CachingFeelings Release Summary

Team members

Name and email	GitHub id	Role of each member and tasks done by each member
Breanna Brown millettb@myumanitoba.ca	bre9425	Team lead, Frontend, Backend, Load testing
Oluwademilade Akinsola akinsol2@myumanitoba.ca	Demi-AK	Testing
Heejeong Kim kimh4@myumanitoba.ca	Heejoy	Documentation, Testing, CI/CD
Rahul Kumar kumarr7@myumanitoba.ca	bochacho	Backend, Database, Testing
Dingyuan Zhang zhangd7@myumanitoba.ca	AIINEWAREAS	Frontend, Backend, CI/CD, Security scan

Project summary

Caching Feelings is an app designed specifically with computer science enthusiasts in mind. We wanted to create an environment where tech enthusiasts can meet and develop meaningful relationships that will hopefully go beyond the digital realm. Users will be matched specifically with people who share the same coding quirks and tech passions, creating connections that go beyond other superficial dating apps that only focus on appearance. Users can then dive into meaningful conversations, spark real connections, and hopefully find their perfect coding partner in crime.

GitHub repository link

https://github.com/cachingFeelings/cachingFeelings

Docker Image Client Link:

https://hub.docker.com/layers/allnewareas/caching-feelings-client/latest/images/sha256:fe08b7c21968fa42babd732bbc85feb1fbdf3951f19fe86b08f76093b9a830cb?uuid=a26fb4cd-6107-450e-9730-64ecd68dfe2c%0A

Docker Image Server Link:

https://hub.docker.com/layers/allnewareas/caching-feelings-server/latest/images/sha256:1b3cc15b1d25a29449e96a204a62e5df56d933692b014b156555c7dd0fbbac28?uuid=a26fb4cd-6107-450e-9730-64ecd68dfe2c%0A

Run Client Docker Image (Copy and Paste):

```
docker run -p 3000:3000 -e REACT_APP_SERVER_URL='http://localhost' -e REACT_APP_SERVER_PORT=8080 allnewareas/caching-feelings-client:latest
```

Run Server Docker Image (Copy and Paste):

```
docker run -p 8080:8080 -e MONGO_URI='mongodb+srv://bochacho:TZq79ZTXcWRjRrvu@m0.vzljdxy.mongodb.net/?retryWrites=true&w=majority' -e PORT=8080 -e JWT_SECRET='ImBATMAN' -e AWS_ACCESS_KEY_ID='AKIA4MTWKSLGE27FUEAJ' -e AWS_SECRET_ACCESS_KEY='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' -e AWS_REGION='us-east-2' -e AWS_S3_BUCKET='cachingfeelings' allnewareas/caching-feelings-server:latest
```

Docker Images Note:

After copying and pasting those commands on each terminal and running them, please open browser and type http://localhost:3000, our app will run on this link by using command line arguments running docker images. We also deployed our app here https://caching-feelings.vercel.app feel free to use it!

List of user stories for each sprint

Sprint1

N/A

Sprint 2

#1. [Done] As a user, I want to sign up so I can have my own account.

#2. [Done] As a user, I want to be able to make changes to and manage my account.

#3. [Done] As a user, I want to be able to share my interests and dating preferences.

#4. [Done] As a user, I want to be able to access the account that I created.

Sprint 3

#1. [Done] As a user, I want to be able to add and manage the photos on my profile.

#2. [Done] As a user, I want to be able to send images and other media to make conversations more fun.

#3. [Done] As a user, I want to be able to block matches if I no longer wish to communicate with them.

#4. [Done] As a user, I want to be able to view all the messages I have with different users so that I can continue to chat with them.

#5. [Done] As a user, I want to be able to send and receive messages with my matches so we can get to know each other more.

#6. [Done] As a user, I want to be able to send a message that is no longer available once the recipient

reads it.

#7. [Done] As a user, I want to be able to view discussions that I am interested in to find others that are interested in the same things as me.

#8. [Done] As a user, I want to be able to view my favourite matches so I can easily locate them at a later time.

#9. [Done] As a user, I want to be able to find other users that match my interests and preferences so I can find others who are like me.

#10. [Done] As a user, I want to be able to view all the users that I have matched with so I can select which ones I want to pursue.

Sprint4

#1. [Done] As a user, I want to send a random message to the person I am currently messaging, so that I can start a smoother conversation.

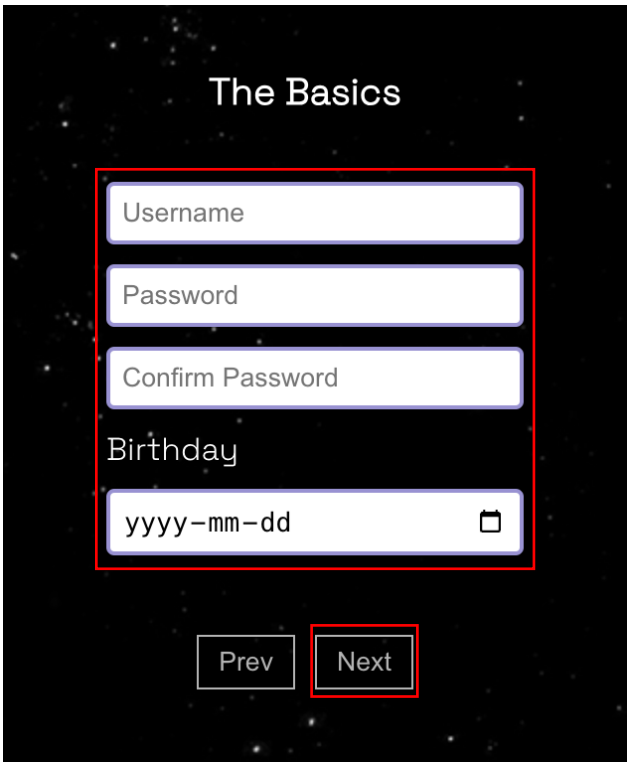
User manual

[Account Creation]

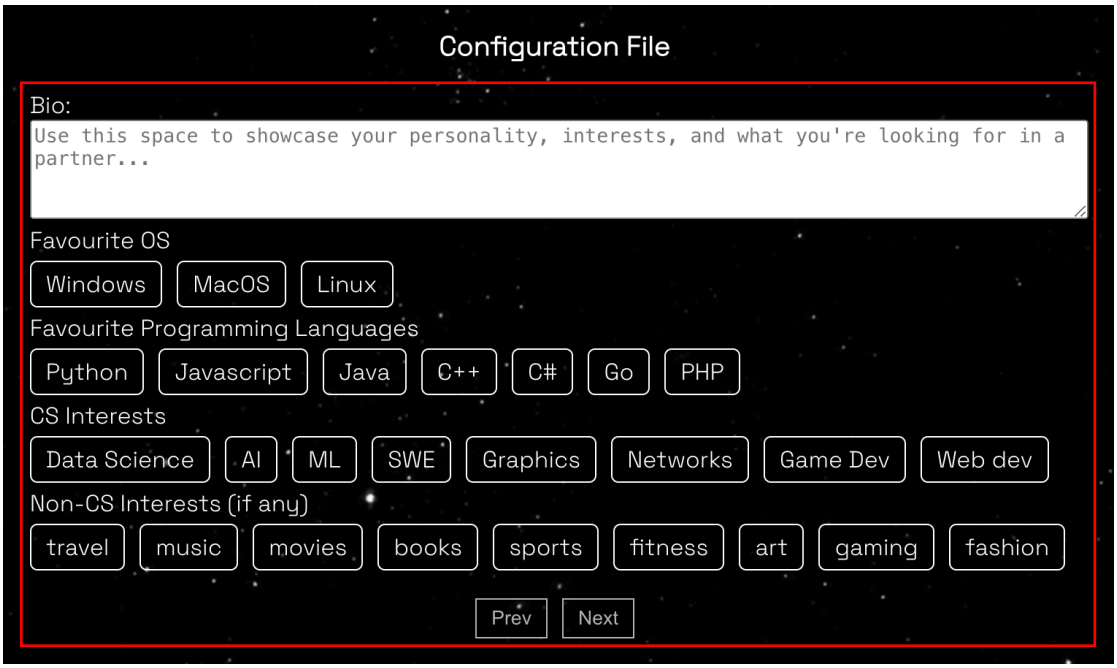
Sign Up:

Step 1. Click Sign Up button to start creating an account.

Step 2. Fill out a valid username, password, confirm password, and birthday then click "Next" button.

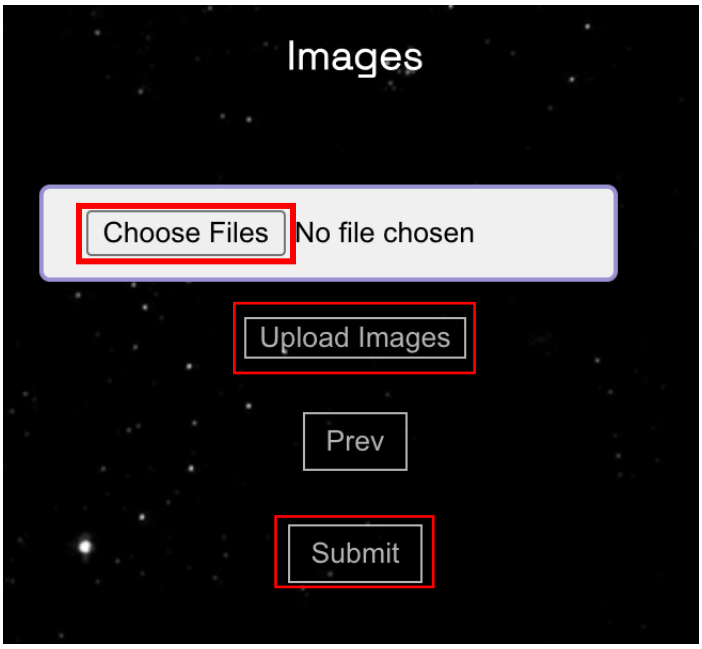


Step 3. Enter your bio in the designated bio text box, choose as many interests as you wish by clicking on the corresponding buttons, and then proceed by clicking the “Next” button.



