <option value={"Art"}>Art</option>
                    <option value={"Cinema"}>Cinema</option>
                    <option value={"Photography "}>Photography</option>
                    {/* <option value={"M"}>Music</option>
                    <option value={"D"}>Dance</option>
                    <option value={"A"}>Art</option>
                    <option value={"C"}>Cinema</option>
                    <option value={"P"}>Photography</option> */}
                  </select>
                   <input class="searchButtons" type='submit' />
                 </form>
               </div>
               <div className="PostFormFormat">
                 <label htmlFor="postbutton">Share your work with us
here!</label>
                 <button className="post" id="postbutton"
onClick={openPost}>Post</button>
               </div>
               <div class="post-popup" id="postform">
                 <form class="post-container" onSubmit={postHandler}>
                   <h2> Post your creative collaborations! </h2>
                   <input type="text" onChange={e =>
stateObj.setPostName(e.target.value)} placeholder="Name" required />
                   <div>
                     <select onChange={e => {
stateObj.setPostCategory(e.target.value); }}>
                      <option >Music</option>
                      <option >Dance</option>
                      <option >Art</option>
                      <option >Cinema</option>
                     <option >Photography</option>
                     </select>
                  </div>
                   <label htmlFor="post-file"> Select file: </label>
       
```

```
                  <input type='file' className='custom-file-input'
id='customFile' onChange={onChange} />
                  <div>
                   <input type='submit' />
                  </div>
                   <div>
                    <button type= "button" onClick={closePost}>Close</button>
                   </div>
                 </form>
              </div>
               <div class="RecentPosts" id="recent-posts">
                 </div>
                 <div class="SearchPost" id="search-post">
               </div>
               <p id="vote"></p>
               <div id='highlight-section' >
                 <Highlights />
               </div>
             </div>
           </header>
         </div>
         <Footer />
       </Router>
     </>
  );
}

export default CommunityPage;
```

Code Review:

Gmail Nimiksha Mahajan <nimiksha98@gmail.com>

CSC648 Team 02 Code Review

Nimiksha Mahajan <nimiksha98@gmail.com> Mon, Dec 7, 2020 at 11:53 AM
To: Luke Chang-Knezevich <lchangknezevich@mail.sfsu.edu>

Hi Luke,

Thank you for reaching out. Please find my review for the Community Page code.
If you have any questions, please let me know.

```
<--- CODE REVIEW
This is good. It is helpful to have a header here
--->
/*
**CSC 648 Team 02 DoReMeet
**File: CommunityPage.js
**Desc: Contains all the code needed for the community page. Calls on the backend
through axios and recieves json responses that it then displays accordingly on the
page.
*/

import React, {useEffect, useState} from 'react';
import '../App.css';
import './CommunityPage.css';
import axios from 'axios';
import Navbar from '../components/Navbar/Navbar';
import Highlights from '../components/Highlights/Highlights';
import Footer from '../components/Footer/Footer';
import { BrowserRouter as Router } from 'react-router-dom';

const CommunityPage = (stateObj) => {

<--- CODE REVIEW
Should add comments. Just onChange doesn't really give a good description of what
this is doing. Or consider more a descriptive name
--->
  const onChange = e => {
    e.preventDefault();
    stateObj.setFile(e.target.files[0]);
    stateObj.setFileName(e.target.files[0].name);
  };
```

```
<--- CODE REVIEW
Good comment and descriptive name
--->
  //Handles the posting of projects to the community page
  const postHandler = async e => {
    e.preventDefault();
    const formData = new FormData();
    stateObj.setPostFile(stateObj.file);
    formData.append('file', stateObj.file);
    formData.append('post_title', stateObj.postName);
    formData.append('post_category', stateObj.postCategory);
    closePost();
    await axios.post('/makePost', formData, { headers: { 'Content-Type':
'multipart/form-data' } });
  };

<--- CODE REVIEW
It's helpful to have the alert for what the user is searching for, but considering
from a UI/UX standpoint it might not be the most efficient thing to do. Alerts for
every single search can be tedious/unnecessary.
But good work with comments and naming.
--->
  //Sends the search terms to the backend and gets the response
  const submitHandler = (event) => {
    event.preventDefault();
    alert("You are searching for " + stateObj.searchTitle + " " +
stateObj.searchCategory);


<--- CODE REVIEW
I think you've done a good job with naming everything, but I wonder if the code is
the most efficient. Is using html components the most effective in terms of both
frontend and backend? I feel you can use more modular code here, that will also
improve the time to display search results
 --->
    axios.get('/searchPost', { params: { post_title: stateObj.searchTitle,
post_category: stateObj.searchCategory } }).then(response => {
      if(response.data.length > 0){
        console.log(response.data.length);
        let _html = "";
        _html += `<h1>Search Results</h1>`
        _html += `<div class="RecentPostsFormat">
                  <img class="PostImage" src="assets/postImages/${respo
nse.data[0].post_file}" alt="Post Image"/>
```

