**NUS Orbital Milestone 2024**



Team Capy – Lai Wing Hyun & Koh Wei En

**Table of contents**

[**Project Introduction 2**](#_efk6awm1mrc6)

[Proposed Level of Achievement 2](#_d56lnzbtlylw)

[Aim 2](#_h3lh8gzw8de)

[Motivation 2](#_y42g45ou8jtc)

[How are we different from similar platforms? 2](#_s8rkb26j8sat)

[User stories 3](#_bvq8l1m3ltah)

[Development plan 3](#_gd68rdolgfqa)

[**Scope of Project: 4**](#_49izf8gm7sc5)

[Feature 1: Authentication (Login + Create Account) 5](#_rmi8tesp62zg)

[Feature 2: Create Post 7](#_mv3m9a93in16)

[Feature 3: Feed 9](#_hk0kwfbgxs40)

[Feature 4: Direct Message 11](#_7q8wcapxtnb4)

[Feature 5: Profile 13](#_umr17lfp7pkx)

[Feature 6: My applications 15](#_tuv54va14haw)

[Feature 7: My posts 16](#_447e3puwu2d5)

[Feature 8: Reviews 17](#_8qayemowugh4)

[Feature 9: Insights 19](#_iea1iypbl5zx)

[**System Architecture 21**](#_alq7wtosqsgv)

[Code Base 21](#_sig8lyrrtzm4)

[Tech Stack 21](#_7ujnjfpsd9gt)

[App optimisation 21](#_a9tw9kd3ex0h)

[Vercel + MongoDB deployment regions 21](#_vrwnuhika4dr)

[MongoDB document structuring 21](#_s3vrq2p1kn8l)

[Lazy loading of components 23](#_pt36fjjvknht)

[**Database use and structure 23**](#_q7yrrvdff8vq)

[**Design Decisions 24**](#_wcrb5yog93db)

[User flow diagram 24](#_6s0d1pez3ijt)

[**Testing 25**](#_yjjg81rd0aj0)

[System testing 25](#_jjsor8a7wzuu)

[Unit testing 29](#_8z5jod75ium5)

[User testing 30](#_id609vj6qbgd)

## 

## Project Introduction

### Proposed Level of Achievement

Project Apollo

### Aim

To assist students in matching up with other students to fulfil their academic needs, be it for tutoring or project groupmate finding purposes.

### Motivation

We noticed that it is difficult for people to find group mates who are compatible with them for group projects. Furthermore, we also noticed how many students are struggling with their classes and assignments. Therefore, we created this web app in order to match NUS students with other students in order to fulfil their academic needs of finding group work members and help for their subjects.

### How are we different from similar platforms?

There are no apps on the market right now combining both project group matching and matching tutor/tutees in one app. Therefore, we aim to be a one-stop platform to connect students with each other to fulfil all a university student’s academic needs.

Tutor/tutee matching

* Tuition telegram bots
  + There are a variety of telegram bots/channels that link students with tutors
  + However, posts with all levels of education are included in the channel, with very few postings involving university students looking for tutors.
  + In comparison, our app focuses solely on university students, so that university students do not have to sift through a large number of irrelevant postings. Due to economies of scale, whereby many university students are expected to post on the app, students can find tutors or tutees more easily.

Project group matching

* Canvas
  + There is a function where you can join groups that have vacancies available in canvas. However, you are unable to see a description of the people in the groups with vacancies which can make it hard for you to choose a group to join. Our app will allow students to understand more about the students that they would like to work with so that they can join a group that they would like to work with.
  + Groups with vacancies cannot vet and choose the people that want to join the group before they join. Our app will also allow students to reject applications to their post so that they can choose the most suitable candidate to be their groupmate.

### User stories

1. As a student who wants to be able to find compatible group members to do project work with, I want to be able to select group members to team up with without having to individually contact them first, while ensuring that group members are competent and that I can work well with.
2. As a student who wants to get a small part time job as a tutor, I want to be able to find other people who may be struggling in classes that I am strong in and to help tutor them or to help them while earning a small sum of money
3. As a student who is struggling with a class and needs a lot of extra help, I want to be able to find a tutor who understands my learning needs and can help me in a timely and dedicated manner

### Development plan

**3rd week of May:** Finalised pitch for Orbital Lift-off

**4th week of May:** Pick up the necessary technologies, do mockup of App, finish frontend of Login and create account feature

**5th week of May:** Finish backend of Login and create account features

**1st week of June:** Finalise minimum data sets and do diagrams

**2nd week of June:** Finish frontend and backend of profile creation and posting features

**3rd week of June:** Finish frontend and backend of feed and direct messaging features

**4th week of June:** Testing and debugging

**1st week of July:** Implementation of peer teams’ suggestions and improve on feedback from user testing

**2nd week of July:** Finish my posts and applications tab and review feature

**3rd week of July:** Finish insights feature + app optimisation

**4th week of July:** Testing and debugging

## Scope of Project:

Bumbledore provides a student-matching interface for students to find tutors and groupmates for projects. The features of Bumbledore are outlined below. The project can be accessed here: <https://bumbledore.vercel.app/>

### Core Feature 1: Authentication (Login + Create Account)

Our first complex feature is the Authentication feature, which includes the Login and Create Account feature. Users can then login to their accounts using their email and password. Upon account creation, the user will be assigned a randomly generated username to ensure anonymity.

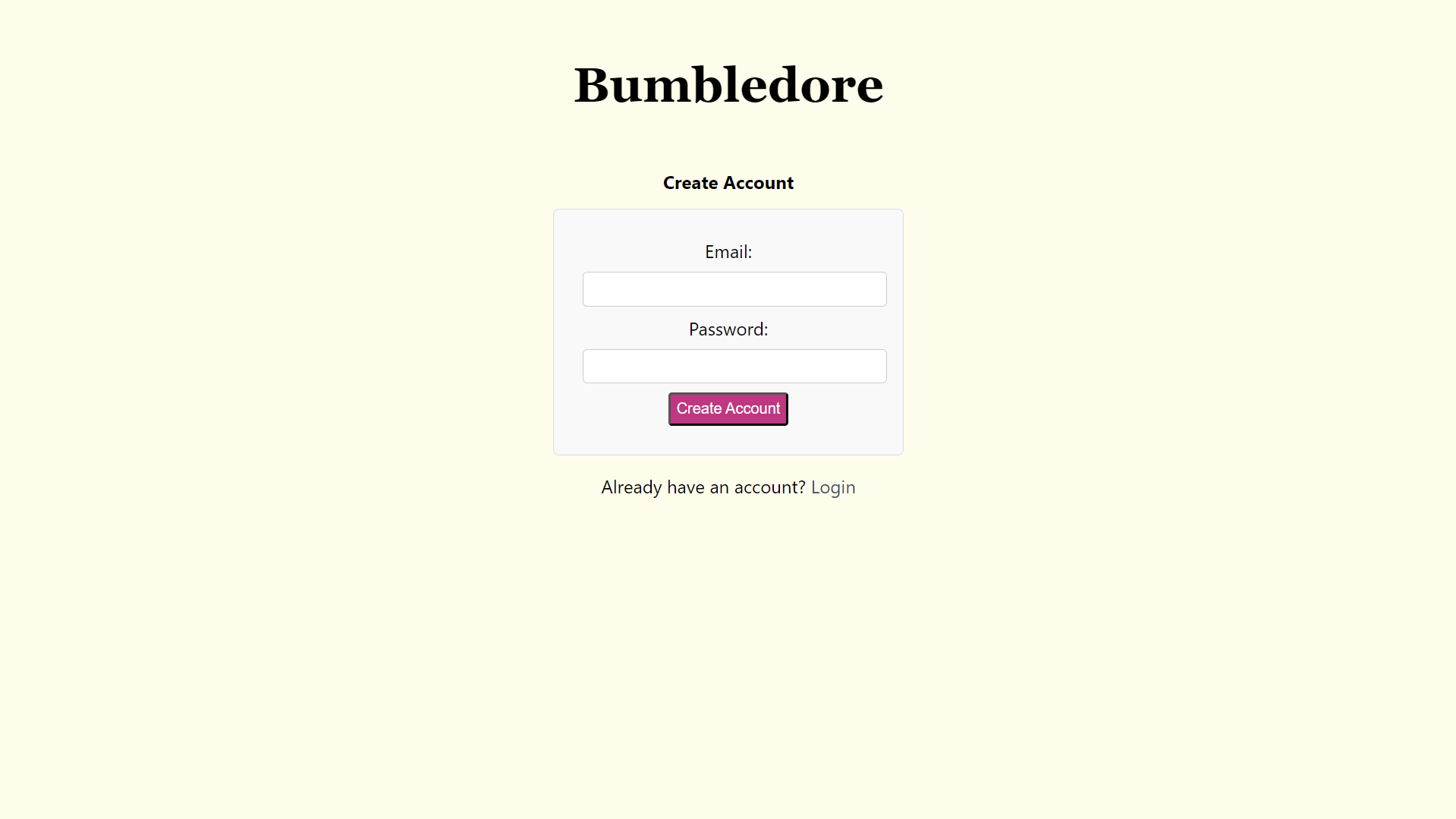


Figure 1: Create Account Page

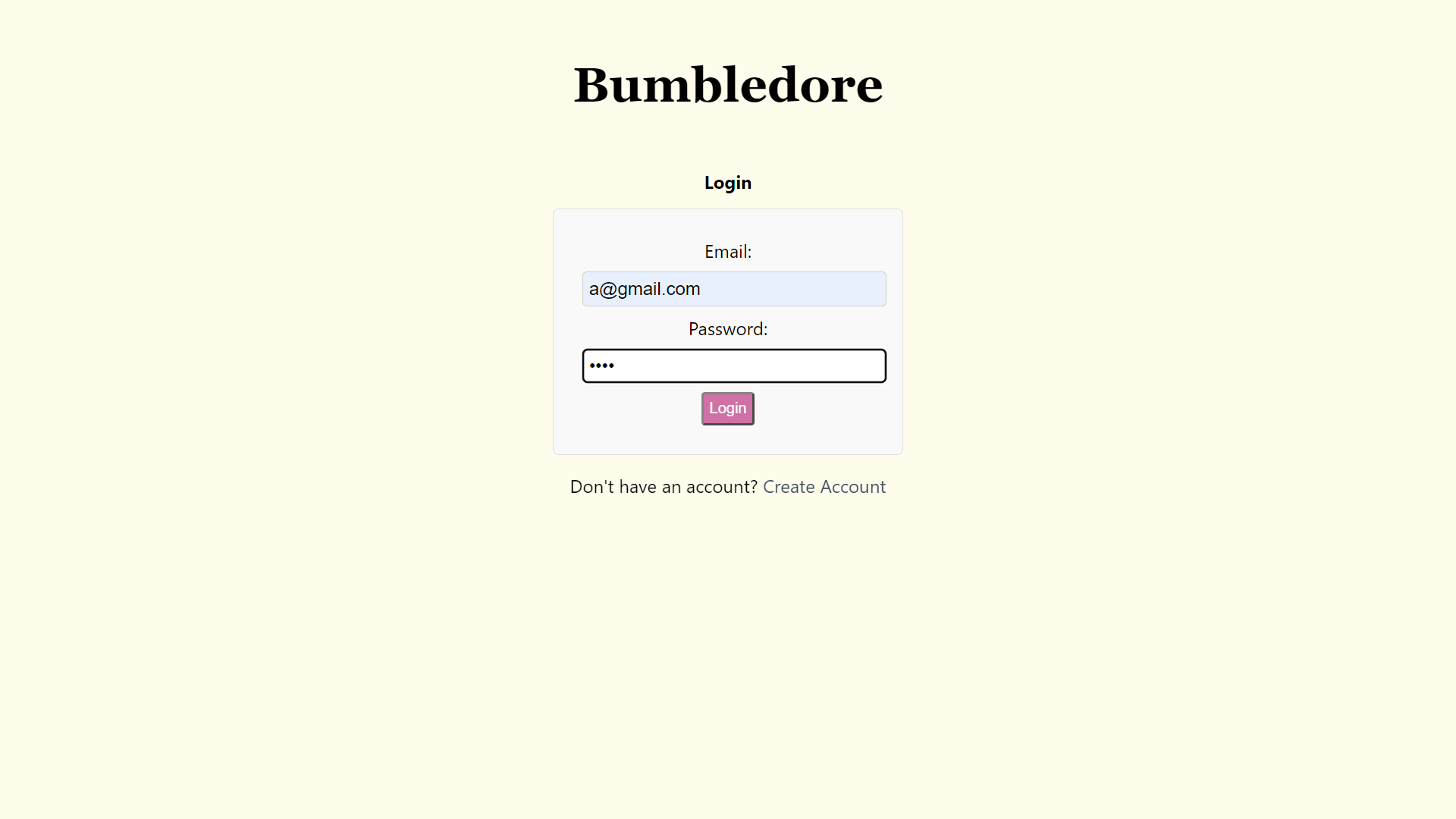


Figure 2: Login Page, with mouse hovering over ‘Login’ button

Milestone 1

In Milestone 1, we created this function by creating two pages, a login page and a create account page, each with its own form with email and password as form inputs, in our frontend React app. We also set up a collection, ‘userAccountInfo’, in our MongoDB database. After the user keys in their email address and password into the ‘/create-account’ page and submits the form, the email and password are added to the collection.