Step 4. Select the “Choose files” button, choose the profile picture you wish to upload, and then click

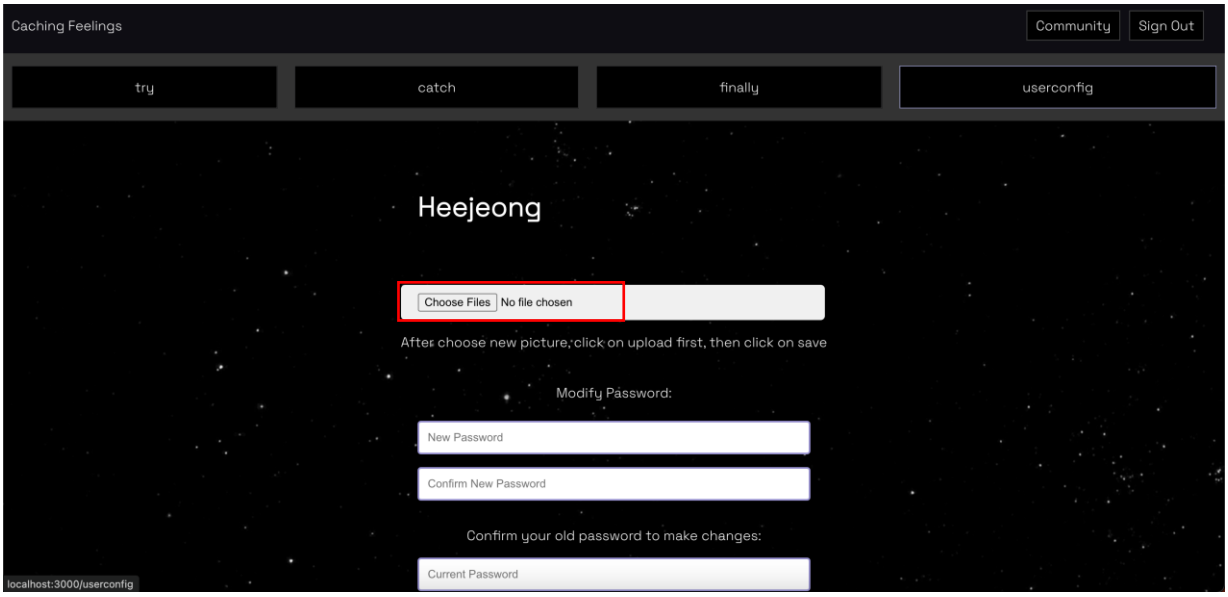
“Upload images.” After that, proceed by clicking the “Submit” button. This will create your account and navigate you to the try page.



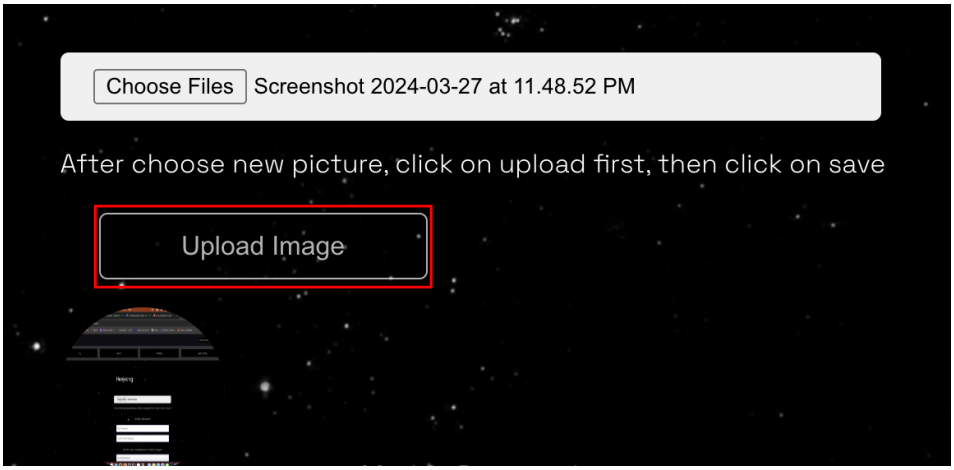
Change Profile Image:

Step 1. Click the “userconfig” button on the top right. It will navigate you to the settings page where you can change the profile image.

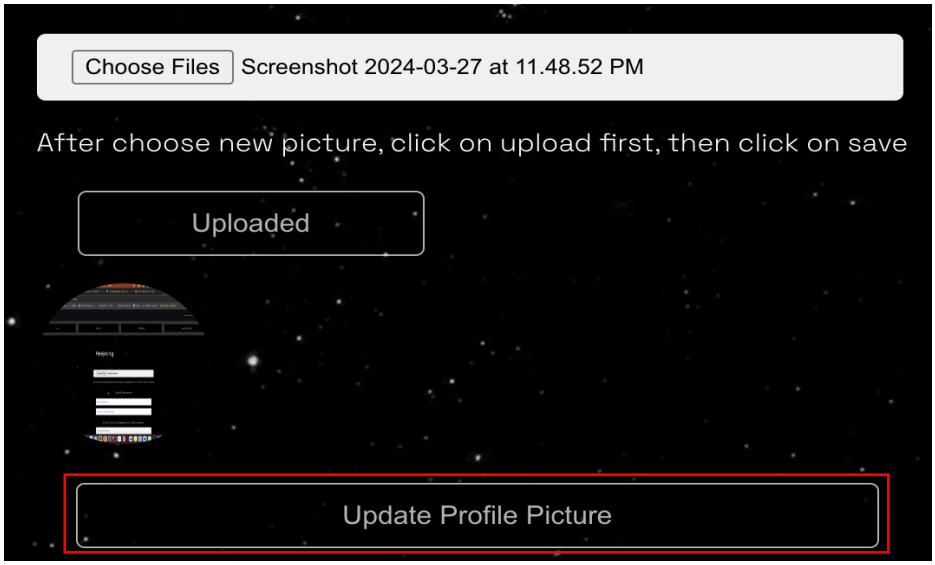
Step 2. Click the “Choose files” button and select an image that you want to change or upload.



Step 3. Click “Upload Image” to upload an image that you choose.



Step 4. Click the “Update Profile Picture” button to save a new profile image. This will update your profile picture.

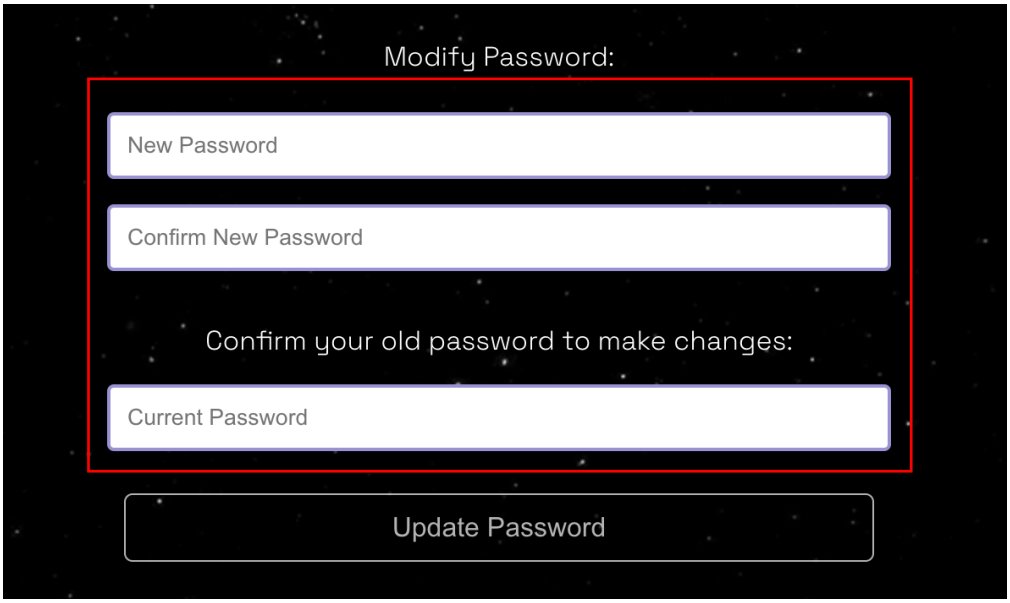


Change Password:

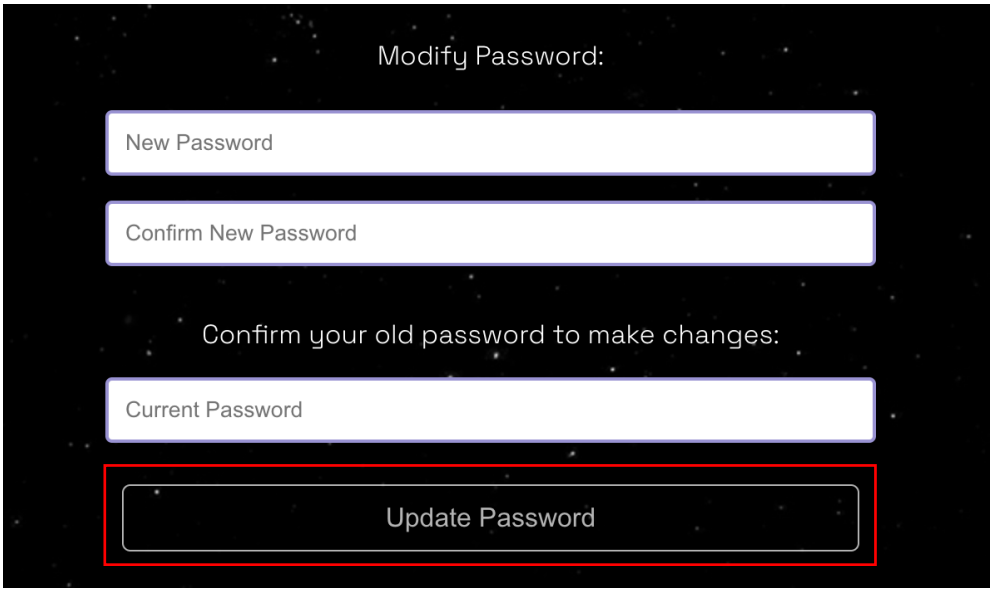
Step 1. Navigate to the “userconfig” page and locate the Modify Password section.