```
                    <h2 class="PostTitle">${response.data[0].post_title}</h2>
                    <h3 class="PostCategory">${response.data[0].post_category}</h3>
                    <h4 class="PostVotes">${response.data[0].post_votes}</h4>
                    <p class="PostDescription">Post Description</p>
                    <p hidden class="PostID">${response.data[0].comm_pg_id}</p>
                    <button id="PlusButton" type="button">+</button>
                    <button id="MinusButton" type="button">-</button>
                    </div>`;
        document.getElementById("search-post").innerHTML = _html;
        document.getElementById("recent-posts").innerHTML = '';

<--- CODE REVIEW
I'd suggest renaming the buttons, to be more indicative of their purpose - to Vote
up or Vote down
--->
        document.getElementById("PlusButton").addEventListener("click", voteplus);
        document.getElementById("MinusButton").addEventListener("click",
voteminus);
      }

<--- CODE REVIEW
I think you should make the error message more descriptive, and display why the
search failed
--->
      else{
        let _html = "";
        _html += `<h1>Search Results</h1>`
        _html += `<div class="NoResult">
                    <p>Sorry, we couldn't find anything</p>
                    </div>`;
        document.getElementById("search-post").innerHTML = _html;
        document.getElementById("recent-posts").innerHTML = '';
      }

    }).catch(function (error) {
      stateObj.setResultTitle("Not Found");
      stateObj.setResultCategory("Not Found");
      stateObj.setResultFile("");
    });
  };
<--- CODE REVIEW
Naming - use camel case for better readability and consistency
--->
  const voteplus = () => {
```

```
    axios.post('/voteplus');
    console.log("vote test");
  };
  const voteminus = () => {
    axios.post('/voteminus');
    console.log("vote test");
  };

<--- CODE REVIEW
The comments are good. Really helps the reader see what function is doing what.
Also helps with debugging and testing
--->
  //Simply opens up the post box
  const openPost = () => {
    document.getElementById("postform").style.display = "block";
  }
  //Closes the post box
  const closePost = () => {
    document.getElementById("postform").style.display = "none";
  }
    //Upon entering the page the most recent 5 posts are displayed
     useEffect(() => {
       axios.get('/recent5').then(response => {
         console.log(response.data[0]);
         let _html = "";
         _html += `<h1>Recent Posts</h1>`;
         response.data.forEach(post => {_html += `<div class="RecentPostsFormat">
                  <img class="PostImage" src="assets/postImages/${post.post_file}"
alt="Post Image">
                  <h2 class="PostTitle">${post.post_title}</h2>
                  <h3 class="PostCategory">${post.post_category}</h3>
<--- CODE REVIEW
What is the purpose of all the Post Descriptions? I suggest adding comments and
making the code more modular instead of arbitrarily using 'Post Description'
--->
                  <p class="PostDescription">Post DescriptionPost DescriptionPost
DescriptionPost DescriptionPost DescriptionPost DescriptionPost DescriptionPost
                  DescriptionPost Description</p>
                  <button id="PlusButton" type="button">+</button>
                  <button id="MinusButton" type="button">-</button>
                  </div>`;})
         document.getElementById("recent-posts").innerHTML = _html;
         document.getElementById("search-post").innerHTML = '';
         document.getElementById("PlusButton").addEventListener("click", voteplus);
```

```
        document.getElementById("MinusButton").addEventListener("click",
voteminus);
      }).catch(function (error) {
        console.log('fail')
      });
    });

<--- CODE REVIEW
The code is good, but I think it should be reformatted to be more readable.
--->
  return (
    <>
      <Router>
        <div className="App">
          <Navbar />
          <header className="App-header">
            <div class="description">
              <h2 align='center' top='30%'> Community Page </h2>
              <p style={{fontSize: 22, marginLeft: 40, marginRight: 40 }}
align='center'>See how other people are using DoReMeet to bring their artistic
dreams to life, or post some of your own amazing collaborations with fellow
DoReMeet users

<--- CODE REVIEW
Should avoid using <br>. Instead set padding or margins.
--->
              <br />
              <br />
<--- CODE REVIEW
You should use a consistent naming format,and not use Camel case and '-'
alternatively. Since camel case is more prominent in the code, I suggest ensuring
that all variables/ids/functions are named using Camel case style.
--->
              Check out our <a href="#highlight-section"> Highlights section </a>
to see the most liked posts of the month.</p>
            </div>
            <div class="PageContainer">
              <div class="SearchContainer">
                <form class="search" onSubmit={submitHandler}>
                  <input class="searchBar" onChange={e =>
stateObj.setSearchTitle(e.target.value)} type="text" placeholder="Search" />
                  <select class="searchButtons" onChange={e => {
stateObj.setSearchCategory(e.target.value); }}>
                    <option value={"Music"}>Music</option>
```