Thereafter, when a user attempts to login to the website through the login page, the email and password keyed in would be used to find a matching entry in the userAccountInfo collection. If a matching entry is present, the user is logged in, redirecting users to the home page to view the website. However, if a matching entry is absent, an error will show up prompting users to try again.

Milestone 2

We added usernames to each account created. To do this, a random unique username is generated by selecting a random adjective and animal from a list of adjectives and animals, and checking that other entries in the collection do not have the same username. Thereafter, with every account creation, the username is generated and stored in the database together with the email and password.

We also added session tokens that store the username of the user. Upon successful login, a token is stored in session storage containing the username of the user. This token is then used to authenticate the user and allow them to access the site.

Milestone 3

Through the feedback we received from Milestone 2, we realised that there were a few bugs in our application. Firstly, while creating a user token, the error message in the login feature disappeared. Furthermore, users were able to login through either the ‘/’ and the ‘/login’ url; however, upon successful login through the ‘/login’ url, the screen turned blank. We fixed both bugs in Milestone 3.

### 

### Core Feature 2: Create Post

Users can post an anonymous request stating whether they are looking for group members or tutors. Anonymising the request (i.e. no name, gender, race etc) is done to reduce bias and allows for selection of students based on relevant qualities. Other details such as number of group members they are looking for or number of people in their group can be included for the former, and specific assignments or topics they need help with for the latter.

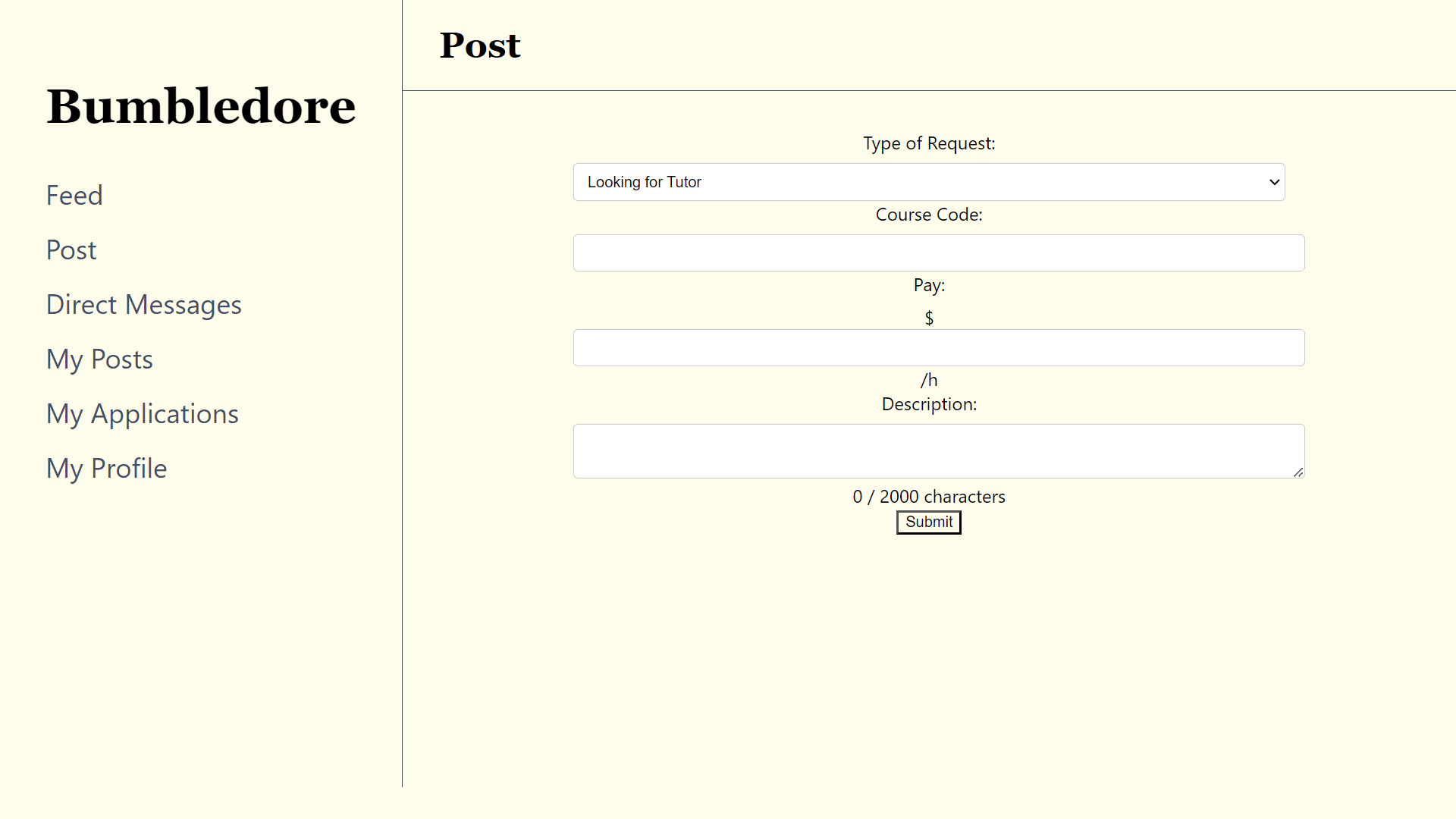
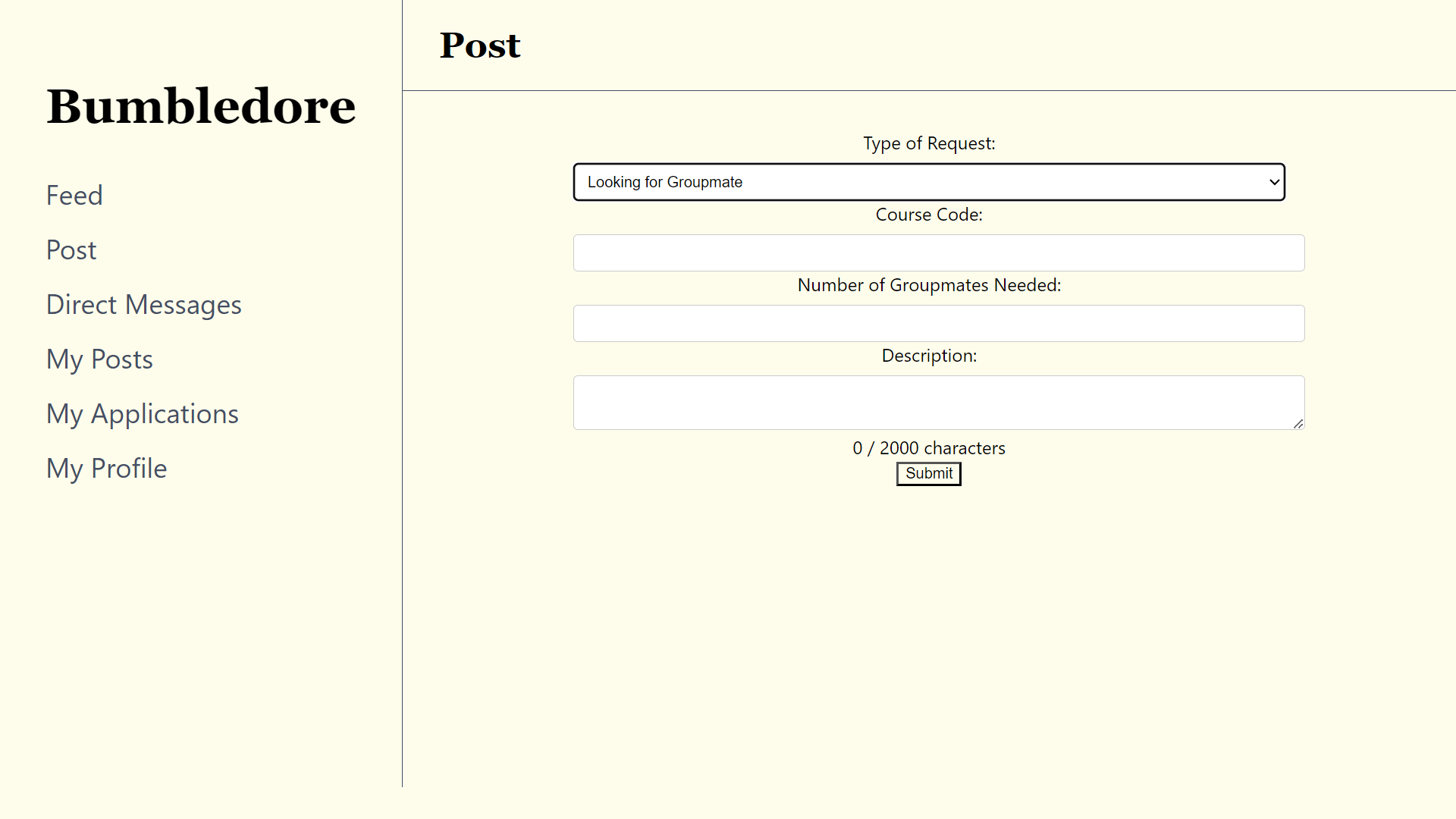
Figure 3: Post page, with ‘Looking for Tutor’ option selected under ‘Type of Request’

Figure 4: Post page, with ‘Looking for Groupmate’ option selected under ‘Type of Request’

Milestone 2

We implemented this feature in Milestone 2, by creating a page with a form containing inputs such as type of request, course, and description. Based on the type of request, we also included additional inputs, with an input for pay for requests looking for tutors, and an input for number of group members required for requests looking for project group mates. After form submission, the information is stored in a collection, ‘posts’, that we created in our MongoDB database. Additional information such as the time of creation of the post and the username of the user who created the post is stored as well.

Milestone 3

We added a status field in the document that stores the information of the post. Upon creation of the post, the post would have a status of ‘open’; this allows the post to show up on the feed.

### Core Feature 3: Feed

Users can scroll through the posts in their feed and select students to work with. Users can also click on a poster’s username to look through their profile. If users are interested in working with the student, they can click the ‘apply’ button. A message confirming the application will then be displayed in the feed, and a direct message will be sent to the poster informing them of their application; if the chat between the user and the poster has not been created yet, clicking ‘apply’ will create a chat. If the user is simply interested in knowing more about the application before applying, they can click the ‘message’ button, and a chat will be created with the poster.

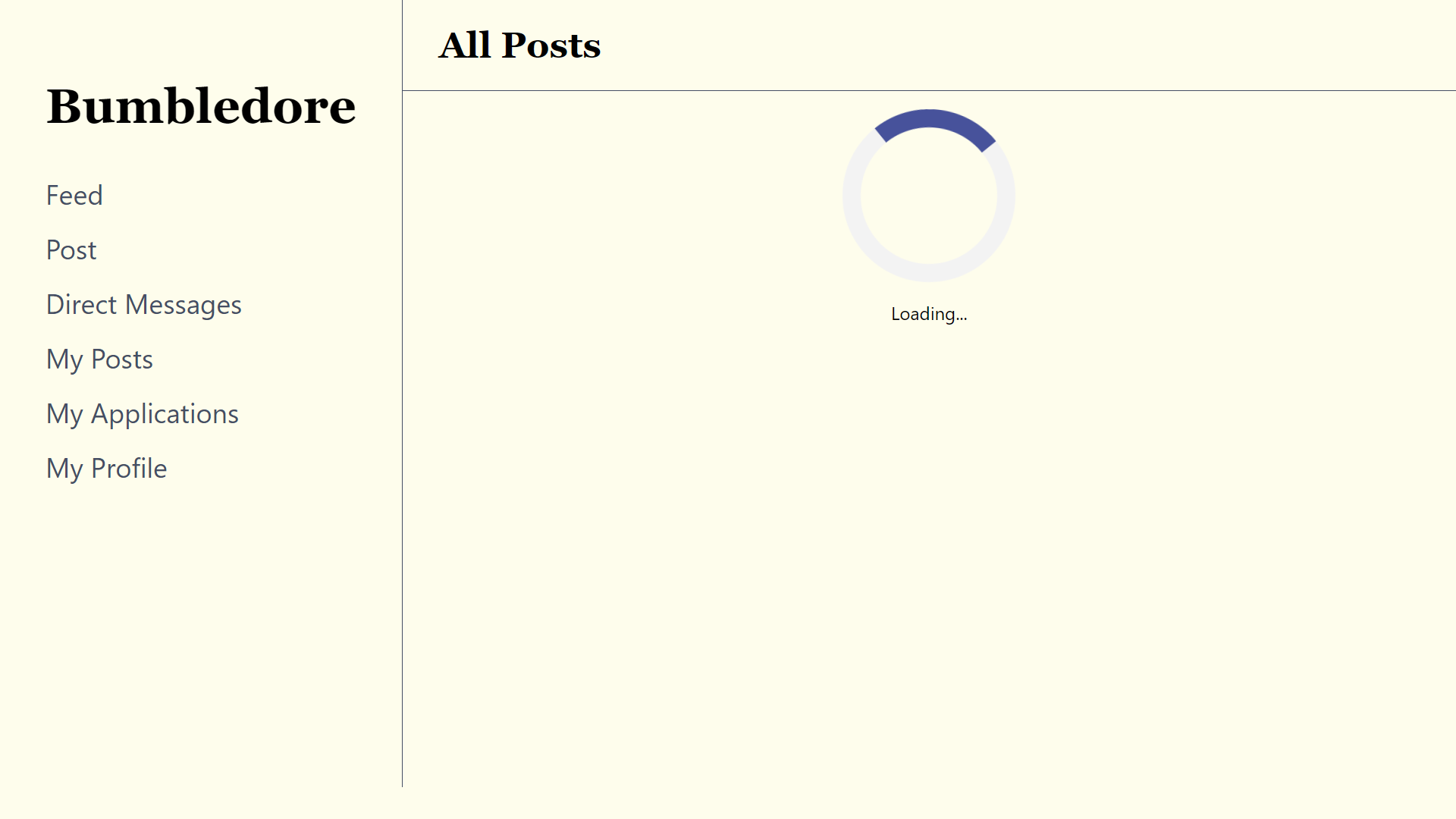


Figure 5: Home Page, with loading screen displayed as post data is fetched from database