Step 2. Enter new password, confirm it by entering it again, and provide your current password for

verification.



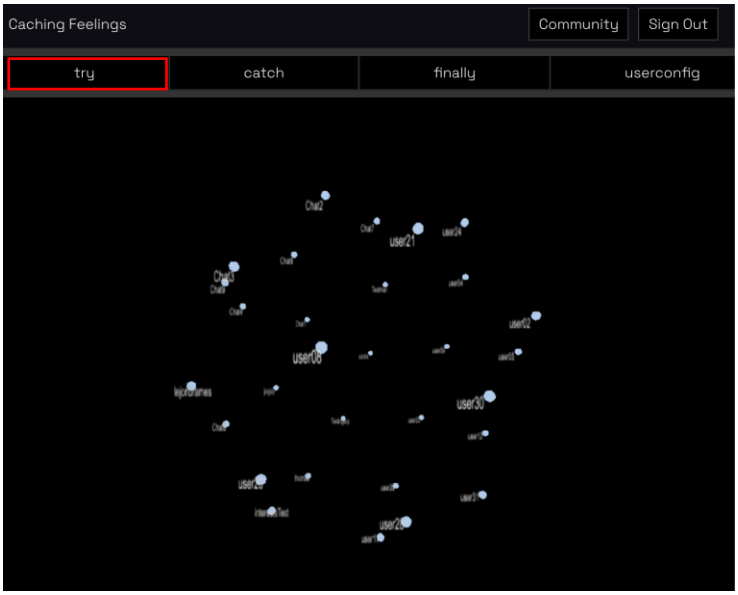
Step 3. Click the “Update Password” button. This action will replace your existing password with the new one.



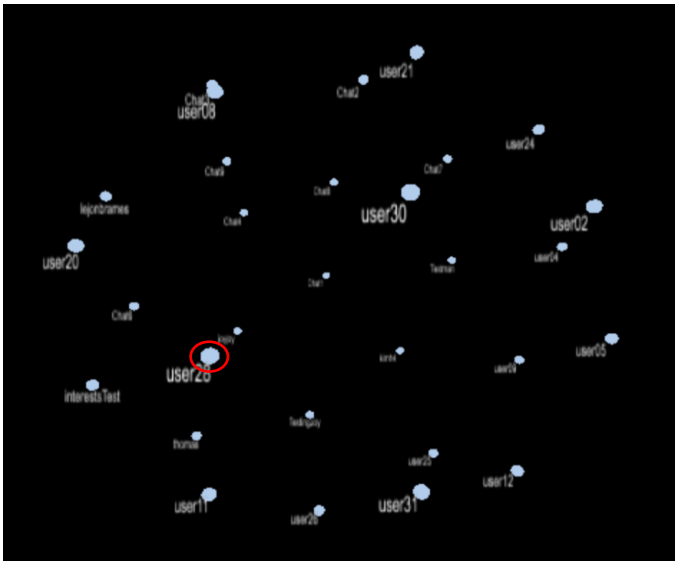
[Student(soul) Matching]

Find people and Like Users:

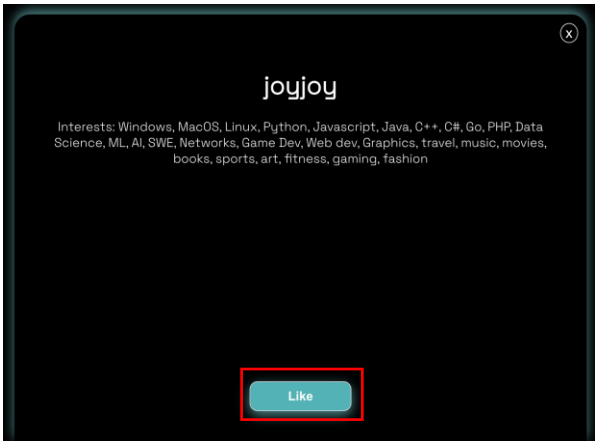
Step 1. Visit the try page, where you will find a sphere for discovering users.



Step 2. Click on a user's node to view their information.



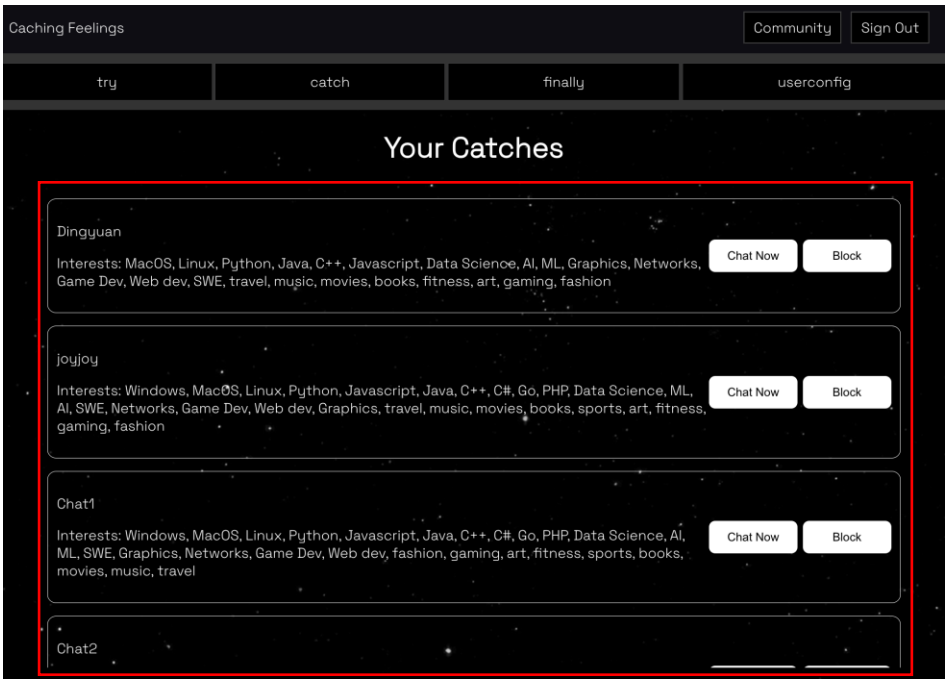
Step 3. If you are interested, you can click the "Like" button to like them.



View Matches:

Step 1. Navigate to the catch page.

Step 2. You will see a list of your matches. Note that users appear in this list only if there's a mutual liking between both parties.

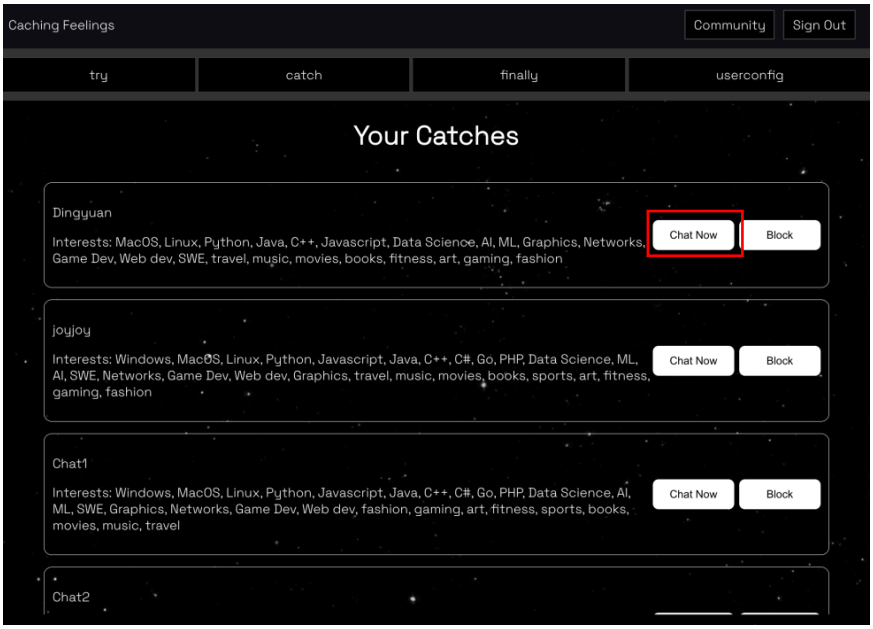


[Interact with Matches]

Start Chat:

Step 1. Navigate to the “catch” page.