```
                    <option value={"Dance"}>Dance</option>
                    <option value={"Art"}>Art</option>
                    <option value={"Cinema"}>Cinema</option>
                    <option value={"Photography "}>Photography</option>
                    {/* <option value={"M"}>Music</option>
                    <option value={"D"}>Dance</option>
                    <option value={"A"}>Art</option>
                    <option value={"C"}>Cinema</option>
                    <option value={"P"}>Photography</option> */}
                  </select>
                   <input class="searchButtons" type='submit' />
                 </form>
               </div>
               <div className="PostFormFormat">
<--- CODE REVIEW
Naming - use camel case for better readability and consistency
--->
                 <label htmlFor="postbutton">Share your work with us here!</label>
                 <button className="post" id="postbutton" onClick=
{openPost}>Post</button>
               </div>
               <div class="post-popup" id="postform">
                 <form class="post-container" onSubmit={postHandler}>
                   <h2> Post your creative collaborations! </h2>
                   <input type="text" onChange={e => stateObj.setPostName(e.target.
value)} placeholder="Name" required />
                   <div>
                     <select onChange={e => { stateObj.setPostCategory(e.tar
get.value); }}>
                      <option >Music</option>
                      <option >Dance</option>
                      <option >Art</option>
                      <option >Cinema</option>
                     <option >Photography</option>
                     </select>
                  </div>
                   <label htmlFor="post-file"> Select file: </label>
<--- CODE REVIEW
This is not ideal. You should use styling to add the space instead of  arbitrarily
using spaces.
Also code should be formatted in a hierarchy structure to be more readable.
--->
       
```

```
                <input type='file' className='custom-file-input' id='customFile'
onChange={onChange} />
                <div>
                 <input type='submit' />
                </div>
                 <div>
                  <button type= "button" onClick={closePost}>Close</button>
                 </div>
               </form>
             </div>
<--- CODE REVIEW
Should clean up this code, and condense it to avoid multiple divs. can us <div ....
/> instead.
--->
              <div class="RecentPosts" id="recent-posts">
                </div>
                <div class="SearchPost" id="search-post">
              </div>
              <p id="vote"></p>
              <div id='highlight-section' >
                <Highlights />
              </div>
            </div>
          </header>
        </div>
        <Footer />
      </Router>
    </>
  );
}