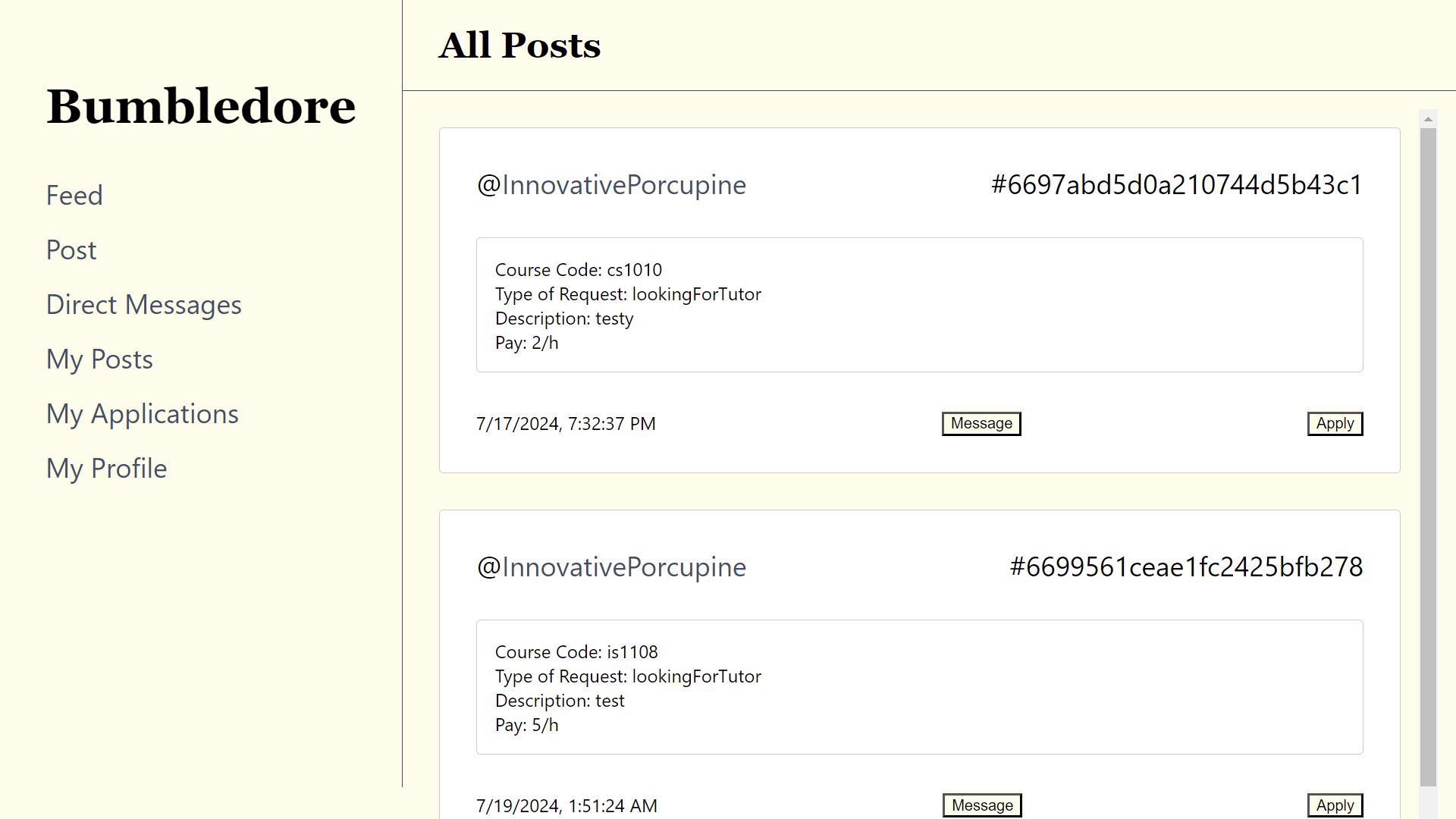
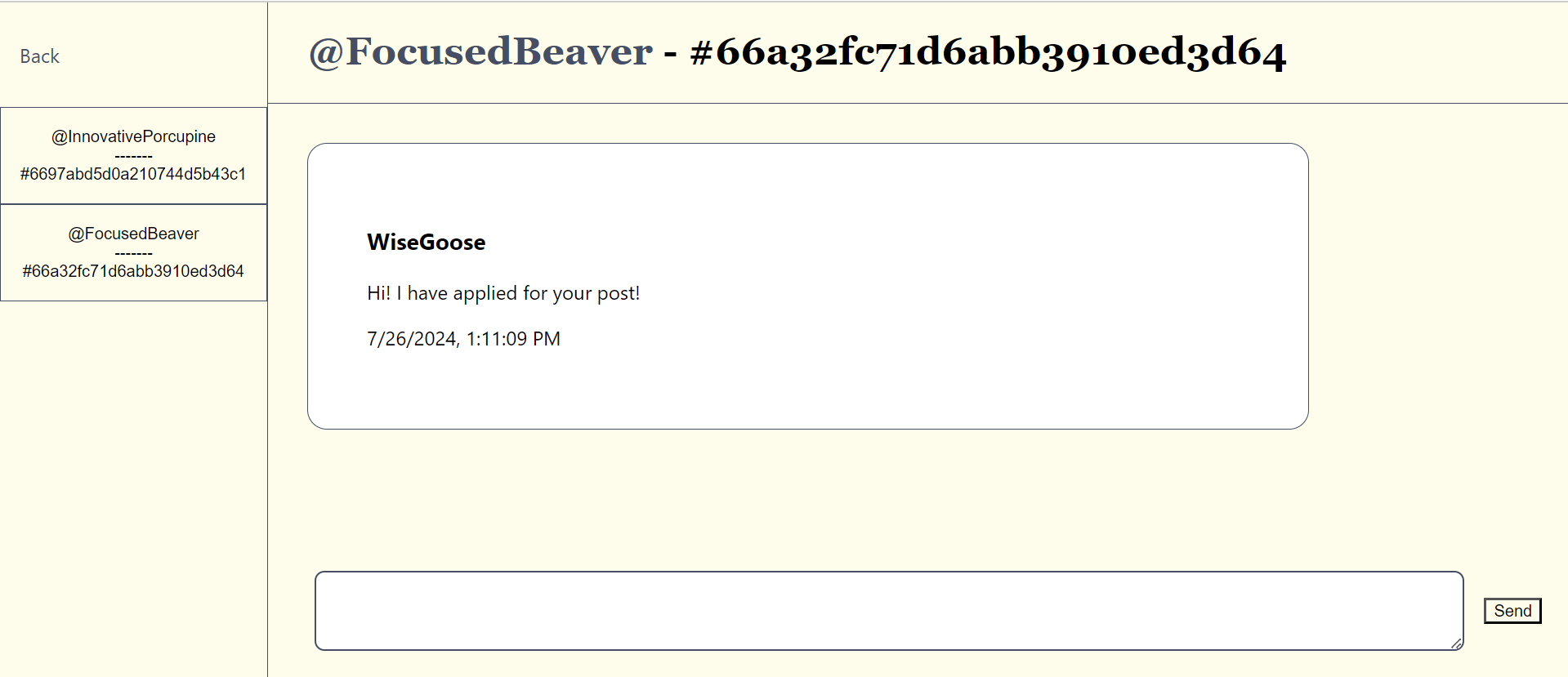


Figure 6: Home Page, with posts in scrollable feed and a sidebar

Figure 7: Direct Messages page, with chat created and automated message sent to inform poster of post application

Milestone 2

We implemented this feature by getting the post data from the ‘posts’ collection in our database and putting the data from each document in the collection in its own post. We also added a message button for interested applicants to message the poster. Upon clicking the button, the username of the applicant and the username of the poster are sent to the backend through data attributes, where the information is stored in the ‘chats’ collection in our database. This would allow for the creation of a chat between the poster and the applicant.

Milestone 3

In milestone 3, the status field was implemented such that all posts would have a status of either ‘open’, ‘closed’ or ‘successful’. Posts that are ‘open’ would appear on the feed. Furthermore, each post also stores an array of applicants as well as the accepted applicant in the ‘posts’ collection, which will be used to update the status of the post in the my applications and my posts features.

### 

### Core Feature 4: Direct Message

Users can create a chat by clicking on the ‘message’ button in a post located in the feed. Upon clicking into the messages tab, the chats that a user has with other users will appear on the sidebar of the page. To view the messages in a chat, click on the username and post id. A text bar is present at the bottom of the screen, allowing users to type in a message and send it to the user.

After interested parties create a chat with the poster, the poster can message the interested parties, inform them of their interest in working together and exchange contact information.

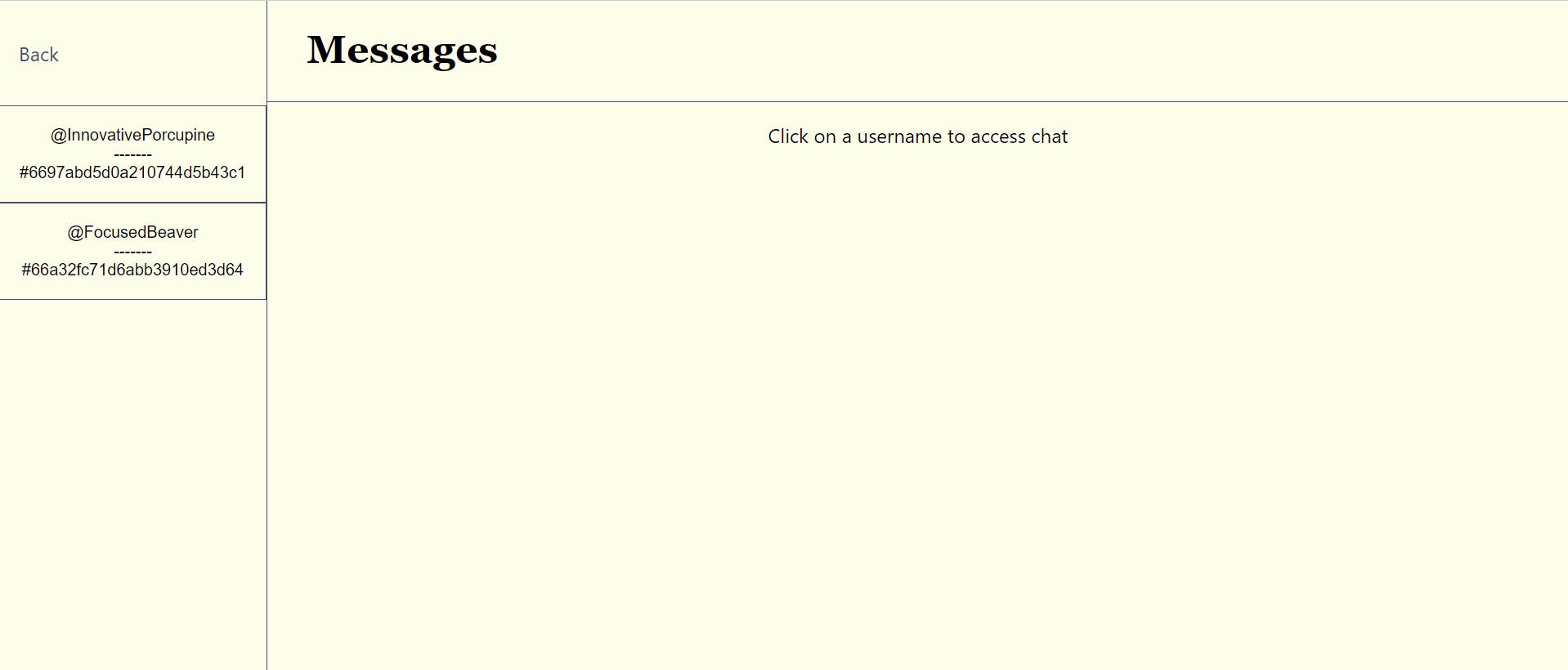
Figure 8: Direct Messages Page, with chats in sidebar that can be accessed upon clicking the chat