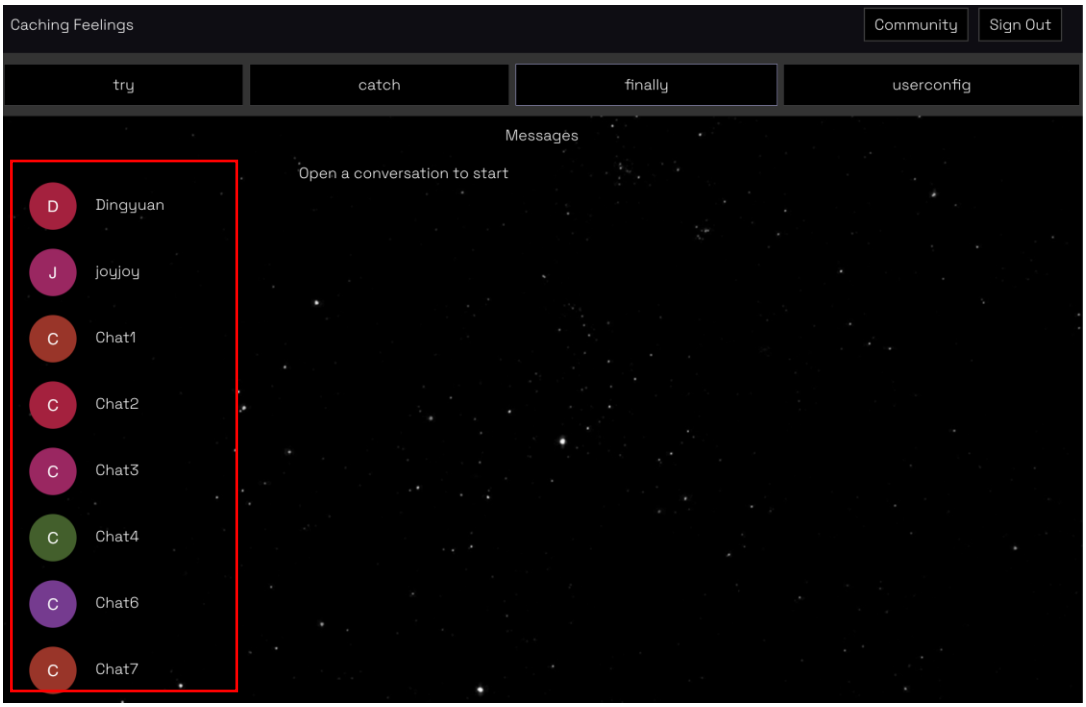
Step 2. Within the list of matches, click the “Chat Now” button. This will navigate you to the finally page, where you can begin exchanging messages with the user.



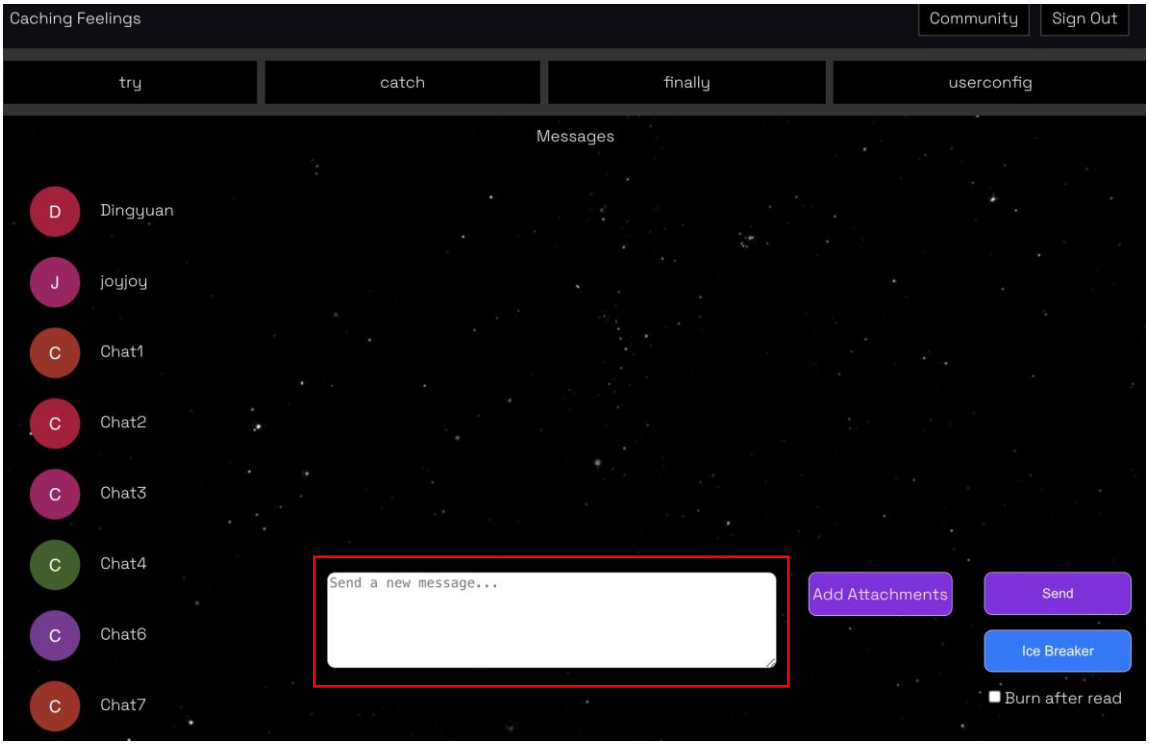
Send and Receive Messages:

Step 1. Go to the “finally” page.

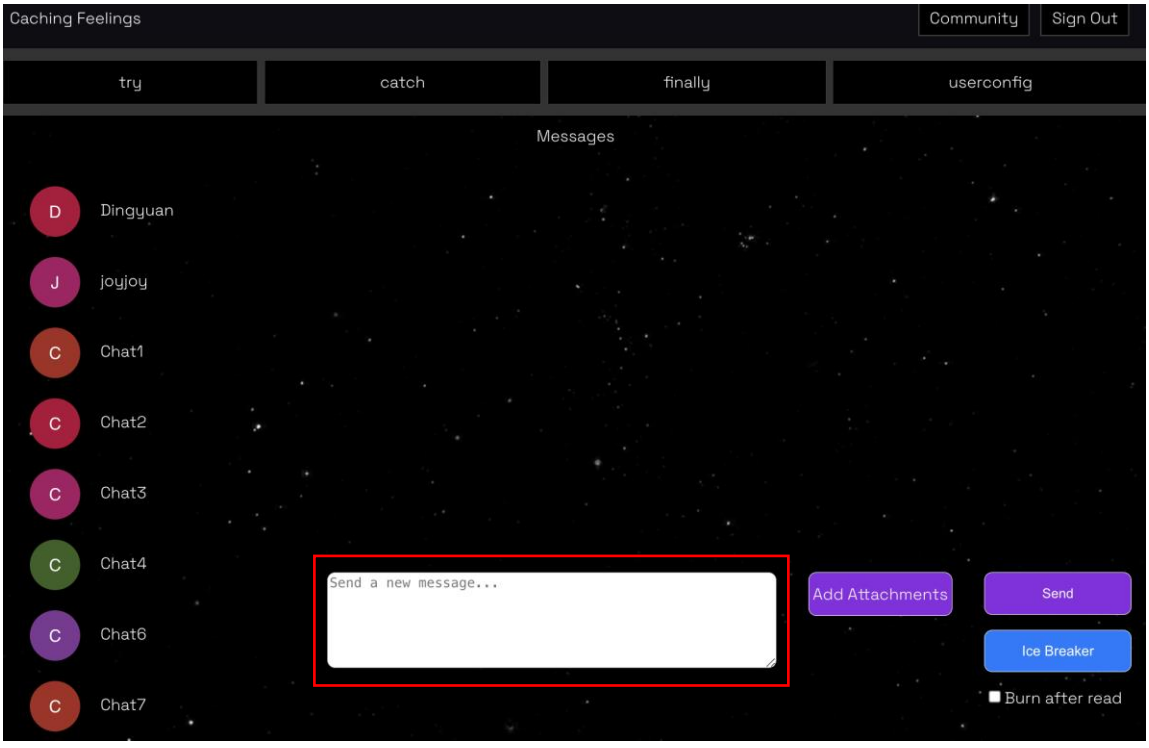
Step 2. Choose a user from the list of chat users on the left.



Step 3. Enter your message in the white text box.

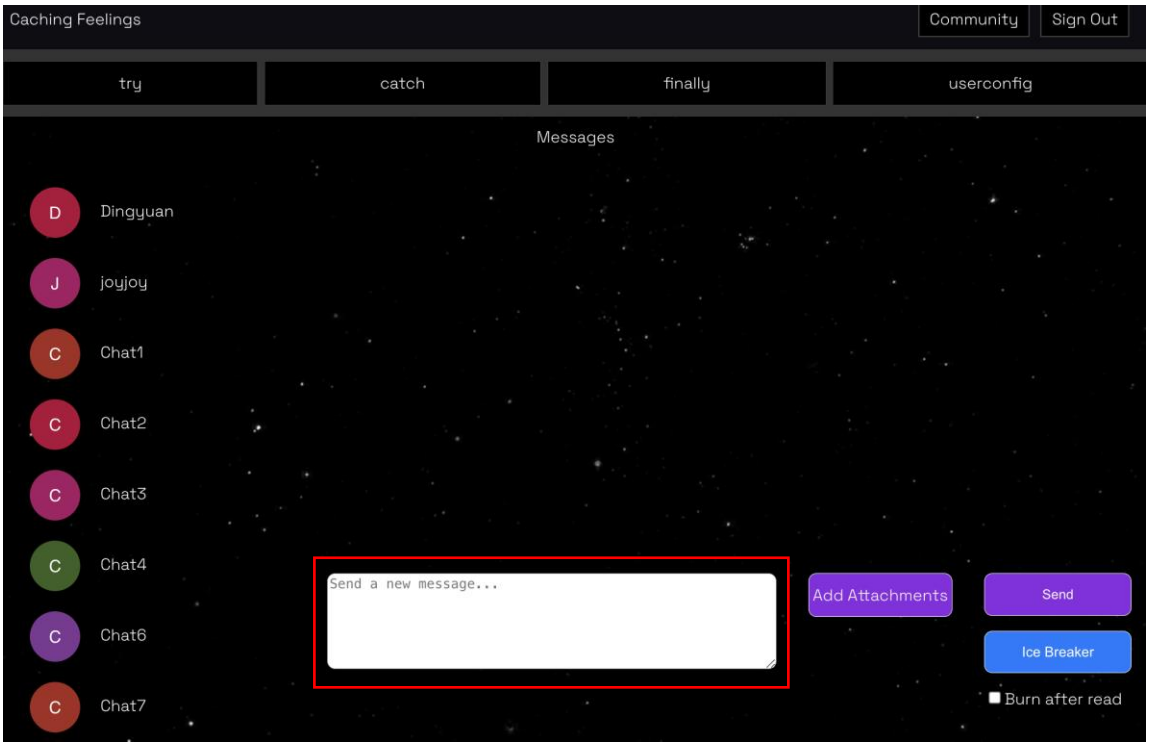


Step 4. Click the “Send” button to dispatch your message.