export default CommunityPage;
```

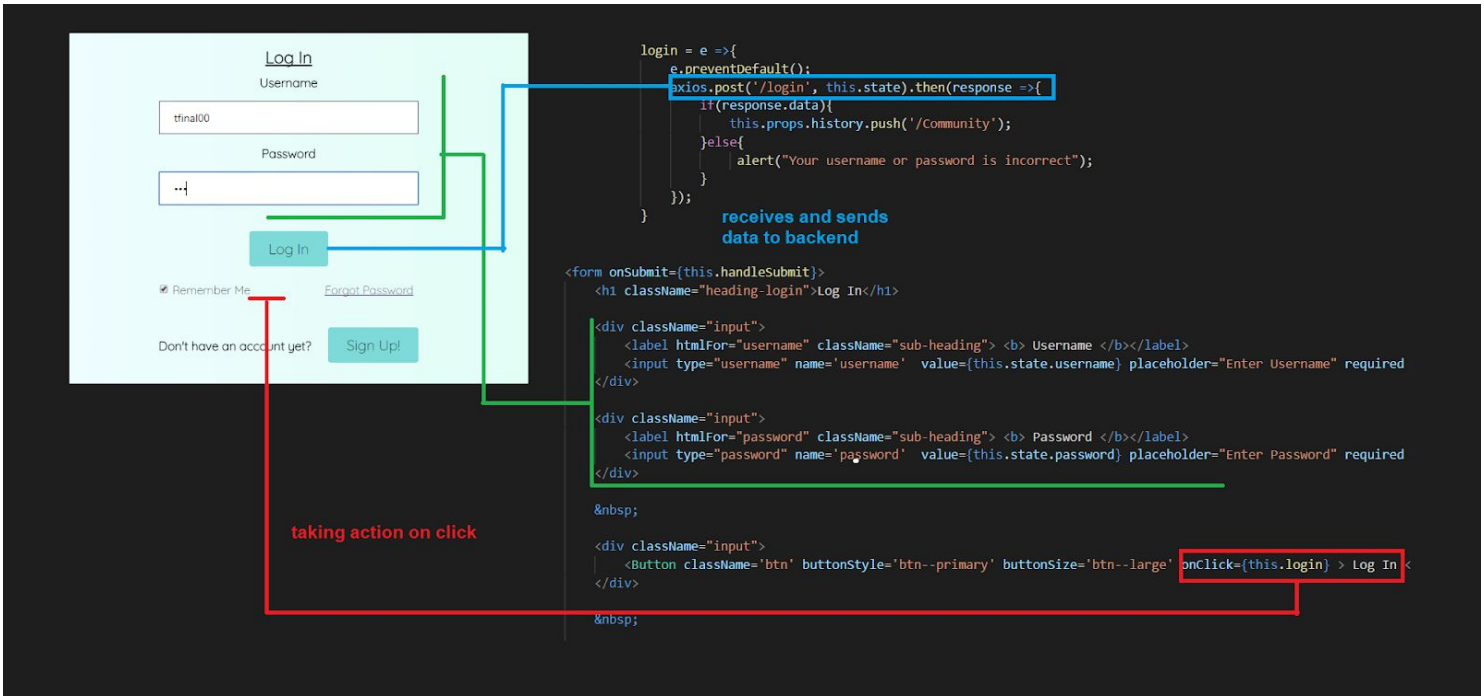
5. Self-Check on best Security Practice

1. List major assets you are protecting

- Password
- Users' personal information (email, phone #, DOB)
- Media files (images) on server
- Users Location
- Database

2. Password encryption in database

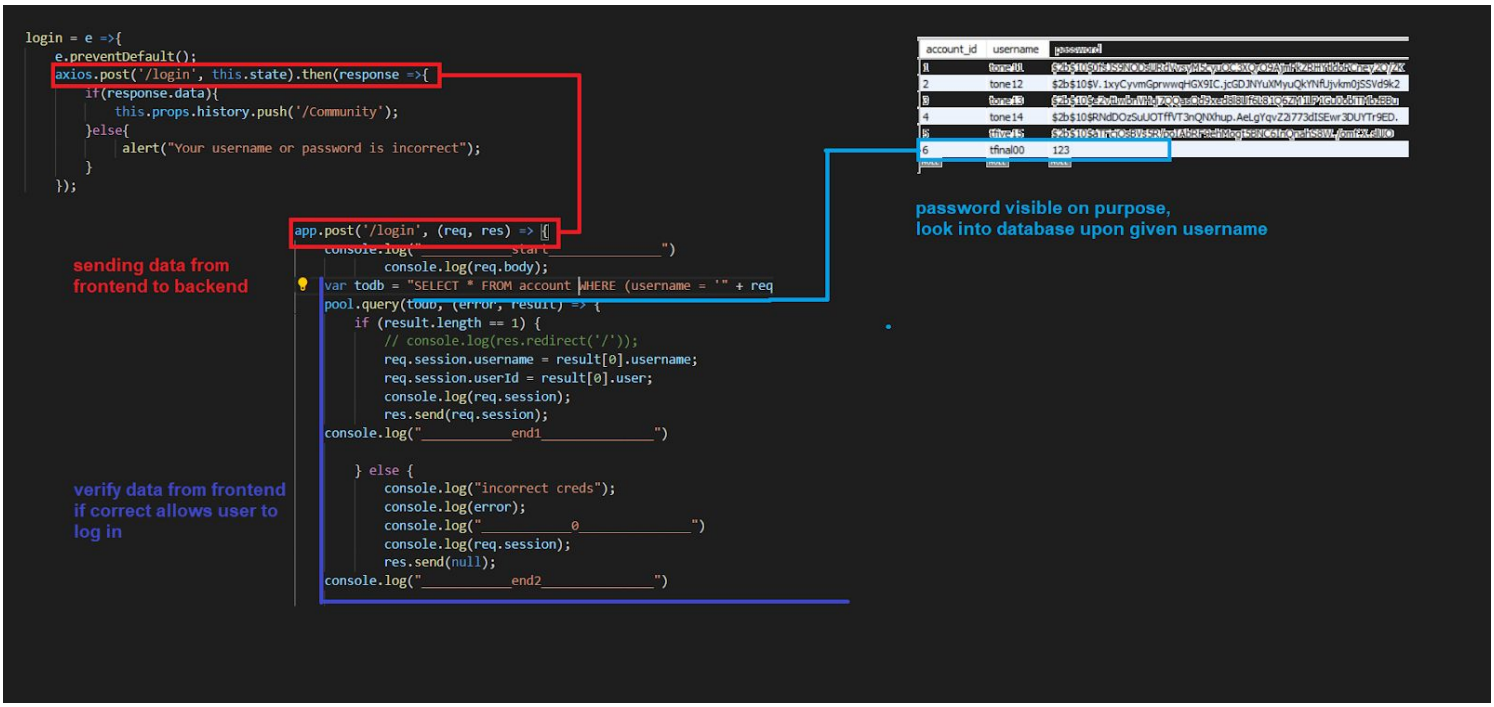
Login


Log In
Username
tfinal00
Password
Log In
Remember Me
Forgot Password
Don't have an account yet?
Sign Up!
taking action on click
receives and sends
data to backend
login = e =>{
e.preventDefault();
axios.post('/login', this.state).then(response =>{
if(response.data){
this.props.history.push('/Community');
}else{
alert("Your username or password is incorrect");
}
});
}
<form onSubmit={this.handleSubmit}>
<h1 className="heading-login">Log In</h1>

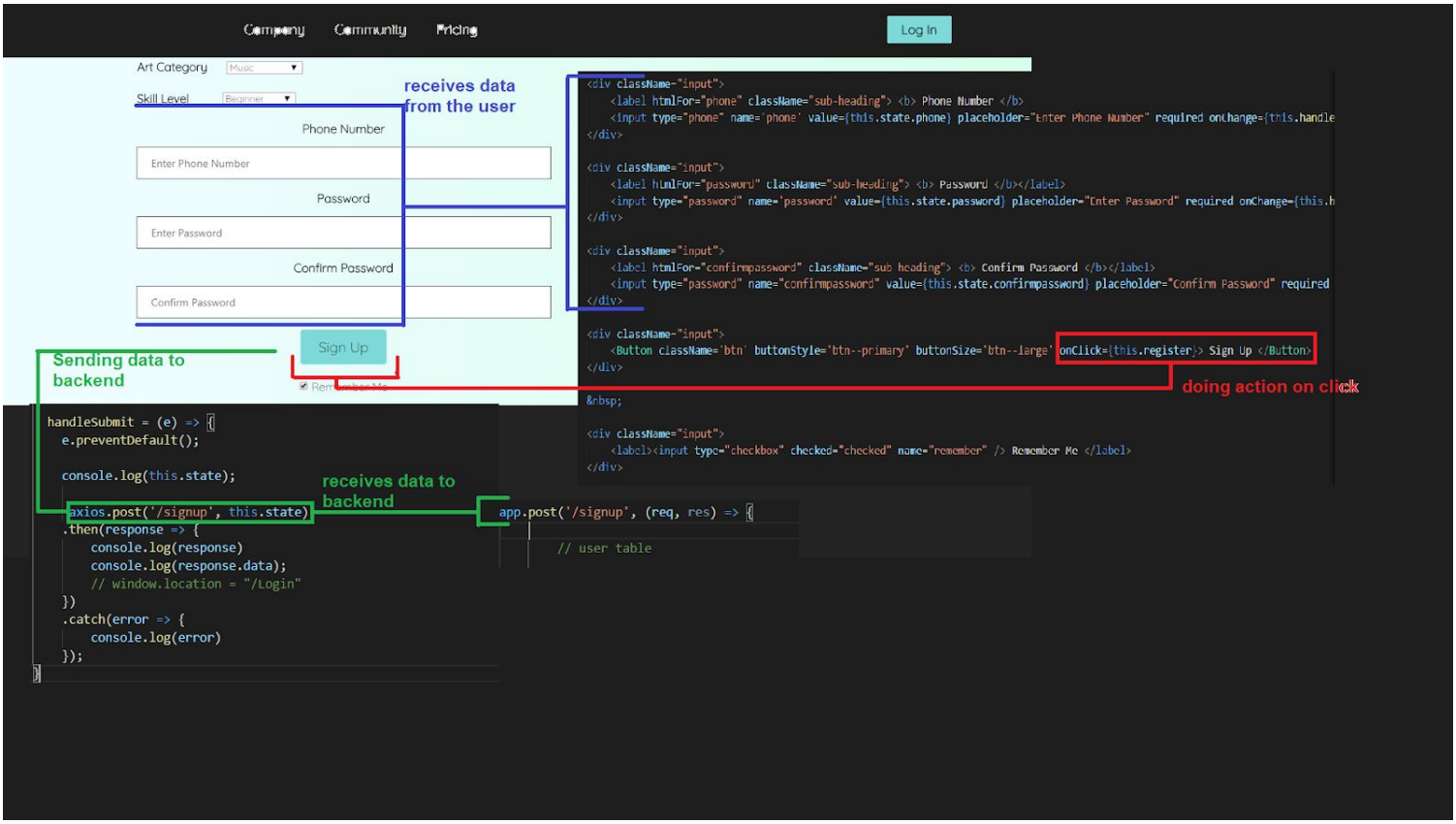
<label htmlFor="username" className="sub-heading"> Username </label>
<input type="username" name='username' value={this.state.username} placeholder="Enter Username" required

<label htmlFor="password" className="sub-heading"> Password </label>
<input type="password" name='password' value={this.state.password} placeholder="Enter Password" required

<Button className='btn' buttonStyle='btn--primary' buttonSize='btn--large' onClick={this.login} > Log In