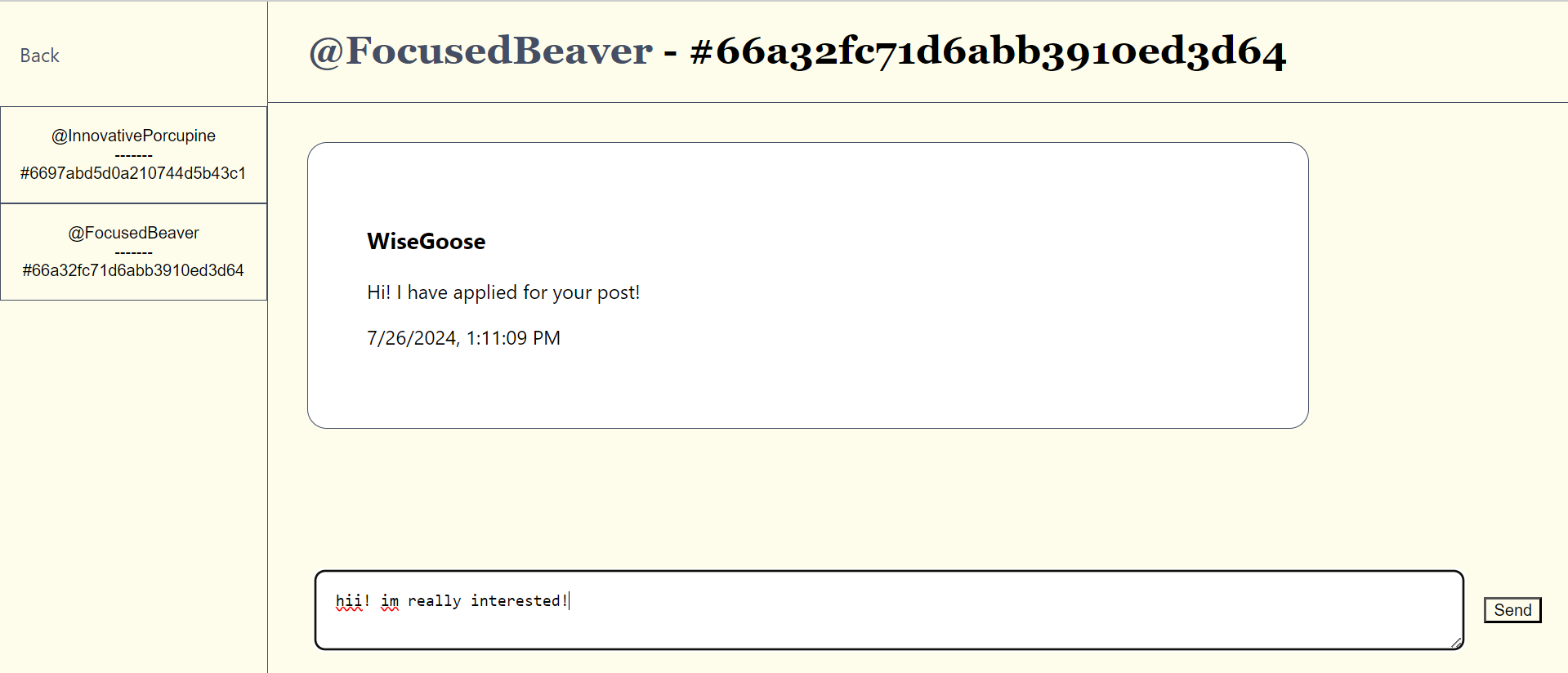
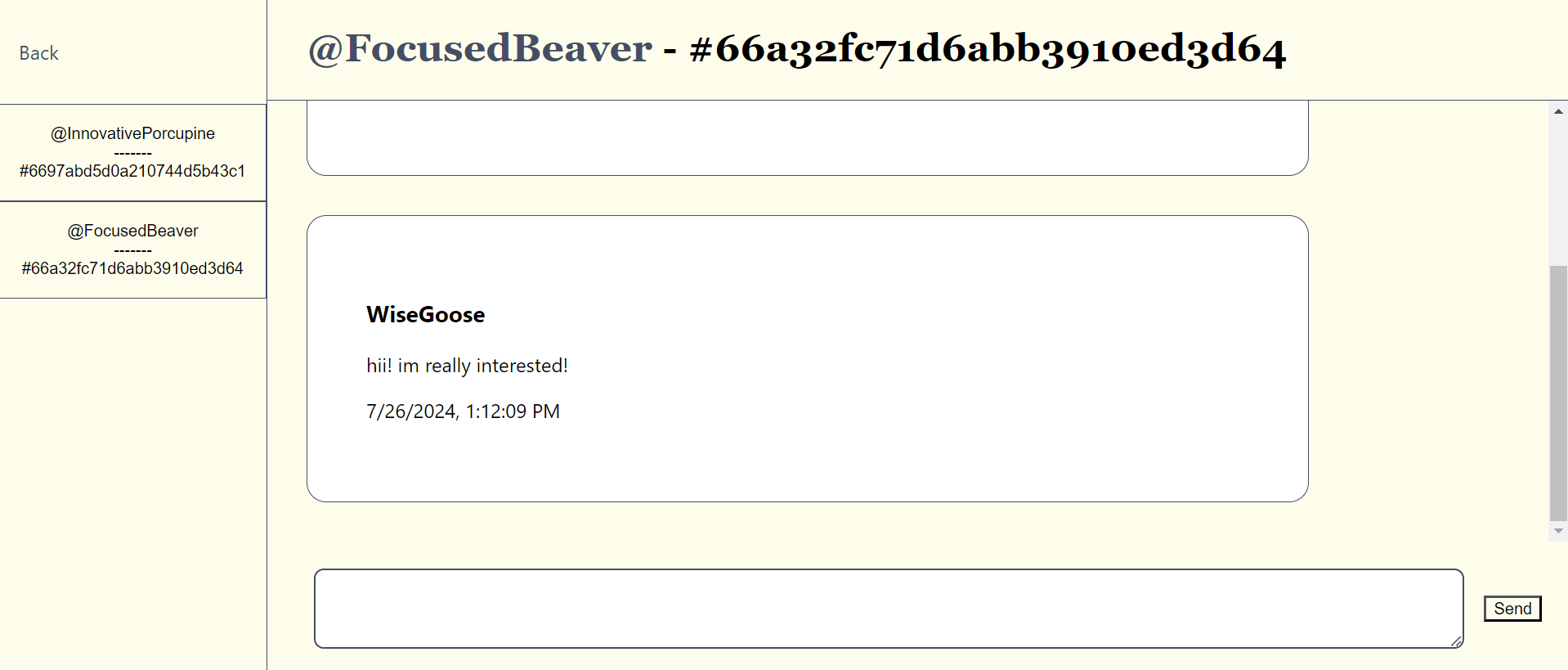
Figure 9: Direct Messages Page, where users can view and type messages in input bar

Figure 10: Direct Messages Page, where messages sent can be viewed in the chat

Milestone 2

We implemented this feature in milestone 2 by creating two collections: ‘chats’, which stores information about the list of chats that a user has with others, and ‘messages’, which stores the messages that users send to and from each other. In the messages page, the sidebar containing the chats that a user has will be taken from the ‘chats’ collection and displayed. Upon clicking into the chat, the messages between the two parties are loaded and displayed in chronological order. These messages are taken from the ‘messages’ collection. The text bar at the bottom of the screen is an input form taking in text, and clicking submit will send the text data, as well as the usernames of the two parties, into the ‘messages’ collection.

Milestone 3

We added the postID field into the documents of both ‘chats’ and ‘messages’, in order to facilitate the creation of different chats for the different posts that a user may make. Also, after applying for a post in the feed, a message would be sent informing the poster of the request.

### Core Feature 5: Profile

Users can click into the profile tab displaying the profile information of the user; if the profile has not been created yet, the fields would be empty. To set up a profile, users can click ‘Edit Profile’ and fill in a form with inputs taking in their year, course and description of themselves and submitting the form. Subsequently, users can view other peoples’ profiles by clicking on their usernames in the feed or in direct messages. Users can also view their profile insights by scrolling to the bottom of the page.

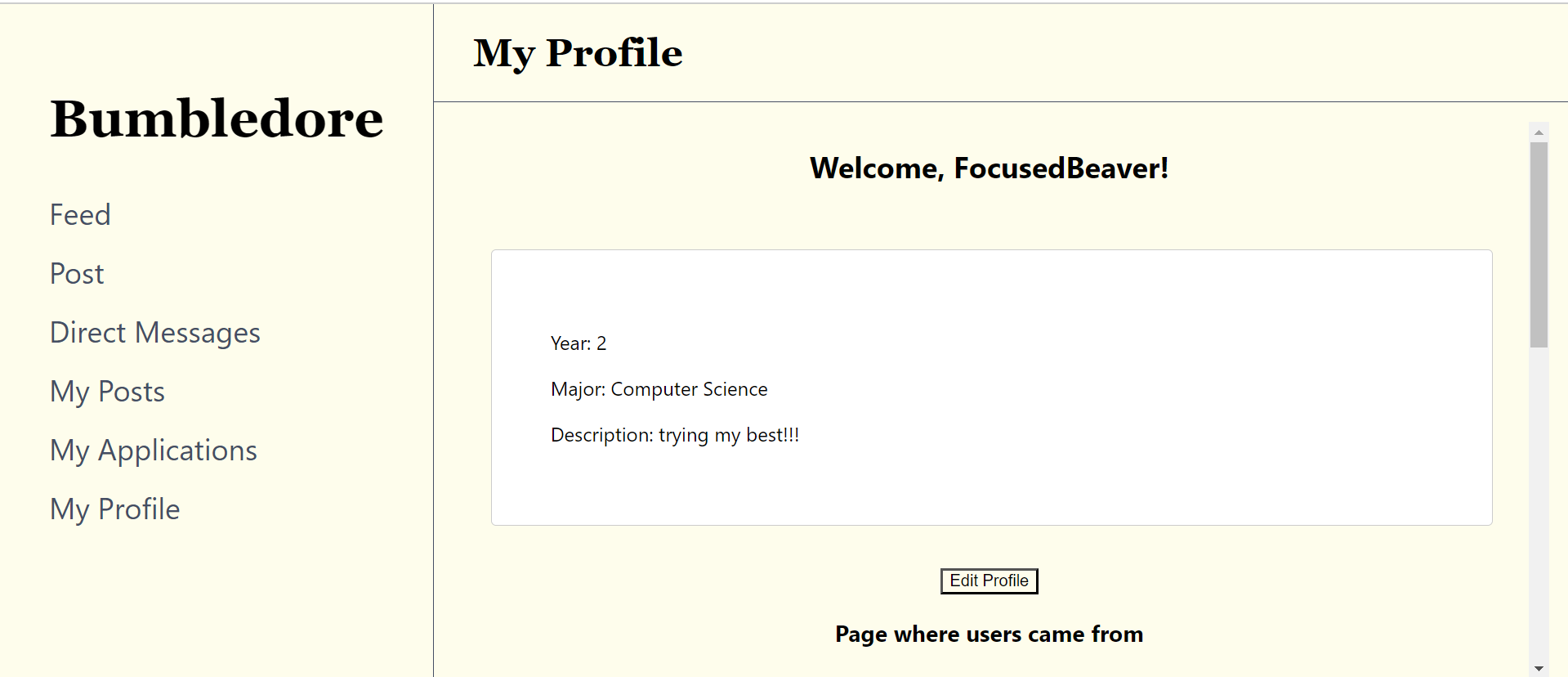
Figure 11: My Profile Page, where profile information can be viewed above

Figure 12: My Profile Page, where user can edit profile upon clicking ‘Edit Profile’ button

Milestone 2

We implemented this feature in milestone 2, whereby users can create a profile by filling in a form with year, course and description as form inputs. Upon form submission, users will be taken back to their profile page, where they can view their own profile information. The information from the form would be added as a document into the ‘userProfileInfo’ collection we made in our database, where it is subsequently retrieved and displayed in the user’s profile. For users viewing their own profile, we made use of the authentication token to find the user’s profile information in the database, as well as displaying the user’s own username in the profile page.