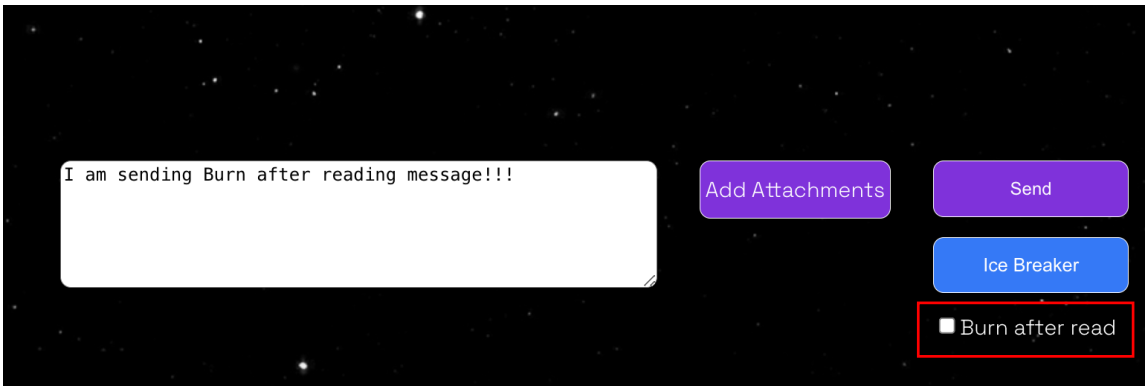


Burn After Read:

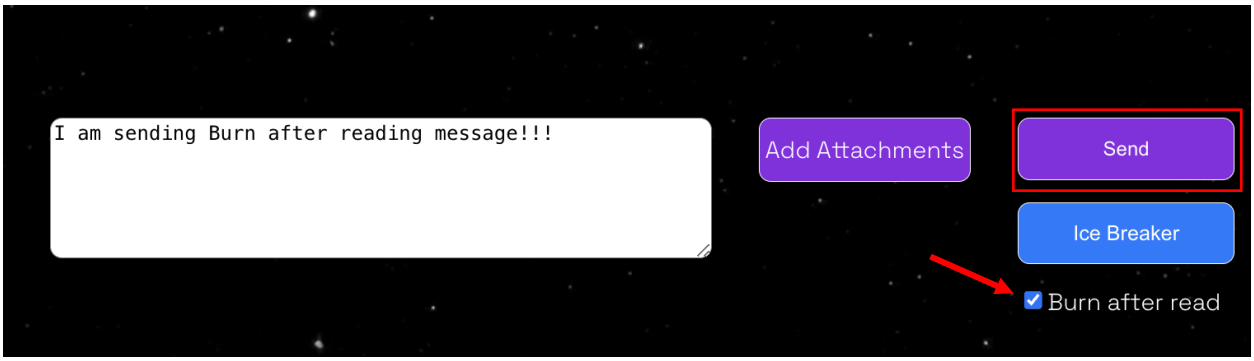
Step 1. Type your desired message in the chat area.



Step 2. Select the “Burn after read” checkbox.



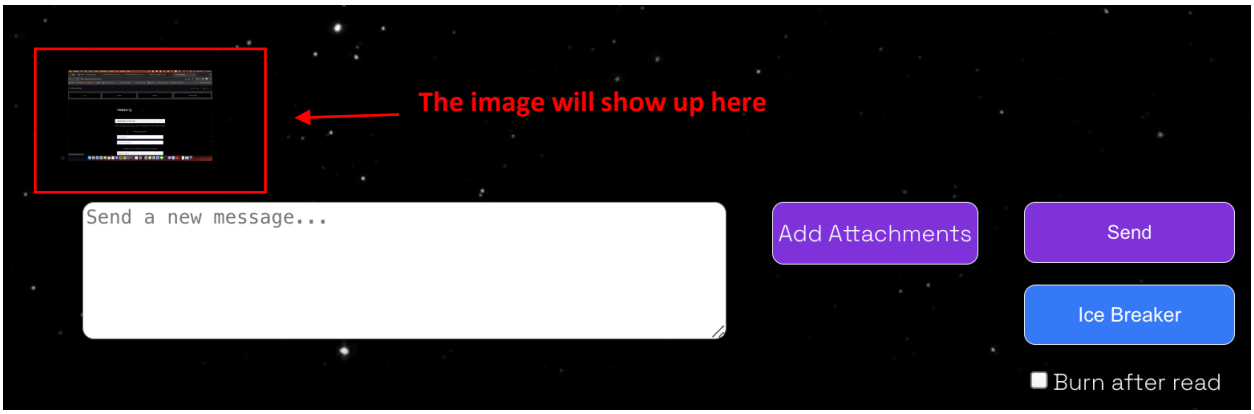
Step 3. Click the “Send” button to dispatch your message with the burn after reading option.



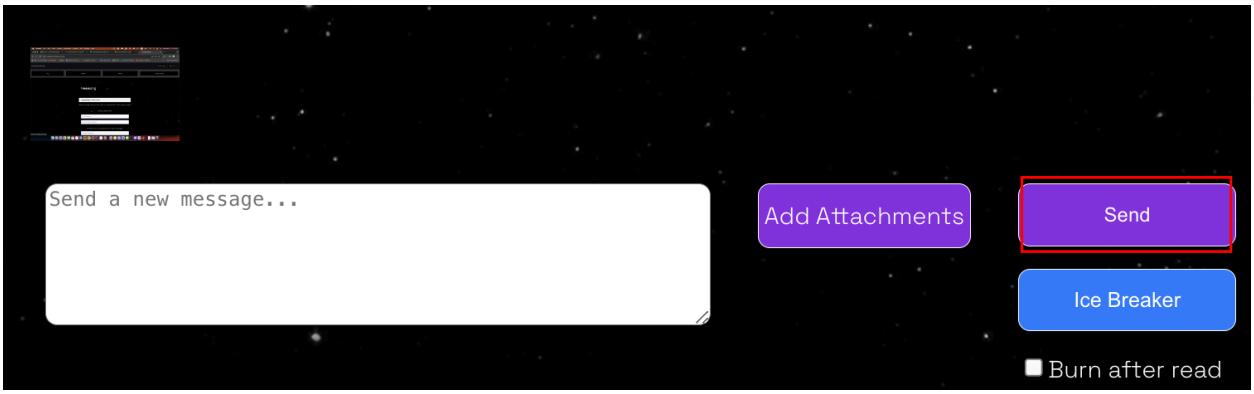
Send Image in Messages:

Step 1. Click “Add Attachments” button.

Step 2. Select the image that you want to send and open it.



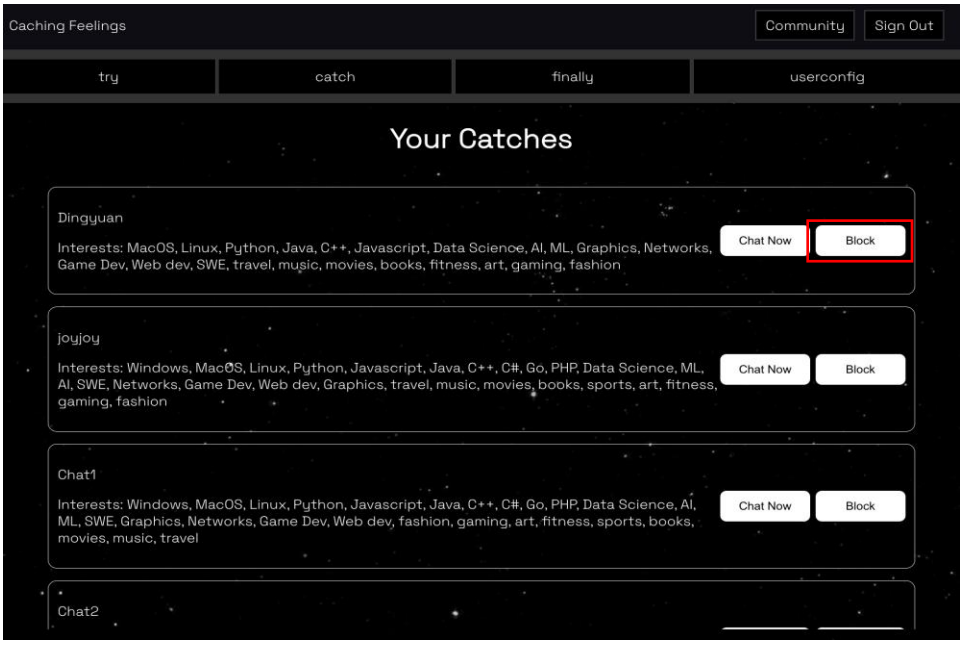
Step 3. Click the “Send” button to send the image.



Blocking Matches:

Step 1. GO to the “catch” page.

Step 2. Within the list of matches, click the “Block” button. This will remove that user from the matches.