login = e =>{
e.preventDefault();
axios.post('/login', this.state).then(response =>{
if(response.data){
this.props.history.push('/Community');
}else{
alert("Your username or password is incorrect");
}
});
sending data from
frontend to backend
app.post('/login', (req, res) => {
console.log(req.body);
var todb = "SELECT * FROM account WHERE (username = '" + req
pool.query(todb, (error, result) => {
if (result.length == 1) {
// console.log(res.redirect('/'));
req.session.username = result[0].username;
req.session.userId = result[0].user;
console.log(req.session);
res.send(req.session);
console.log("_____end1_____")
} else {
console.log("incorrect creds");
console.log(error);
console.log("_____0_____")
console.log(req.session);
res.send(null);
console.log("_____end2_____")
verify data from frontend
if correct allows user to
log in
account_id
username
password
1
tone10
2
tone12
$2b$10$V.1xyCyvmGprwwqHGX9IC.jcGDJNYuXMyuQkYNfUjvkm0jSSVd9k2
3
tone13
4
tone14
$2b$10$RNdDOzSuUOTffVT3nQNXhup.AeLgYqvZ2773dISEwr3DUYTr9ED.
5
tone15
6
tfinal00
123
password visible on purpose,
look into database upon given username

Sign Up


Company
Community
Pricing
Log In
Art Category
Music
Skill Level
Beginner
Phone Number
Enter Phone Number
Password
Enter Password
Confirm Password
Confirm Password
Sign Up
receives data from the user
Sending data to backend
receives data to backend
doing action on click
handleSubmit = (e) => {
e.preventDefault();
console.log(this.state);
axios.post('/signup', this.state)
.then(response => {
console.log(response)
console.log(response.data);
// window.location = "/Login"
})
.catch(error => {
console.log(error)
});
}
app.post('/signup', (req, res) => {
// user table

<label htmlFor="phone" className="sub-heading"> Phone Number
<input type="phone" name='phone' value={this.state.phone} placeholder="Enter Phone Number" required onChange={this.handle

<label htmlFor="password" className="sub-heading"> Password </label>
<input type="password" name='password' value={this.state.password} placeholder="Enter Password" required onChange={this.h

<label htmlFor="confirmpassword" className="sub heading"> Confirm Password </label>
<input type="password" name="confirmpassword" value={this.state.confirmpassword} placeholder="Confirm Password" required

<Button className='btn' buttonStyle='btn--primary' buttonSize='btn--large' onClick={this.register}> Sign Up </Button>

<label><input type="checkbox" checked="checked" name="remember" /> Remember Me </label>

3. Input Data Validation