Milestone 3

We added an insights feature to display 2 bar charts. The first bar chart displays the number of profile clicks from different pages, such as the Home page, the Direct Messages page and the My Applications page. The second bar chart displays the number of profile clicks from different posts.

### Extension Feature 6: My applications

Users can view the status of the posts that they have applied or are applying for in this tab. If the poster has accepted their application, they will be able to view the status of the post as “successful”, if the poster has rejected their application, they will be able to view the status of the post as “closed”.

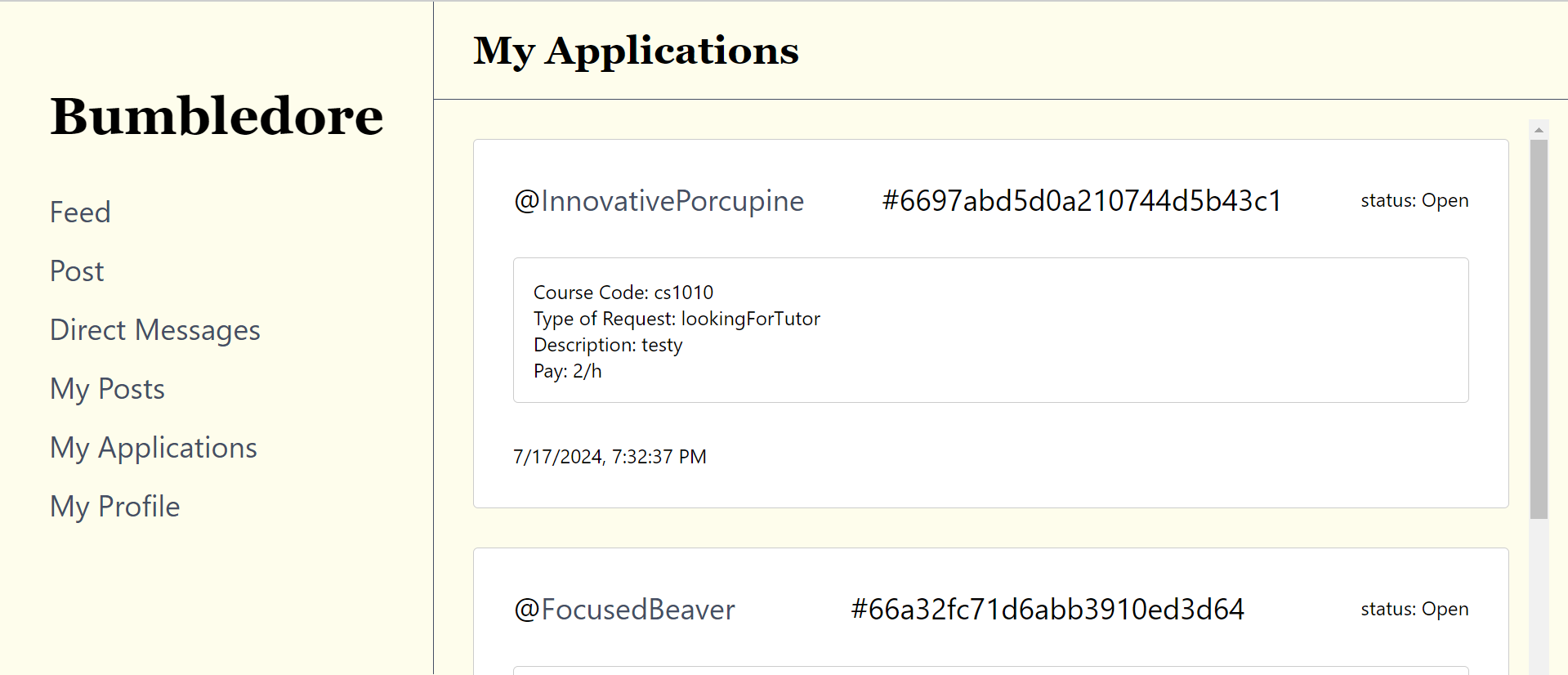


Figure 13: My Applications Page, where user can view the posts that they have applied for, as well as their statuses

Milestone 3

In milestone 3, the status field was implemented such that all posts would have a status of either ‘open’, ‘closed’ or ‘successful’. All posts that the user has applied for of various statuses would be displayed in the My Applications tab. To see if the user has applied for this post, each post stores an array of applicants, and the posts retrieved from the database are thus filtered for the user appearing in the array of applicants. Furthermore, if the user is an accepted applicant, the status of the post would be displayed as successful.

### Extension Feature 7: My posts

In this tab, users can view the posts they have made and their post insights.

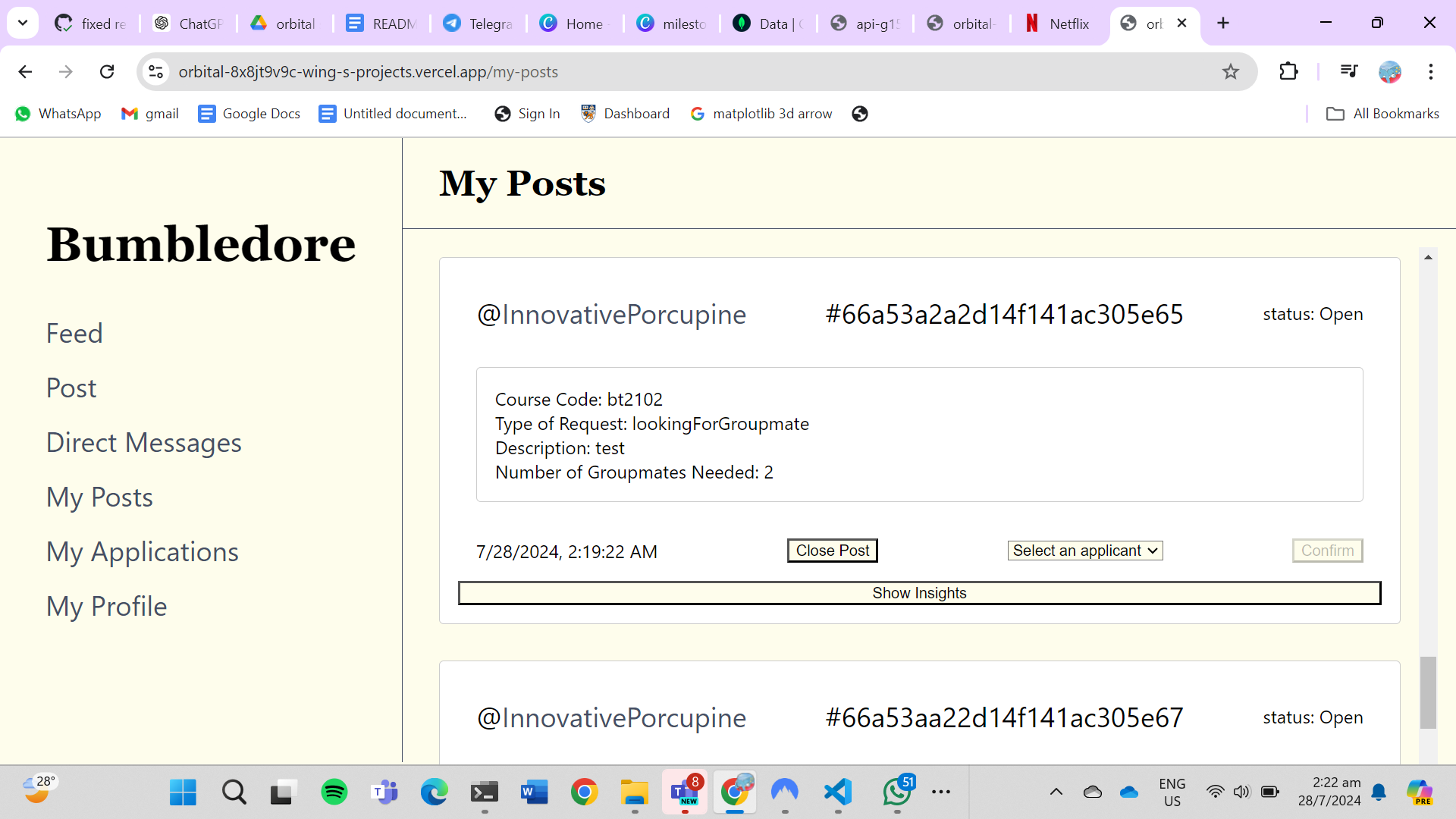


Figure 14: My Posts page, where users can view the posts that they have created, their statuses, and can close posts and select applicants

Milestone 3

Users can view their open and closed request posts in this tab. In addition, they can edit their request posts, and close their request posts in this tab. Users will be able to see the users who applied for this post, and be able to select an applicant to accept. Once successfully selected, the user will be able to see the status of their post as “successful” in their My Applications page.

There is also a ‘Show insights” button on each post, upon clicking this button, users can see a bar chart of the number of clicks for the “Apply” button, “Message” button and profile link for that post.

### 

### Extension Feature 8: Reviews

In this tab, users can leave reviews for each other after being successfully matched. Reviews can be seen when you click on each others’ profiles.

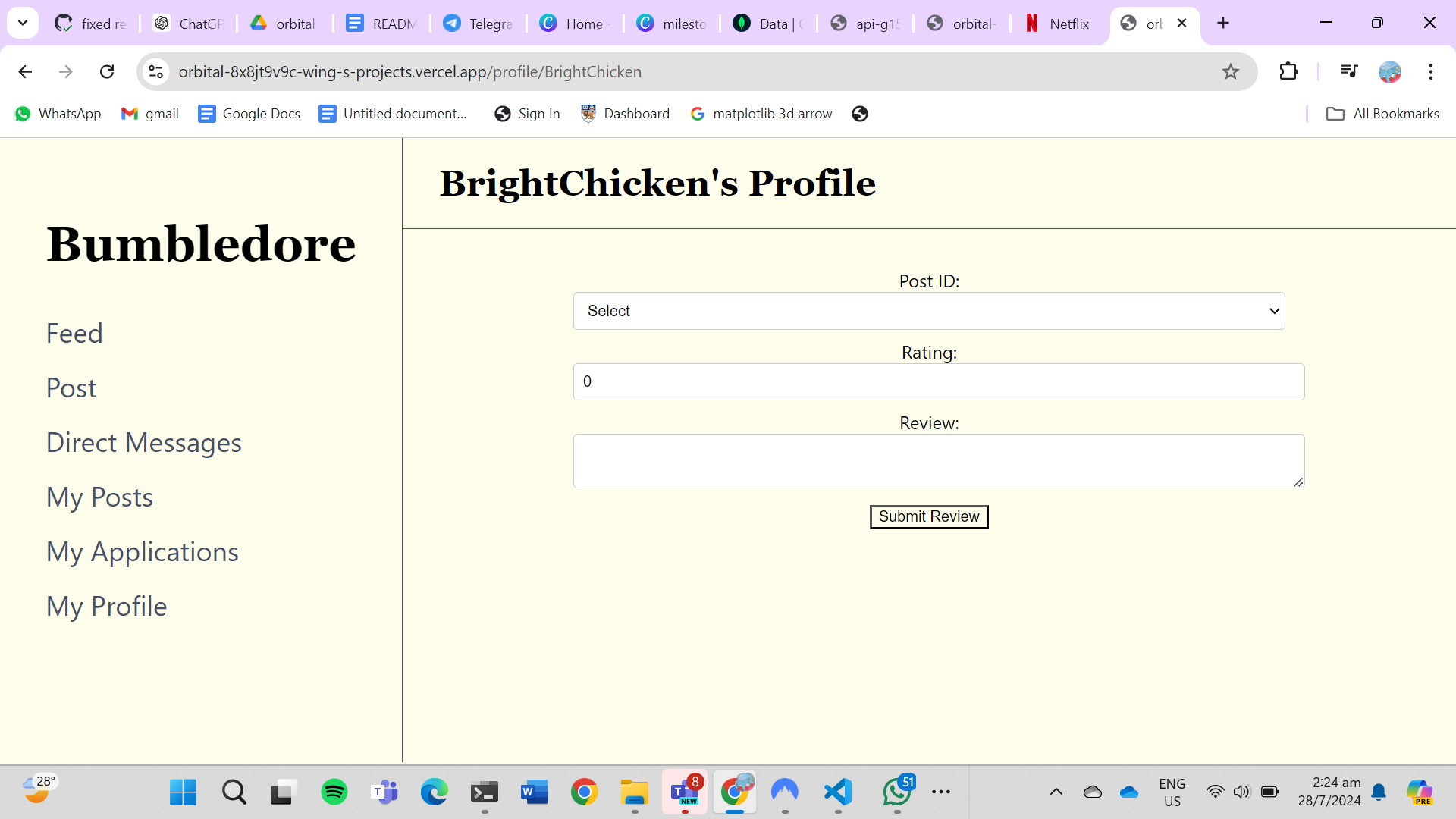


Figure 15: Reviews Page, where users can write reviews for users that they have worked with