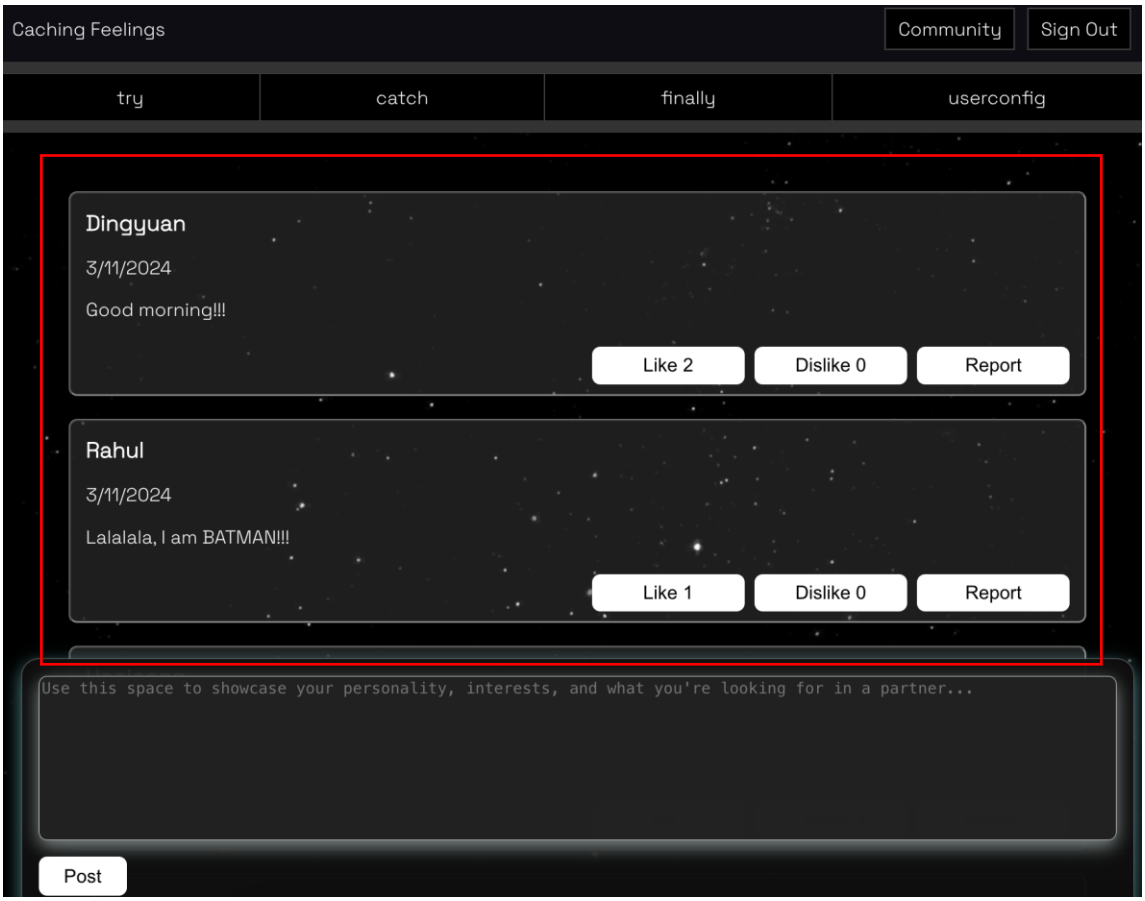


[Community Discovery]

View Posts in Community Page:

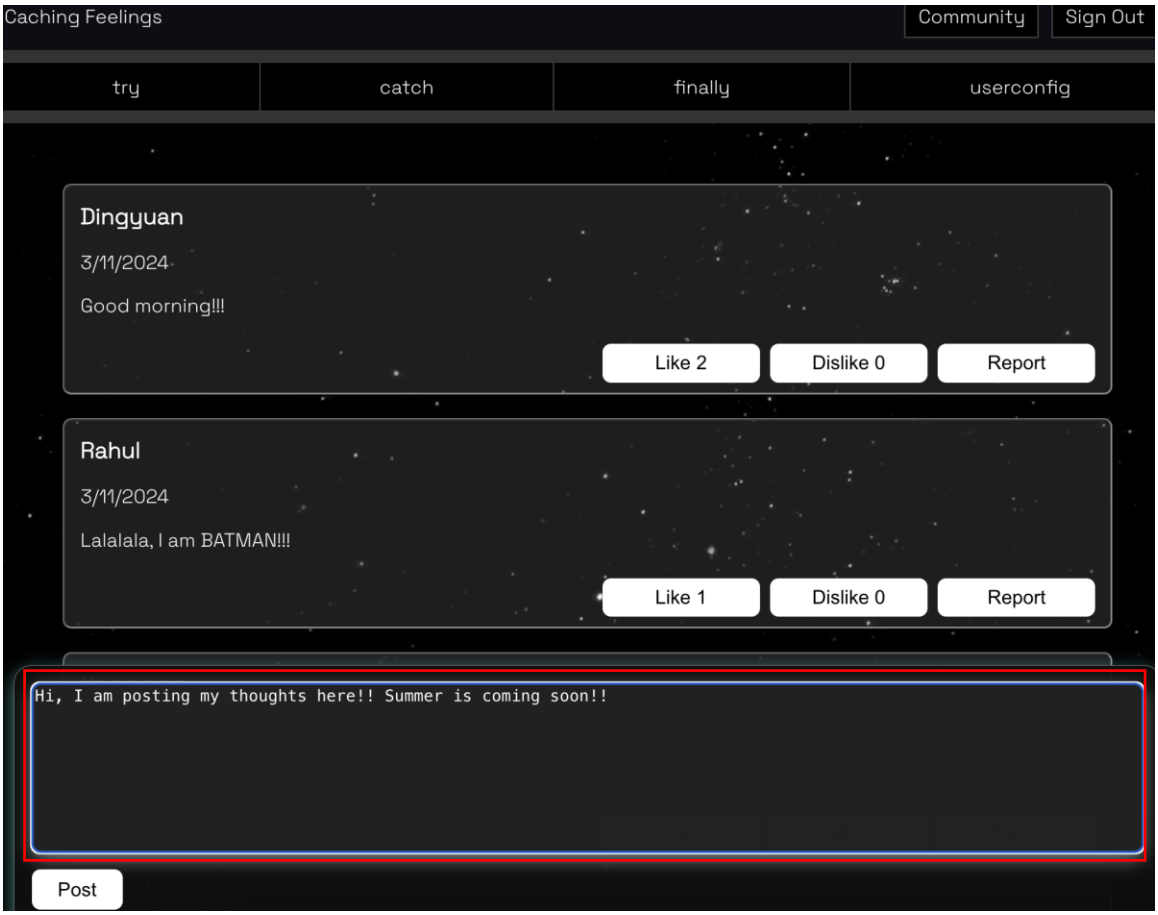
Step 1. Click the “community” button on the top right.

Step 2. You can see all the posts from users on this community page.

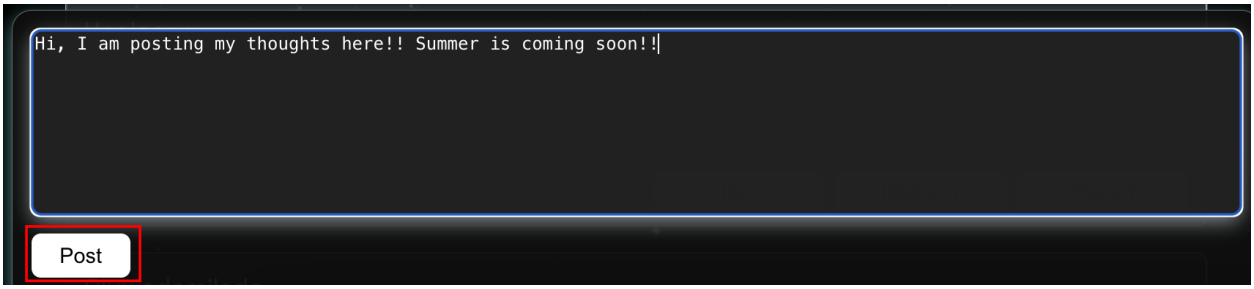


Post Thoughts to Community:

Step 1. In the community page, write your thoughts in the text box.

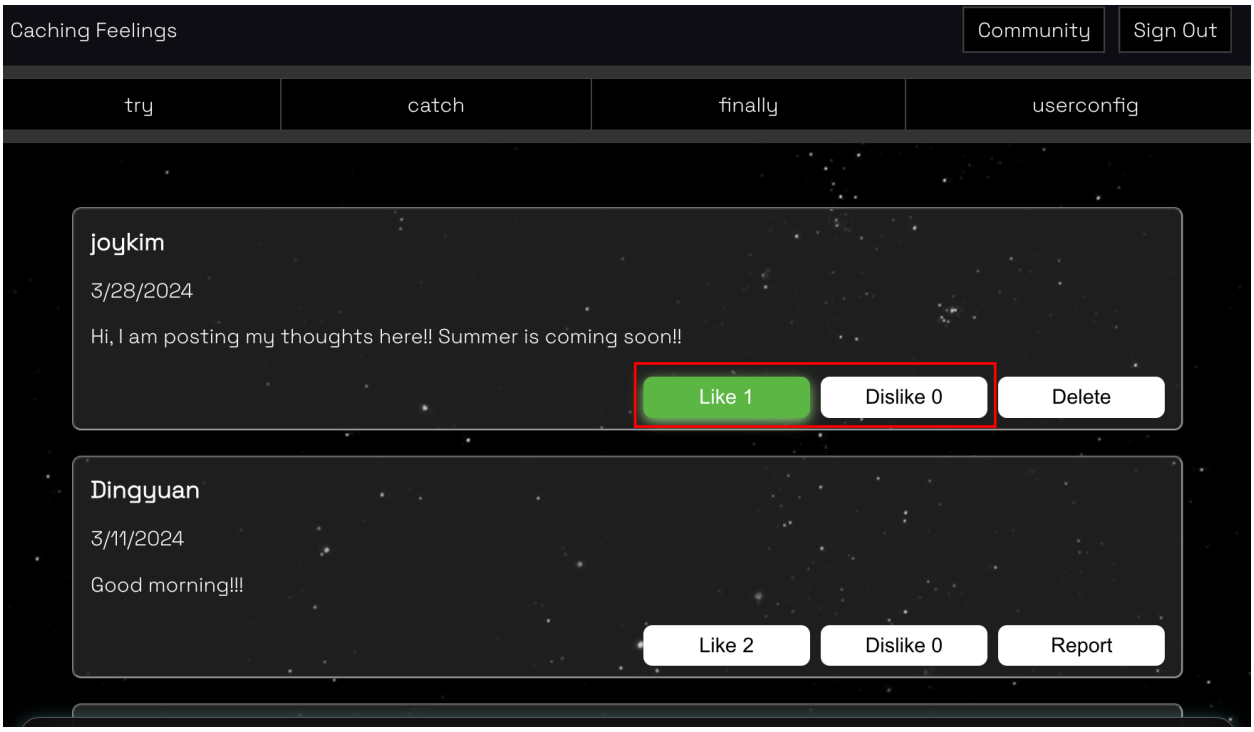


Step 2. Click the “Post” button to upload your thoughts.