```
var post_title = req.query.post_title;
var post_category = req.query.post_category;

var todb = 'SELECT * FROM communityPage WHERE post_title = ? AND post_category = ?;';
pool.query(todb,[post_title, post_category] ,(err, result) => {
    if (err || result == ''){
        console.log(err);
        console.log("searching fail");
        res.send(err)
    }else{
        //console.log(result);
        res.send(result);
        console.log("searching pass");
    }

})
});
```

when there are no results

comm_pg_id	first_name	last_name	post_title	post_category	post_file	post_votes
NULL	NULL	NULL	NULL	NULL	NULL	NULL

OR

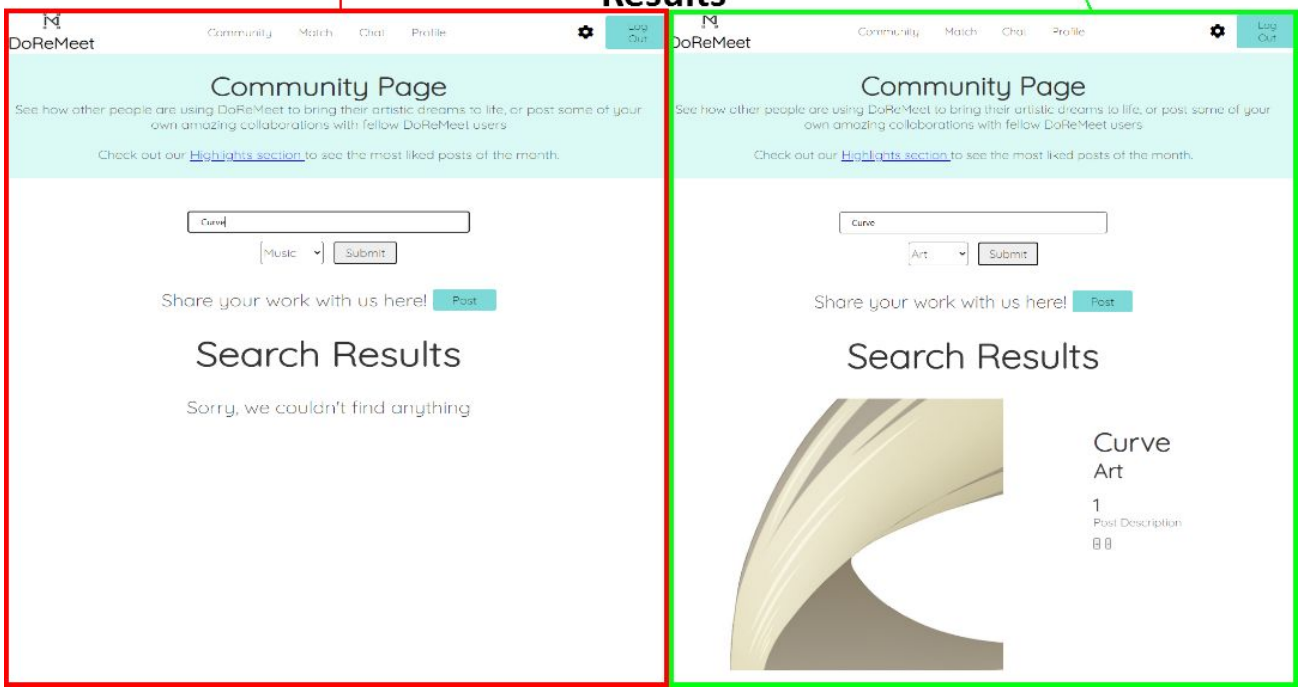
when result exists

comm_pg_id	first_name	last_name	post_title	post_category	post_file	post_votes
6	NULL	NULL	Curve	Art	2020-12-07_13-57-36.png	1