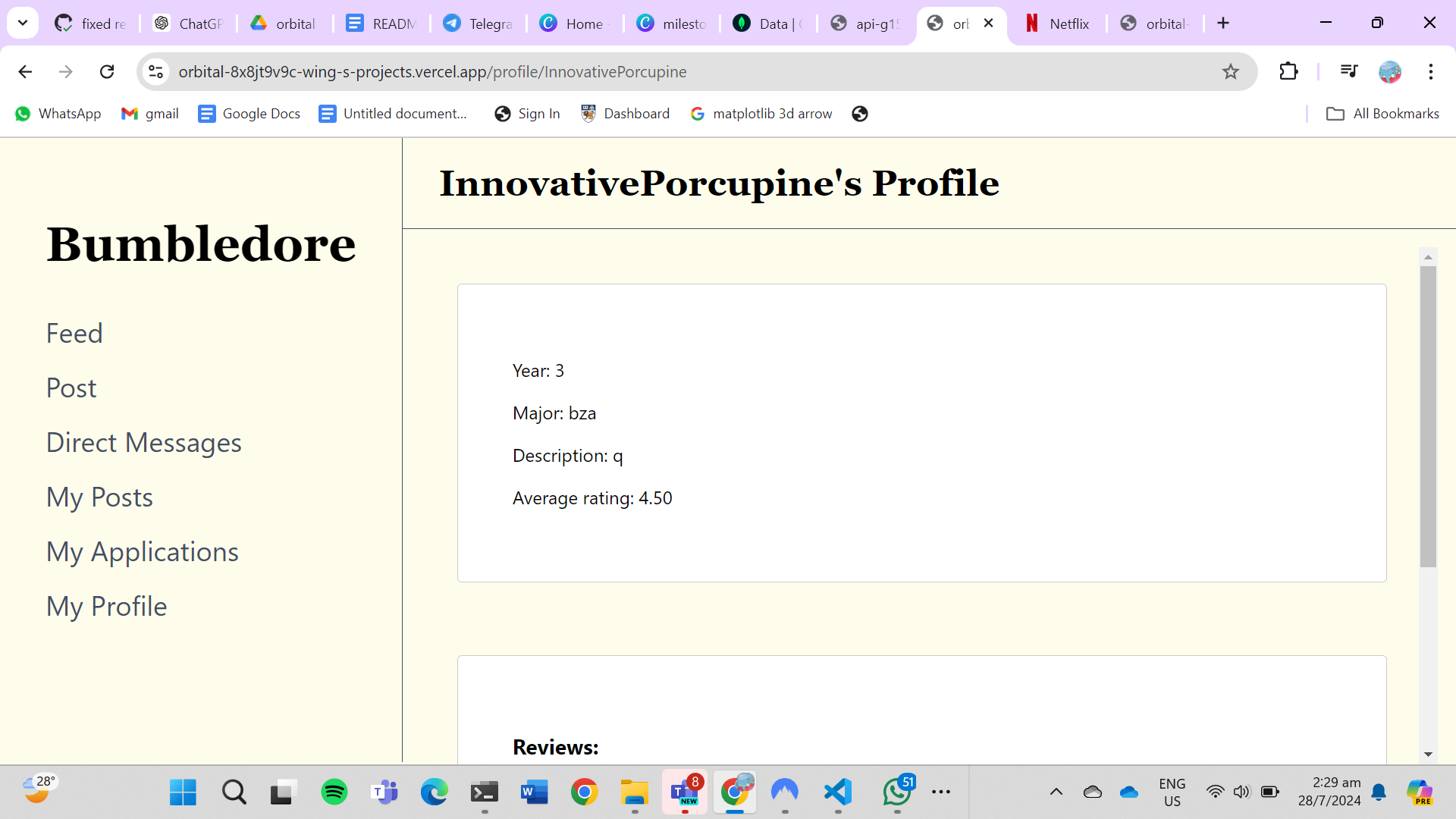


Figure 16: Other user’s Profile page, where you can see their profile description, including their average rating

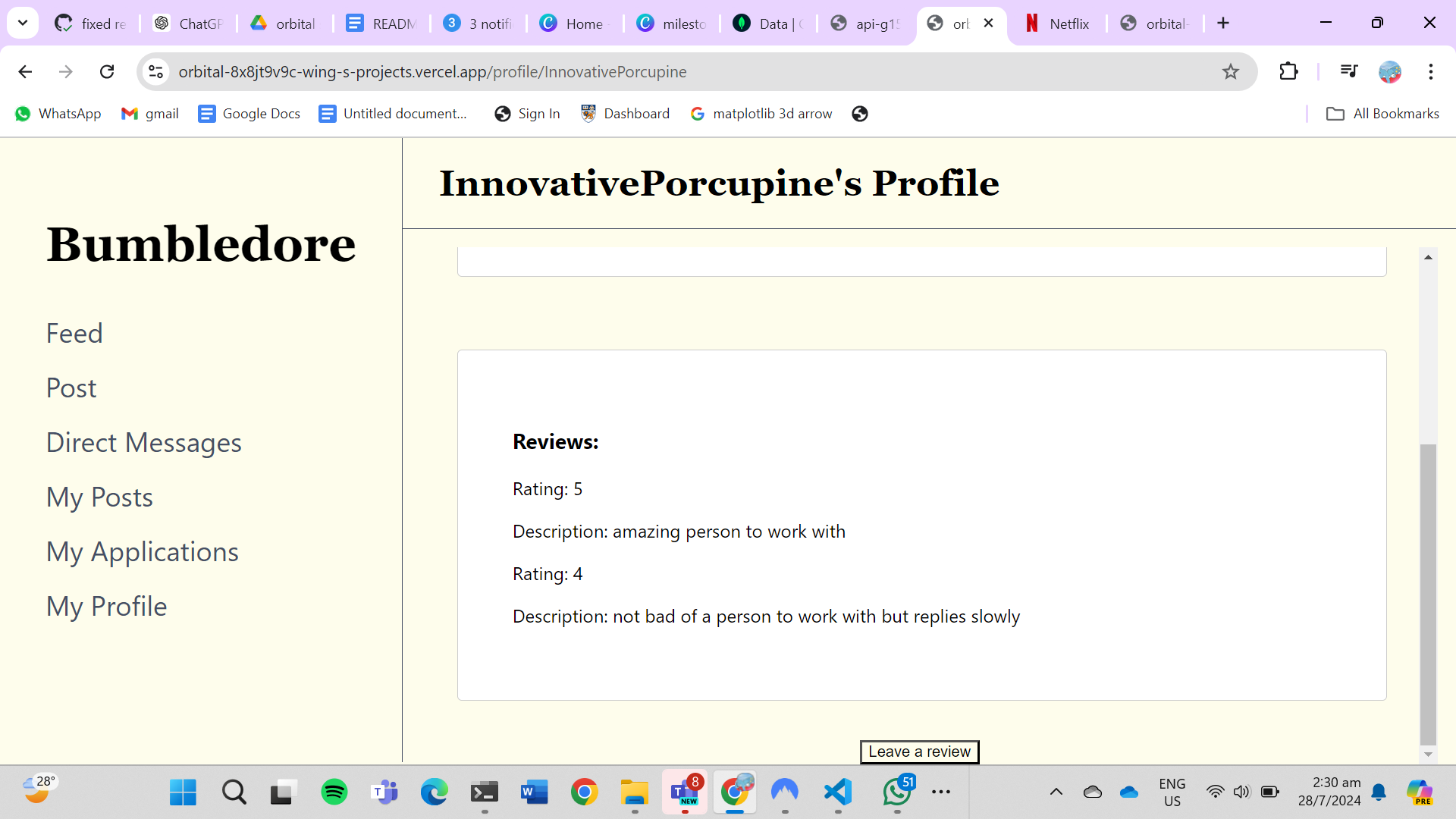


Figure 17: Other user’s Profile page, where you can see the reviews that are left

Milestone 3

Students can leave reviews for each other after working together with them. This incentivises students to be committed to their work and create a better experience for all.

### Extension Feature 9: Insights

Insights are present in the My Posts tab, under each user’s post request, and the My Profile tab.

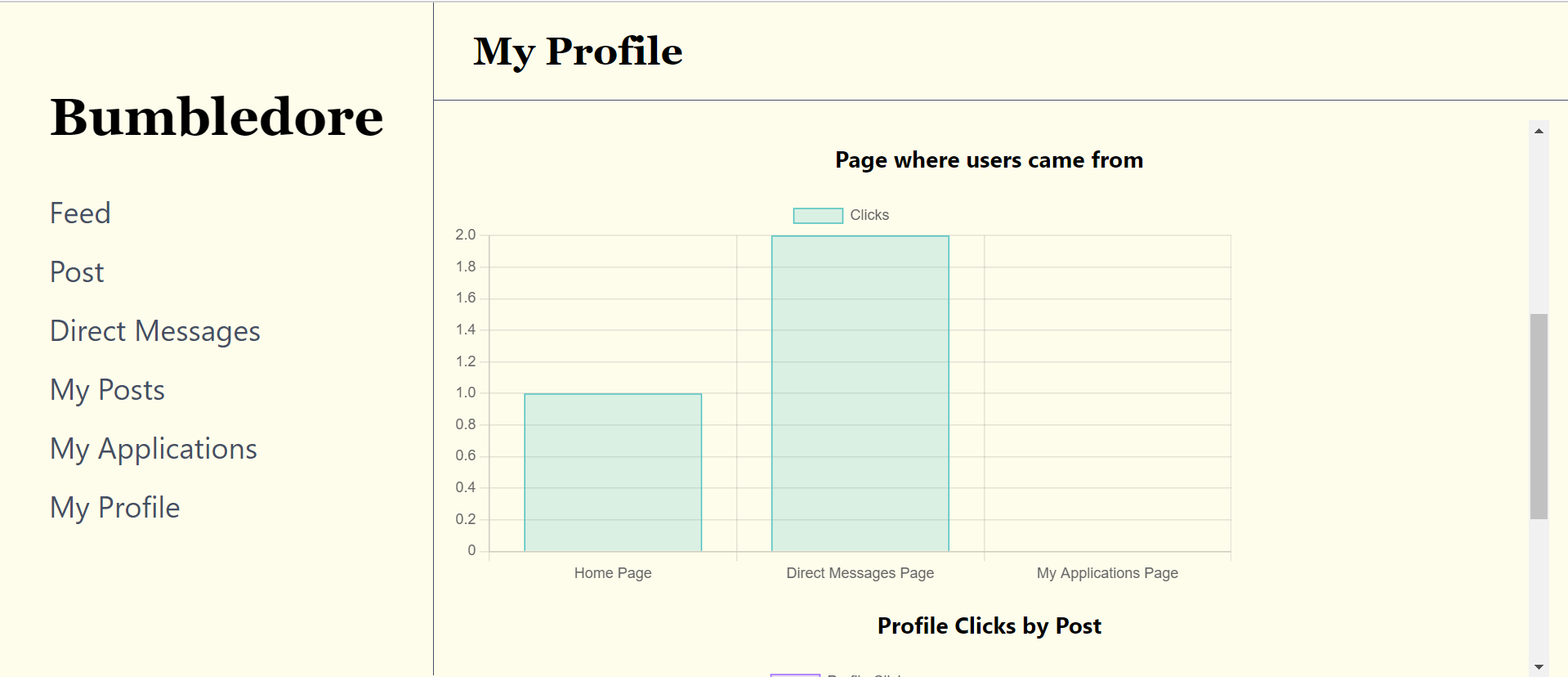
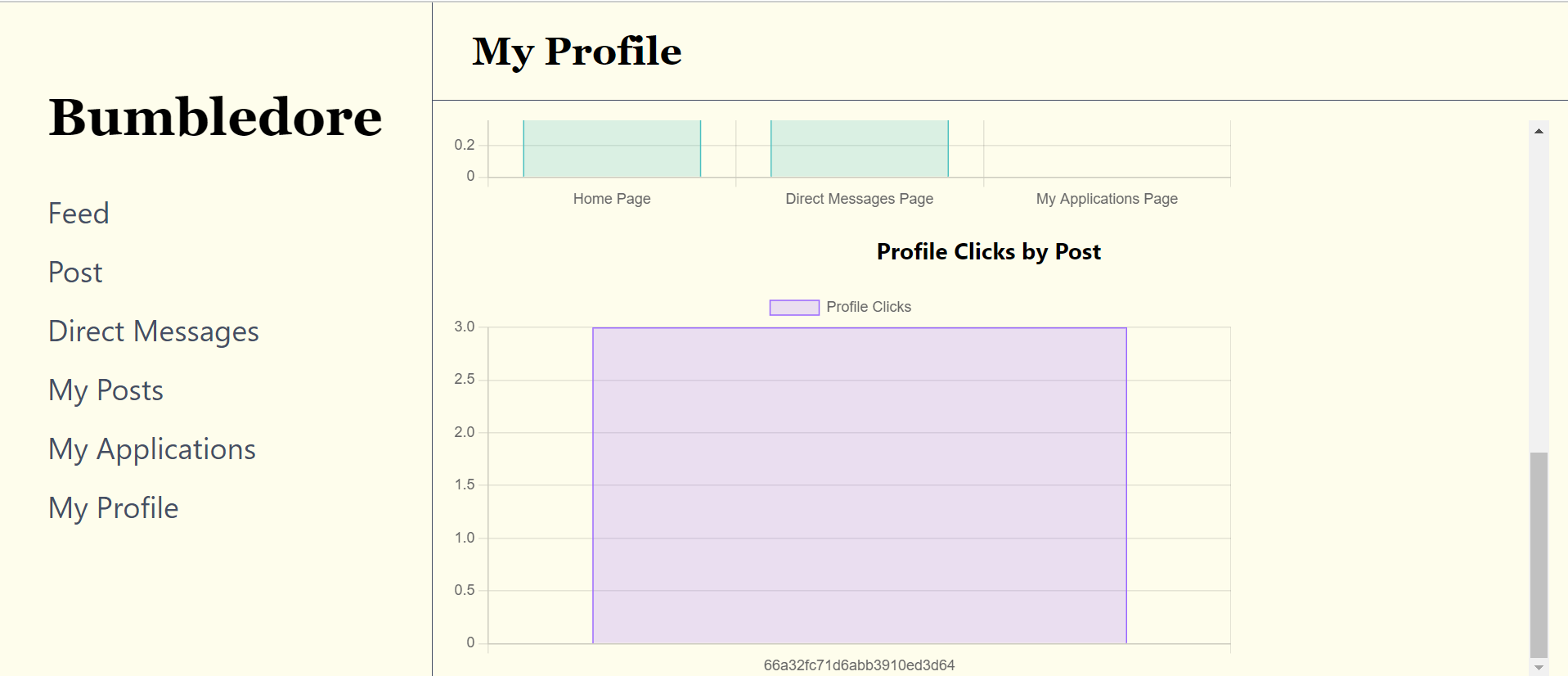