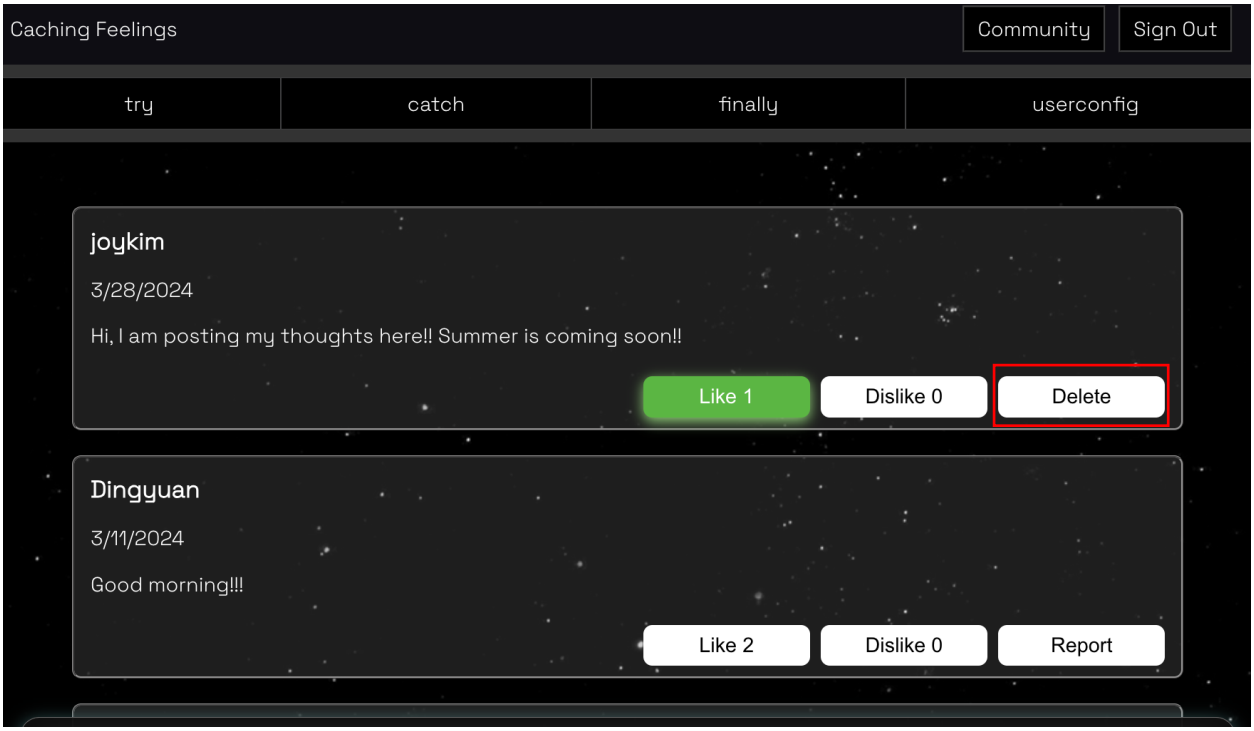
Like and Dislike Post:

Step 1. On the community page, within the list of posts, click either the Like or Dislike button to increment the count of Likes or Dislikes for that post.



Delete My Post:

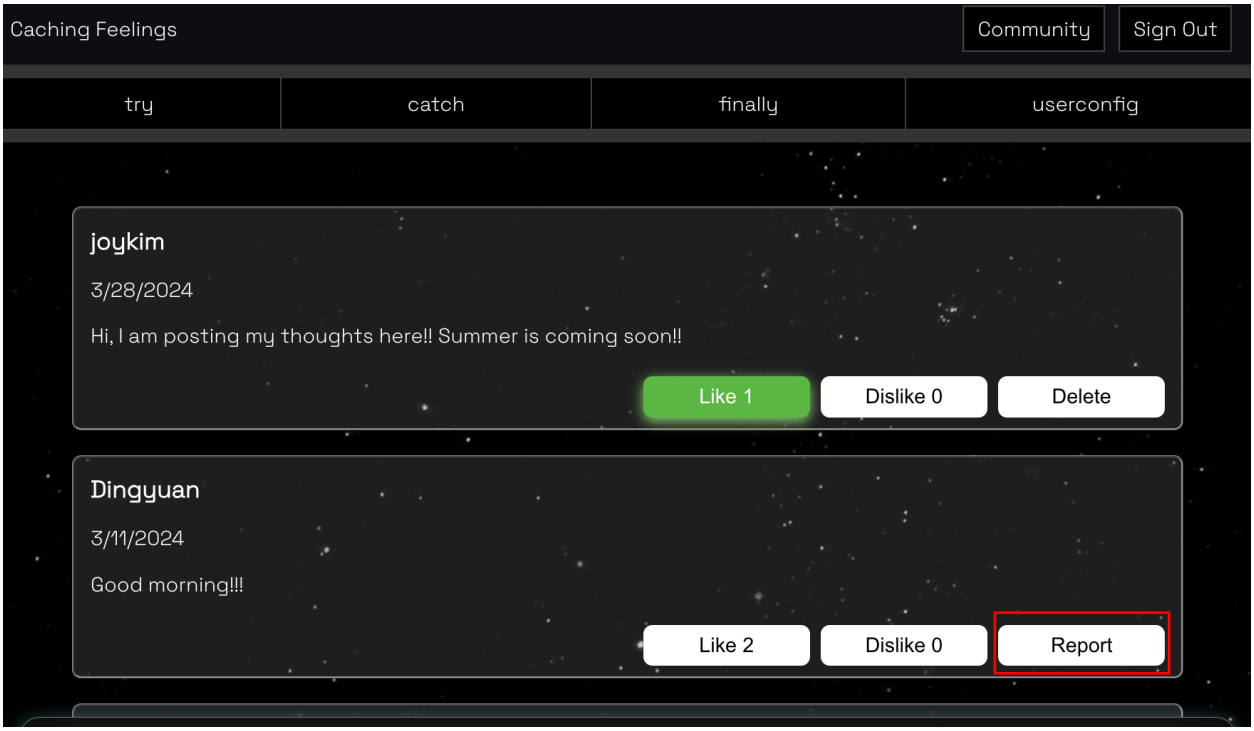
Step 1. On the community page, within the list of posts, click the “Delete” button to delete my post.



Report Post:

Step 1. On the community page, within the list of posts, click the “Report” button to delete that post in

my community.

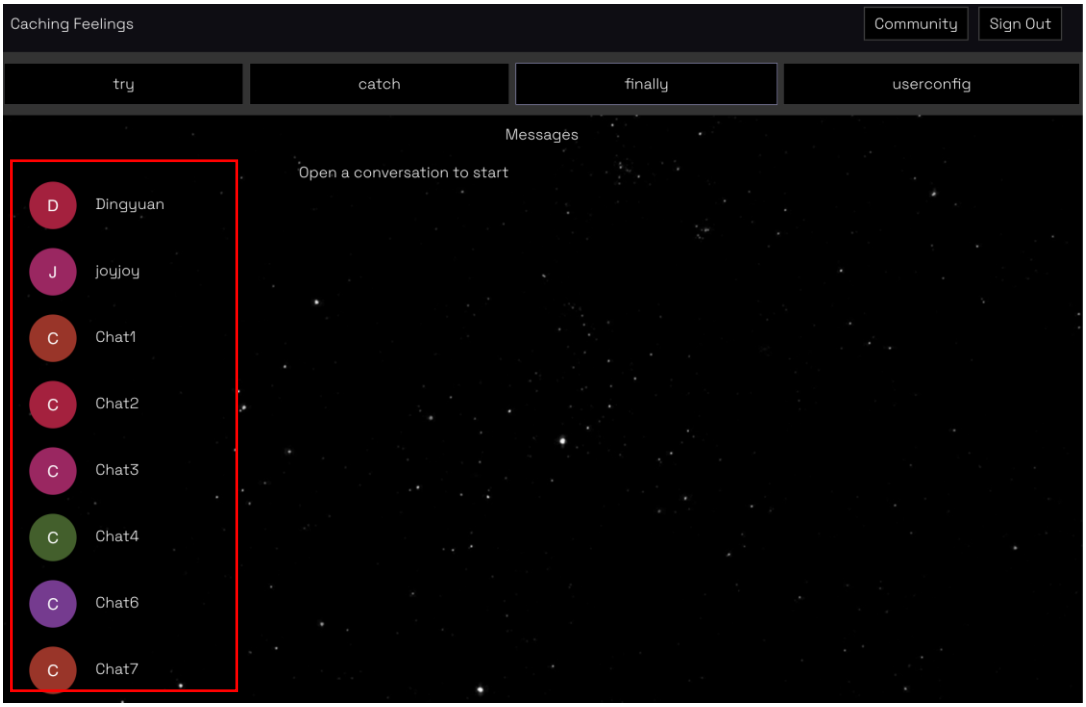


[Ice Breaker]

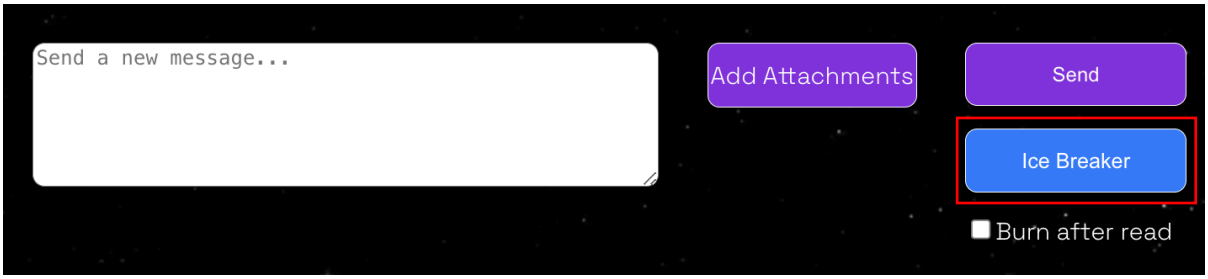
Sending Ice Breaker Messages:

Step 1. Go to the “finally” page.