```
//Sends the search terms to the backend and gets the response
const submitHandler = (event) => {
  event.preventDefault();
  alert("You are searching for " + stateObj.searchTitle + " " + stateObj.searchCategory);

  axios.get('/searchPost', { params: { post_title: stateObj.searchTitle, post_category: stateObj.searchCategory } }).then(response => {
    if(response.data.length > 0){
      console.log(response.data.length);
      let _html = "";
      _html += `<h1>Search Results</h1>`
      _html += `<div class="RecentPostsFormat">
                <img class="PostImage" src="assets/postImages/${response.data[0].post_file}" alt="Post Image"/>
                <h2 class="PostTitle">${response.data[0].post_title}</h2>
                <h3 class="PostCategory">${response.data[0].post_category}</h3>
                <h4 class="PostVotes">${response.data[0].post_votes}</h4>
                <p class="PostDescription">Post Description</p>
                <p hidden class="PostID">${response.data[0].comm_pg_id}</p>
                <button id="PlusButton" type="button">+</button>
                <button id="MinusButton" type="button">-</button>
                </div>`;
      document.getElementById("search-post").innerHTML = _html;
      document.getElementById("recent-posts").innerHTML = '';
      document.getElementById("PlusButton").addEventListener("click", voteplus);
      document.getElementById("MinusButton").addEventListener("click", voteminus);
    }else{
      let _html = "";
      _html += `<h1>Search Results</h1>`
      _html += `<div class="NoResult">
                <p>Sorry, we couldn't find anything</p>
                </div>`;
      document.getElementById("search-post").innerHTML = _html;
      document.getElementById("recent-posts").innerHTML = '';
    }
```

Results



6. Self Check: Adherence to Non Functional Specs

** **Statements in red** -> These Non-Functional requirements are not implemented given time, resource, and knowledge limitations. Some of these (especially those pertaining to maintenance) cannot be implemented given that we will only have test users for M5, and there is not enough time to do actual maintenance. These requirements were defined for a product that is fully implemented and meets industry standards.

** **Statements in Purple** -> These Non-Functional requirements have not been implemented because they are related to Priority 2 and Priority 3 Functional requirements. Given those Priority 2 and 3 functionalities have not been implemented, the corresponding Non-Functional requirements have also not been fulfilled.

Security:

1. Accounts shall optionally have 2 factor identification enabled.
2. User's password shall be encrypted in the database. DONE
3. The site shall verify the users' credentials before successfully logging them in. DONE
4. The website will have a SSL certificate (https://).
5. The website shall have OAuth 2.0 implemented.
6. Accounts shall only have one password. There shall be no master key. DONE
7. The website shall warn users when clicking on links that take them to another website. DONE
8. Premium account users' payment information shall be encrypted in the database.

Storage:

9. Each account shall be able to store up to 80mb of associated content. ON TRACK
10. Each premium account shall be able to store up to 120mb of associated content.
11. A single media shall be less than or equal to 10mb. ON TRACK
12. User's data shall be saved into the MYSQL database. DONE
13. Users data should be removed from the database when the user deletes their account. ON TRACK
14. User accounts that have not been logged in for 1 year shall be disabled.

Availability:

15. The site shall be up and running at all times excluding system failures and scheduled maintenance. DONE
16. The site shall notify of all scheduled maintenance, 1 week in advance.
17. The users shall be notified via email about all scheduled maintenance, 1 week in advance.
18. Any/all scheduled maintenance shall be scheduled past 10pm PST on Mondays.

Functionality:

19. The website shall be hosted and deployed on the AWS server, EC2 instance. DONE

20. Account details on the database shall be modifiable. DONE
21. User details on the database shall only be modified by the user themselves. DONE
22. Only flagged user details, that have been reviewed, shall be modified by the admin.
23. The user shall log in using either email or username. DONE
24. The user shall be required to enter both password and username to log in. DONE
25. The user shall need a valid email to create an account. DONE
26. Users should be minimum 18 years of age to create an account. DONE
27. The users' password shall be censored on the sign in page. DONE
28. The users' password shall be censored on the sign up page. DONE
29. The website shall be responsive to user input. DONE
30. A premium account shall accept Visa, MasterCard and American Express.
31. A premium account shall accept GooglePay and ApplePay as a valid payment method.

Fault Tolerance:

32. Upon a system crash a global error page shall be displayed for all users trying to access the site. DONE
33. There shall be a backup for the website before any update is pushed. ON TRACK
34. There shall be a backup for the database every week on Tuesday past 10:00pm PST.
35. Error messages shall be relevant to the user (ex. "Site is down at the moment" or "Your account has been suspended" etc.)

Scalability:

36. The number of accounts that the system shall be able to handle should be scalable based on consumer demand/traffic. ON TRACK

Privacy:

37. Developers and staff shall not have direct access to sensitive user information. DONE
38. Anyone without a registered account shall not be able to view profiles of registered users. DONE
39. The site shall accept cookies/cache only if the user explicitly allows it. ON TRACK
40. Users shall not be able to see other users' password. DONE
41. Users shall not be able to see other users' email. DONE
42. Users shall not be able to see other users' phone number (if provided). DONE
43. Users data shall not be shared with any third-party applications/sites/organizations. DONE
44. Users data shall not be sold to any third-party applications/sites/organizations. DONE
45. The site shall not gather any unnecessary data from users. DONE
46. Users' geolocation shall only be known by the website if they allow it. DONE
47. Premium account users' payment information shall only be visible to that particular user.

Compatibility:

48. The site shall be compatible with Safari (version 10.0 - 14.0), Chrome (version 80 - 85.0) and Mozilla Firefox (version 78 - 81) and Edge (version 85). DONE

49. The site shall be compatible with Mac OS (10.12.6 - 10.15), Windows (7 and 10) and Linux. DONE
50. The site shall be compatible with mobile browsers on both iOS and Android. DONE

Look and Feel:
51. The site shall maintain a core color scheme. DONE
52. The website shall have a functional navigation bar on every page. DONE
53. The website shall have a user settings page. DONE
54. The website's logo shall direct the user to the main home page. DONE
55. The website shall be mobile-friendly. DONE

Ease of Use:
56. The site shall maintain less formal vocabulary and diction. DONE
57. A preview of what the site has to offer will be displayed on the site landing page. DONE
58. Website UI shall be simple and efficient. DONE
59. Website UI shall be easy to use. DONE
60. Website UI shall be intuitive. DONE
61. Users shall be able to find what they need on the website in 5 seconds. ON TRACK

Coding Standard:
62. The whole site shall use the same coding standards related to code indentation. DONE
63. All functions shall be modularized. DONE

Marketing Requirements:
64. The website shall not make any political marketing campaigns. DONE
65. The website shall have a logo on the navigation bar. DONE
66. The website shall have a logo on the "tab" for site identity. DONE

Legal Requirements:
67. The site shall explicitly not endorse any activity that is illegal, according to the U.S law. DONE
68. Access to the website's policies shall be available at the bottom of every webpage. DONE
69. The site shall protect all user data under the privacy laws in the U.S. DONE
70. The site shall be transparent about the use of user data. DONE
71. An unregistered user shall have to agree to the website's terms & conditions to successfully create an account. DONE

Performance:
72. Match results shall be filtered in at least 1 second. ON TRACK
73. The website's response time shall be at most 1 second. ON TRACK
74. 2 factor identification code shall be sent to the user's phone in at most 5 seconds.

Expected load:
75. The system shall be able to handle up to 1000 users at once.

7. List of Contributions

Member	Role	Contributions
Nimiksha	Team Lead & Documentation Lead	Product Summary Usability Test (shortlisting requirements and write up) QA Test plan (write up and ran all the QA Test) Code Review (gave review) Self-Check Adherence to Non-Functional Requirements Code Cleanup to meet Coding standards established M2, M3 Version 2 Documentation M4 Documentation
Meet	Backend and Database Lead	QA Testing (for allow location feature) Fixed Log in password encryption in the database Self-Check on best Security Practices (Password Encryption) Self-Check Adherence to Non-Functional Requirements Configure the LocationIQ API to grab user's location Code the expand and reduce search radius feature for potential matches Documentation Review
Luke	Frontend Lead	QA Testing (for sign up - Date of Birth Validation) Self-Check Adherence to Non-Functional Requirements Coded the Chat page to have a more friendly and intuitive UI Updated the Profile page to make the user information editable and add media files Built the global 404 Error page for the website Host updated version of the product on EC2 instance Documentation Review
Mike	GitHub Master & Backend Engineer	Self-Check omn best Security Practices (Data Validation) Self-Check Adherence to Non-Functional Requirements Configured the matching algorithm and logic Implemented the Match page to show potential matches to the user Fixed pathing errors and errors with uploading images to Community Page Updated Settings and Preferences backend to accept user changes Documentation Review
Jungsun	Database & Backend Engineer	Self-Check Adherence to Non-Functional Requirements Worked on the Update password feature (user can safely update current password from Settings page) Password Recovery implementation Set up redirects for Log out Documentation Review
Vincent	Server Admin & Frontend Engineer	QA Testing - (for leaving website alert message) Self-Check Adherence to Non-Functional Requirements Worked on Match Page frontend (waiting for backend) Implemented Community page voting feature Upload Social media links on users' profile

		Documentation Review