Figure 18: My Profile Page, where you can see the page where users came from

Figure 19: My Profile Page, where you can see profile clicks by post

Milestone 3

In My Posts, there is a ‘Show insights” button on each post, upon clicking this button, users can see a bar chart of the number of clicks for the “Apply” button, “Message” button and profile link for that post. This will allow users to have a better understanding of how their post is doing. If their post is not getting much clicks for all 3 buttons, that means that it is not generating much interest and is not likely to be successful. If the user is getting many profile link clicks but not as many “Apply” button or “Message” button clicks, that could mean that their profile description should be improved on as it is driving away other users from wanting to work with them.

In My Profile, the insights feature displays 2 bar charts. The first bar chart displays the number of profile clicks from different pages, such as the Home page, the Direct Messages page and the My Applications page. This allows users to see where most of the users who view their profile come from. Knowing the source of profile clicks can help users to identify which features or sections of the app they need to improve in or are particularly successful.

The second bar chart displays the number of profile clicks from different posts. This allows users to determine which types of content or posts are driving more profile views. After knowing which of their posts are generating the most interest and interaction, they can try to improve on their future posts to make them more similar to the posts that are driving profile views.

## System Architecture

### Code Base

Here is the link to our github repository:

<https://github.com/kohweien/team-capy-orbital>

### Tech Stack

1. React (Frontend)
2. Express + Node.js (Backend)
3. MongoDB (Database)
4. Vercel (Deployment)
5. Github (Version control)

### App optimisation

In Milestone 2, we received feedback that the app was loading really slowly, despite our database containing only a few entries. Therefore, in milestone 3, we carried out different measures to optimise our app.

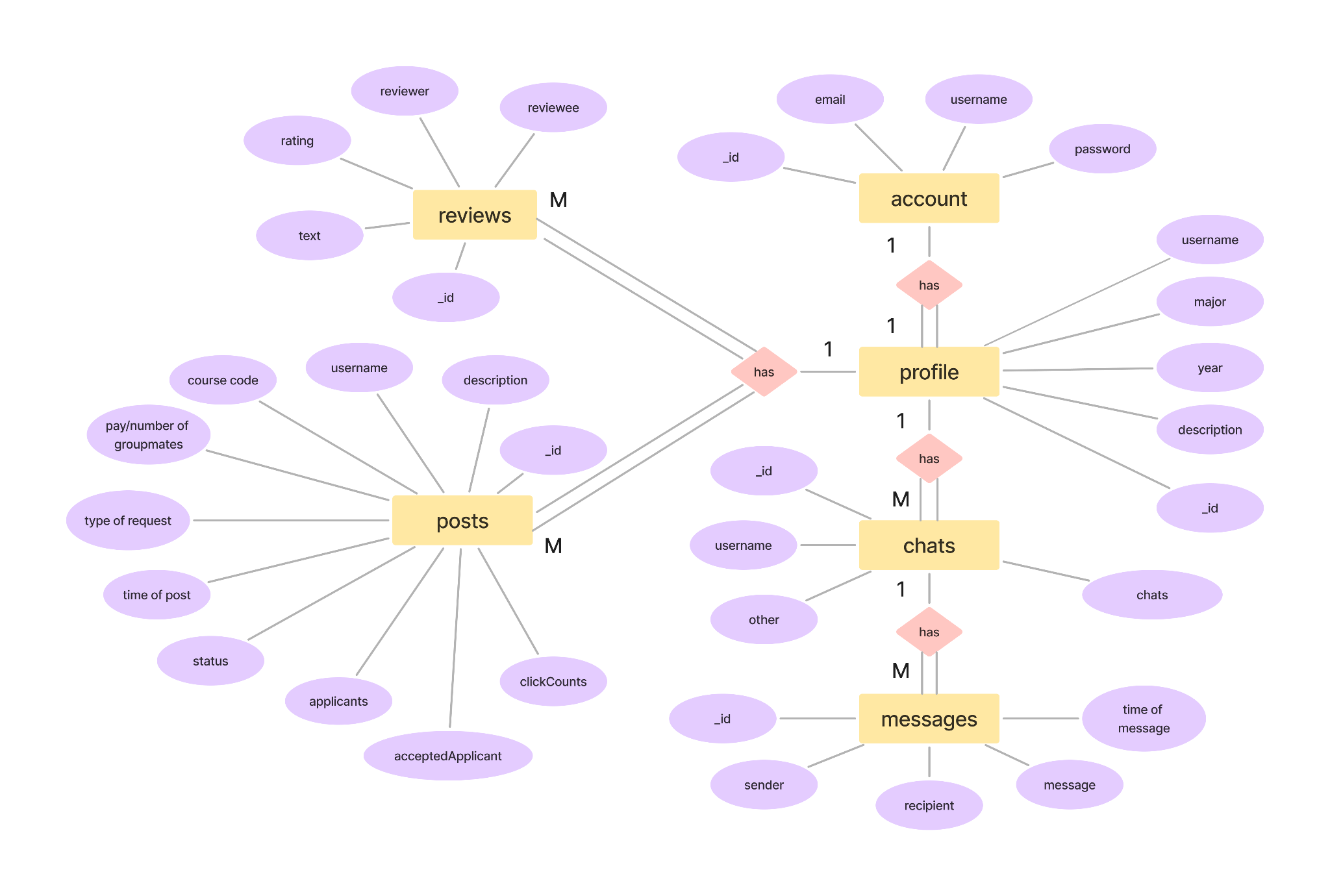
#### Vercel + MongoDB deployment regions

We ensured that the region in which our database was located matched the region the app was deployed in. This is due to the time needed for the database server to communicate with the deployment server. The further the database server is from the deployment server, the greater the response time every time a HTTP request is sent from the frontend deployment to the database. Initially, the default location of the serverless function deployment in Vercel was in Washington DC, USA, while our MongoDB Atlas server was located in Singapore, which utilises AWS web services.. Thus, we changed the default location of the serverless function deployment in Vercel to match the region that our database was located in, which was in Singapore.

#### MongoDB document structuring

Due to the document-based model of MongoDB, instead of normalising all the data and linking collections together using references, we also made use of embedding so as to store related data that needs to be queried together in the same document, reducing the number of read operations which speeds up queries.

To do this, we first created an ER diagram to display all the entities, the attributes they possess and their relationships.

****

We then observed the types of relationships they had, and then considered the size of each item that needed to be stored, the possible number of different items to be stored, and the types of data that needed to be retrieved together.

For instance, for the ‘chats’ collection, each document contained the chat from ‘username’ to ‘other’ and its postID such that every unique chat between two users took up two documents, leading to a lot of repetition. However, as the data needed for each chat was rather small, with each chat only containing the information of the user and the postID, we decided to instead store the data of all the chats a user had with other people in a list of documents, each containing the user and the postID. This reduces the number of read operations needed, especially as all the chat information needs to be loaded together at the same time, optimising the speed of retrieval of chats.

On the other hand, for the ‘reviews’ collection, we initially thought of embedding it under the ‘userProfileInfo’ collection, where each document describing the profile of each user would contain a list of the different documents describing the reviews, including the reviewee, postID, rating and text of the review. However, as each review document stores many different fields, and has the potential to be really large due to the ability of the user to write and submit long paragraphs in their review of other users, we decided to continue storing each review separately in its own document under the ‘reviews’ collection. Storing reviews in the same document as the profile information for each user has the potential to exceed the 16-megabyte document limit size in MongoDB, especially as a potentially limitless number of reviews can be written for each user. Instead, we made use of referencing, using the postID of the post that the users have worked together on as a primary key to link the two collections together.

#### Lazy loading of components

We implemented lazy loading for components that do not show up immediately upon page rendering. This allows only the necessary code that needs to be displayed on the screen to be rendered, speeding up loading rates and improving the user experience.

For instance, for the My Profile page, there is a button which renders a form used to edit the user’s profile when it is pressed. As the form only needs to be rendered upon clicking the button, we decided to use React’s lazy loading feature. Therefore, the code for the form component is only loaded when the ‘Edit Profile’ button is clicked. This lazy loading feature is especially helpful here as the length of code for the profile page is relatively large, with many different aspects such as profile information, reviews and insights needing to be displayed and loaded in the main profile page at the same time. Thus, lazily loading hidden components speeds up rendering time, as less code overall needs to be loaded, with the form component code being loaded only as needed.

### Database use and structure

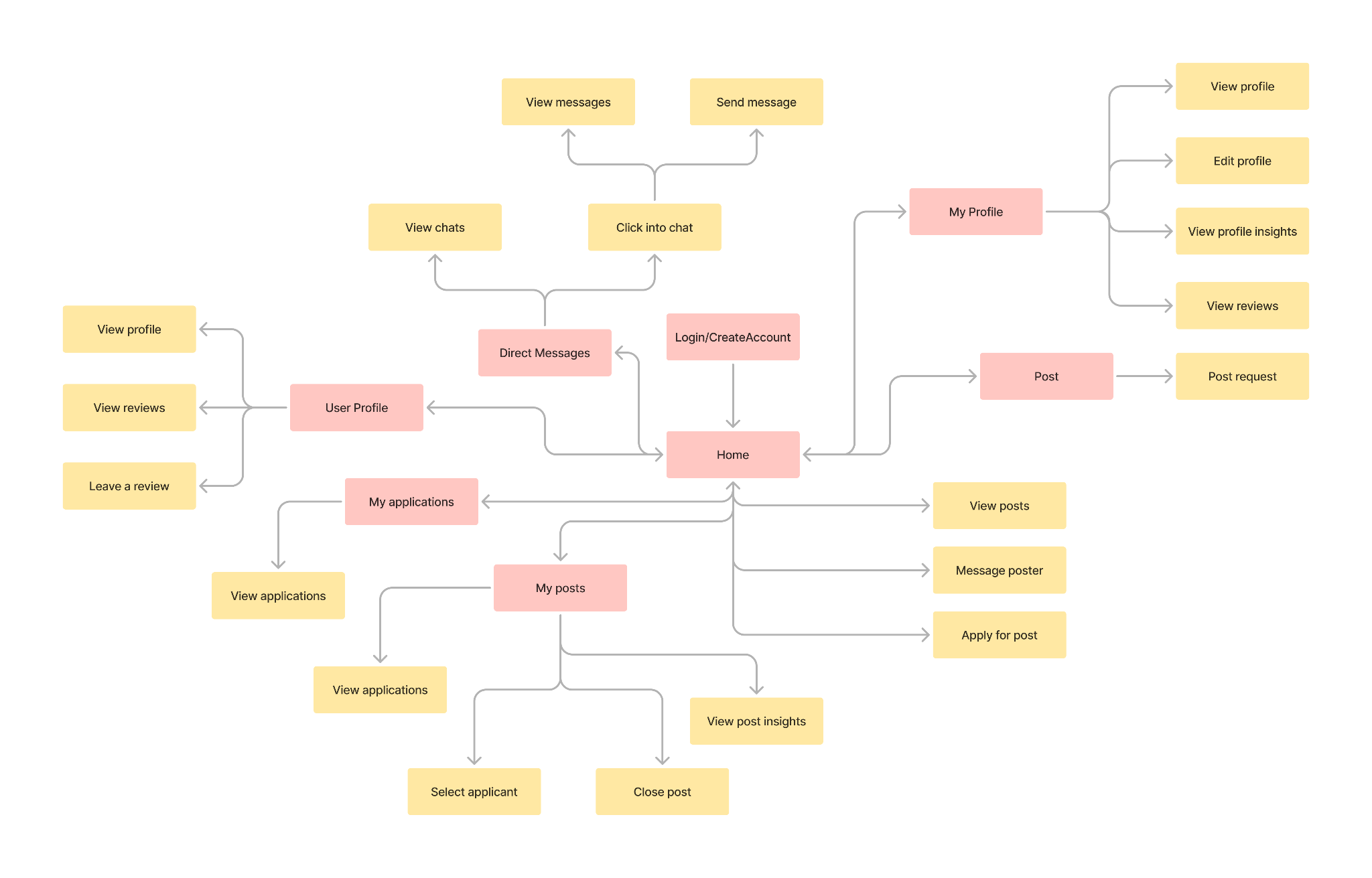
We are using a remote MongoDB Atlas database to store the emails and passwords that users key in when they create an account. We will also use the database to store the profile information about the users, as well as the data included within posts, direct messages, and reviews. We also used Bcrypt to hash passwords that we collect in the create account function before storing them in the database. This enhances security by not storing plain text passwords as Bcrypt is designed to be computationally expensive, which makes it difficult for attackers to use brute force attacks to guess passwords. Bcrypt also automatically handles salting, which involves adding a random string to the password before hashing. This ensures that even if two users have the same password, their hashes will be different. This protects against rainbow table attacks, where precomputed tables of hash values are used to crack passwords.

### Design Decisions

1. We used React in our frontend, and Express + Node.js to link the frontend to the backend, as they are very widely used and have a lot of documentation and user support. It is also relatively user friendly and intuitive to use.
2. We chose MongoDB Atlas as our remote database. We initially wanted to use MySQL as our remote database but we had a lot of unstructured data such as post descriptions, reviews, messages and comments which would be hard to place in a relational database. In addition, MySQL only provides free usage of a remote database for 30 days while MongoDB Atlas did not have such time restrictions. We felt that we would like to expand on this project beyond the orbital period so we did not want to be restricted to only working on this project for 30 days.
3. We chose to use Vercel to deploy our website since Vercel integrates easily with Git repositories, allowing for automated deployments with every push to your repository. In addition, Vercel offers a generous free tier, making it accessible for us. It is also relatively user friendly and intuitive to use.

### User flow diagram

We created a user flow diagram to illustrate the potential activities of someone using the app, including the app’s core functionalities.



## Testing

### System testing

We engaged in rigorous system testing by testing every single functionality in our web app, taking into account the various error scenarios that may occur while performing a task. The description of test cases and the rationale behind the test cases are explained below.

| **Feature tested** | **Description of test case and why we chose these test cases** | **Outcome** |
| --- | --- | --- |
| Create Account | Test creating an account by clicking ‘Create Account’ link on login page, filling in form and submitting; successful account creation will take the user back to the login page.  This is to ensure that users can create an account and login easily. | Pass |
| Create Account | Test creating an account with an email that already exists. This account should not be created.  This is to ensure that users do not have duplicate accounts. | Pass |
| Login | Able to login successfully by keying in valid account details and pressing submit.  This is to ensure that users can login with a valid account. | Pass |
| Login | Test logging in with a valid email but an incorrect password. This user should not be able to login.  This is to prevent malicious actors from accessing our users’ data. | Pass |
| Login | Test logging in with an email that is not registered. This user should not be able to login.  This is to ensure that all our users’ account details are stored in our database. | Pass |
| Feed | Test if able to view posts in feed in the home page upon successful login, and able to scroll to view posts.  This is to ensure that users can view all available posts. | Pass |
| Feed | Test if able to view others’ profiles by clicking the poster's username in post, which redirects to the other person’s profile page.  This is to ensure that users can learn more about different users that they are interested in working with. | Pass |
| Feed | Test if able to create a chat with other posters by clicking on the ‘Message’ button in a post in the feed.  This is to ensure that users can communicate with each other. | Pass |
| Feed | Test if the post was created by the user, the ‘Message’ button will not show up.  This is to prevent users from creating a chat with themselves. | Pass |
| Posting requests | Test creating a post by clicking on ‘Post’ on the sidebar of the home page and filling in a form.  This is to ensure that users are able to make posts. | Pass |
| Direct message | Able to view chats in the sidebar of the home page by clicking ‘Direct Message’ on the sidebar. If the user has no chats, there should be no chats showing, otherwise the user should be able to see the posts IDs and usernames whom they have chats with in the sidebar.  This is so that users can easily access the chats that they have with different users for different posts. | Pass |
| Direct message | Able to view messages by clicking on the username in the sidebar of the messages page. If the users have not sent each other anything yet, no messages should load.  This is so that users can see messages that have been sent to them and messages that they have sent. | Pass |
| Direct message | Able to send messages by clicking into a chat from the sidebar of the messages page, filling in the text bar at the bottom of the page and clicking submit. Upon refreshing the page, the message should appear.  This is so that users can send messages to other users. | Pass |
| Profile | Able to view the user's own profile by clicking on ‘Profile’ on the sidebar of the home page. If no profile has been created yet, the fields should be empty.  This is to ensure that users’ profiles only display information that describes the user. | Pass |
| Profile | Able to update the user’s own profile by clicking on ‘My Profile’ on the sidebar, then clicking on the ‘Edit Profile’ button at the bottom of the white box, then filling in the form and submitting it. If the user does not want to edit their profile, they can click the ‘Back’ button at the bottom of the form.  This is to ensure that users are able to describe themselves for other users to understand them better. | Pass |
| Profile | Able to view profile insights at the bottom of the profile page by clicking on ‘Profile’ at the sidebar of the home page, then scrolling to the bottom of the Profile page.  This is to ensure that users are able to have a better understanding of how other users interact with their profile. | Pass |
| Reviews | Able to click on a user profile, either through the chat they have created with the user, the feed or my applications, and able to leave a review by clicking on ‘Leave a review’ button, filling in the form, and pressing ‘Submit’ button.  This is to ensure that users are able to leave reviews to explain their experience working with this user. | Pass |
| Reviews | Able to click on a user profile, either through the chat they have created with the user, the feed or my applications, and able to view a user’s reviews, if they have any.  This is to ensure that users are able to understand how other users are to work with. | Pass |
| My applications | Able to view posts that the user has applied for, including its status, by clicking on the ‘My Applications’ tab in the sidebar.  This is to ensure that users can keep track of the posts that they have applied for easily. | Pass |
| My applications | If the poster has closed the post or chosen another user as the successful applicant, the status of the post should be “Closed”.  This is to ensure that users know that the post is no longer available for them. | Pass |
| My applications | If the poster has accepted the user as the successful applicant, the status of the post should be “Successful”.  This is to ensure that users know that they are the accepted applicant. | Pass |
| My posts | Able to view posts that the user has created, including its status, ‘Close Post’ button, and ‘View Insights’ button by clicking on the ‘My Posts’ tab in the sidebar. Additionally, if there have been at least one applicant, the dropdown select applicant option and ‘Confirm’ button should both be visible as well.  This is to ensure that users can manage their posts easily. | Pass |
| My Posts | Able to close a post by clicking on the “Close post” button, and the status of the post should be “closed”.  Post should also not show up on the feed anymore.  This is to ensure that users do not see closed posts on their feed and try to apply for it. It is also to ensure that users are clear on whether their closing the post has been recorded down by our database. | Pass |
| My posts | Able to select applicants by clicking on ‘My Posts’ tab in the sidebar, clicking on ‘Select an applicant’ dropdown options and selecting the desired option, and clicking ‘Confirm’ to confirm selection, changing the status from ‘Open’ to ‘Successful’. Post should also be removed from Home feed.  This is to ensure that users are able to choose their applicants and be clear on whether their choice has been recorded down by our database. This is also to ensure that other users will not see closed posts on their feed and try to apply for it. | Pass |
| Insights | Able to view individual posts insights by clicking on ‘My Posts’ tab in the sidebar and clicking on ‘Show Insights’ to view the bar chart of number of message clicks, apply clicks and profile clicks for each post.  This is to ensure that users can access the insights of their posts easily. | Pass |
| Insights | Able to view profile insights by clicking on ‘My Profile’ on the sidebar, then scrolling to the bottom of the page to see two bar graphs. The first one shows the pages that each user has come from, for example the messages page, feed page or my applications page, and the second one shows the number of profile clicks by each post the user has made.  This is to ensure that users can access the insights of their profile easily. | Pass |

### Unit testing

We made use of Jest to write our automated test cases, which can be run with `npm test` under the ‘bumbledore’ folder in powershell. Our test cases mainly focused on the user interactions with our application and whether the components render correctly.

| **File** | **Description of test case** | **Outcome** |
| --- | --- | --- |
| Login.test.js | Page loads correctly: renders Login component. | Pass |
| Login.test.js | Page loads correctly: form able to be filled in with input values updated. | Pass |
| Login.test.js | Correct form input and submission: after form filled in with valid email and password, clicking on 'Login' button navigates user to home page | Pass |
| CreateAccount.test.js | Page loads correctly: form able to be filled in with input values updated | Pass |
| CreateAccount.test.js | Correct form input and submission: after form filled in with valid email and password, clicking on ‘Create Account’ button navigates user to Login page | Pass |
| Home.test.js | Page loads correctly: shows loader while fetching data, then renders the Home component and posts | Pass |
| Post.test.js | Page loads correctly: renders Post component | Pass |
| Messages.test.js | Page loads correctly: renders Messages component | Pass |
| Profile.test.js | Page loads correctly: renders Profile component | Pass |
| UserProfile.test.js | Page loads correctly: renders User Profile component | Pass |
| MyApplications.test.js | Page loads correctly: shows loader while fetching data, then renders the My Applications component and posts | Pass |
| MyPosts.test.js | Page loads correctly: shows loader while fetching data, then renders the My Posts component and posts | Pass |

### User testing

| **Action taken** | **Percentage of users that passed (%)** | **Feedback** |
| --- | --- | --- |
| Users are able to create an account by clicking the ‘Create Account’ link on the login page, filling in the form and submitting; successful account creation will take the user back to the login page. | 100 |  |
| Users are able to login successfully by keying in account details and pressing submit. | 100 |  |
| Users are able to see their randomly generated username with their empty profile information upon clicking ‘My Profile’. | 100 | The main interface of the page loads faster than the data; the content box does not extend towards the full page |
| Users are able to edit their profile information. | 100 |  |
| Users should be able to see a bar graph of the number of profile clicks from different pages, such as the Home page, Messages page and the My Applications page in the My Profile page. | 100 | Graph is not centred in page in My Profile |
| Users should be able to see a bar graph of the number of profile clicks for each post. | 100 |  |
| Users are able to see posts upon clicking ‘Feed’ | 100 |  |
| Users are able to see the profile of another user upon clicking the username of a post in the feed. | 100 |  |
| Users are able to create a new chat with another user for that post upon clicking ‘Message’ under a post in the feed. | 100 |  |
| Users are able to send messages to another user upon clicking the on the other user’s username in the sidebar and clicking the submit button after typing in a message. | 100 | The chats sidebar containing the chats is not well aligned |
| Users are able to apply for a post by clicking on the “Apply” button. A chat should be created with the poster for that post and the message “Hi! I have applied for your post!” should be automatically sent to that user for that post. | 100 |  |
| Users are able to create a post upon clicking ‘Post’ and keying in details before clicking on the submit button. | 100 |  |
| Users are able to see their post upon clicking ‘Feed’. There should not be a message button below their post. | 100 |  |
| They should be able to click on the “Show Insights” button in My Posts to see a graph of the number of “Message” button clicks, “Apply” button clicks and username clicks for that post. | 100 |  |
| Users should be able to click on the “Select Applicant” bar and select an applicant, upon hitting the “Confirm” button, the page should reload and the status of the post should be “Successful” and the username of the selected applicant should be displayed. | 100 |  |
| Users should be able to click on the username of someone that they have successfully matched with and click on the “Make review” button and see a list of posts that they have successfully matched with that user for. Users should then be able to fill in the review form and submit it, and see the review appear on the other user’s profile. | 100 |  |

## Appendix

### Poster

<https://drive.google.com/file/d/1-7bEbEuIWr96mD5Fos4TCDoet7Cq1i3V/view?usp=sharing>

### Project log

<https://docs.google.com/document/d/17QlRCocIW6WinnlCLFqv5xAQTqf2udcDWCXmChnyIPE/edit>

### Project video

https://youtu.be/qvkVezhxNDU