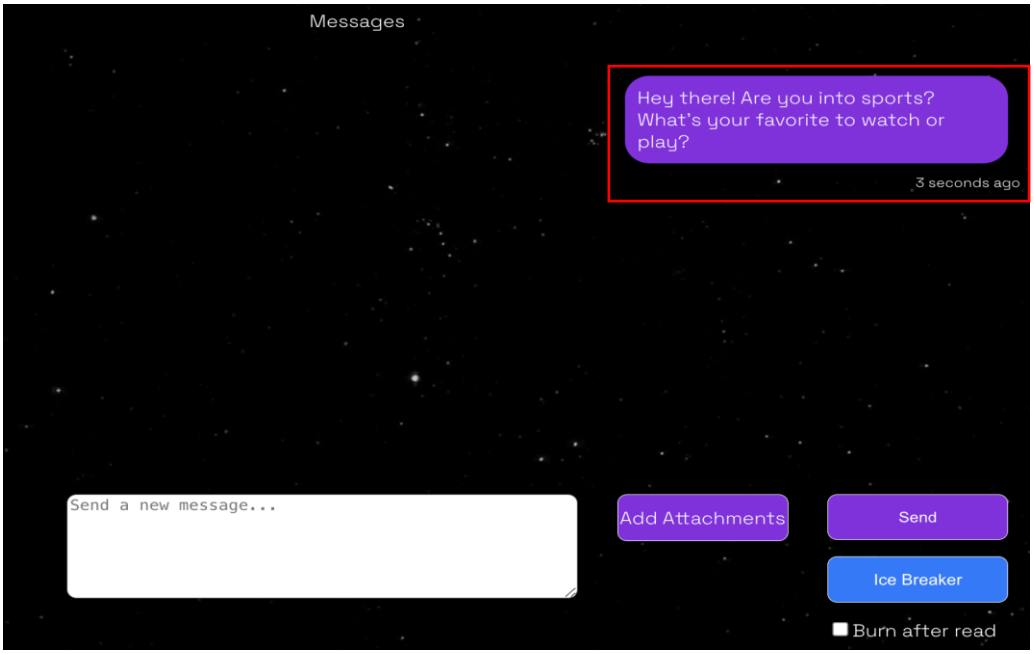
Step 2. Select a user from the list to chat with.



Step 3. Click the “Ice Breaker” button to send random message to the user.



Step 4. This will send random messages to start a smoother conversation.



Overall Arch and Design

https://github.com/cachingFeelings/cachingFeelings/tree/main/Documents/Sequence-diagrams

Infrastructure

Frontend:

- React
- Three.js

Backend:

- MongoDB
- JWT
- ExpressJS
- NodeJS
- AWS – image upload

Name and link

Our group had little experience with full stack development, but a couple group members had at least some familiarity with the MERN stack. Familiarity was the ultimate deciding factor when choosing the tech stack. As for the framework used to generate the 3D sphere, we explored WebGL and ThreeJS. ThreeJS was easier to work with and had the exact libraries that we needed for rendering the sphere that matched our vision.

We had a choice between storing our images using MongoDB as base64 text or another platform as files and we decided to go with AWS to allow users to upload multiple images and to keep the image management more efficient. AWS also allows us to generate signed images that can only be seen for a short while. We chose AWS since the free tier fit our needs perfectly and some members of the group

were familiar with AWS before.

Naming Conventions

Standard: Java naming conventions

Code

Key files: top 5 most important files (full path).

File path with a clickable GitHub link	Purpose (1 line description)
server/src/controllers/userController.js	Backend logic for user creation and matching. Provides the primary functionality of our app.
client/src/context/SignUpContext.js	Collects all the data that the user inputs during the signup process and sends it to the backend
client/src/components/homepage/try/Sphere.js	Fetches the user matches and provides 3D rendering of the returned matches
server/src/config/auth.js	Handles user authentication and authorization.
server/src/config/db.js	Establishes a connection to a MongoDB database using Mongoose

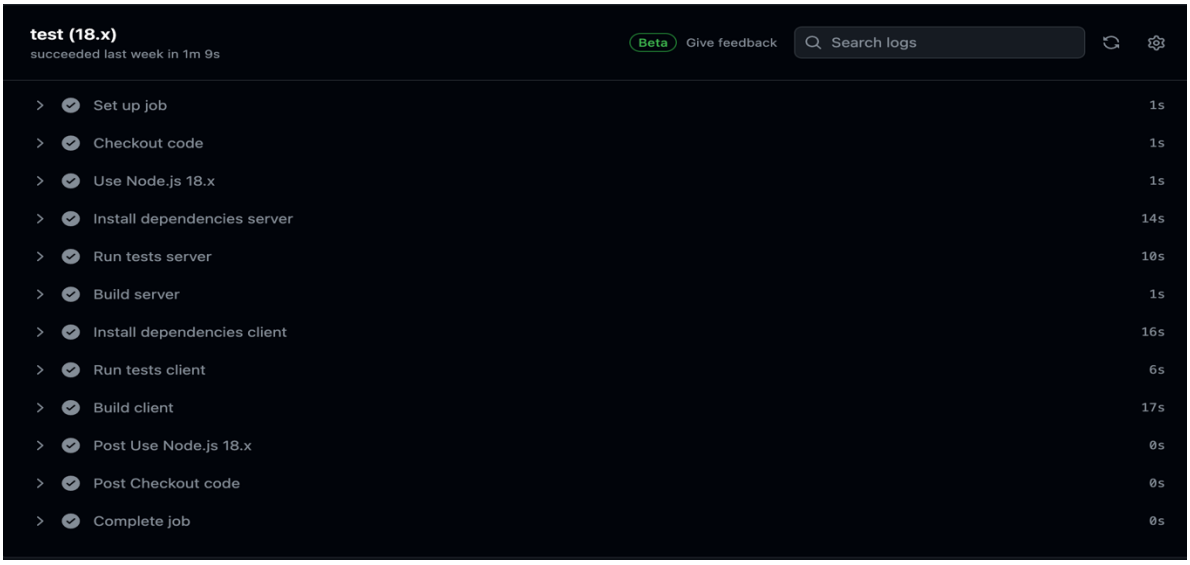
Continuous Integration and deployment (CI/CD)

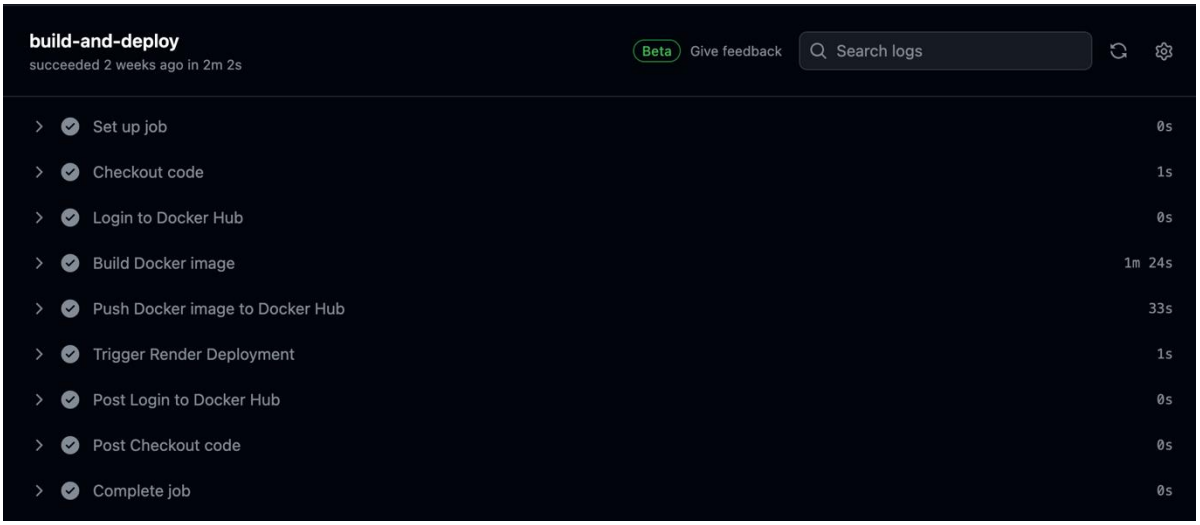
Our project's CI/CD pipeline is designed using GitHub Actions, distributed across various workflows to ensure reliable automation for integration and deployment processes.

The 'build.yml' workflow is triggered on a new release and is responsible for building our Docker images. It checks out the code, logs into Docker Hub using secure secrets, builds the Docker image, pushes it to Docker Hub, and finally triggers a deployment through a webhook.

Security checks are managed through the 'security.yml' workflow. This workflow includes steps for checking out the code, performing a static analysis security scan, and uploading the scan report as an artifact.

Out 'test.yml' workflow handles automated testing. It is triggered by pushes and pull requests to the 'develop' or 'main' branches. It runs tests for both the server and client parts of our application. It also includes a build step for each part to ensure they compile without issues.





Testing

Link to testing plan:
https://github.com/cachingFeelings/cachingFeelings/blob/ab02257733432d0205e31440b096e482a91ef72e/Documents/Testing_Plan_Gifhub.pdf

All the tests are run in CI and acceptance tests are written in the testing plan file.

10 most important unit tests:

Test File path with clickable GitHub link	What is it testing (1 line description)
[Test Account Registration] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/user.test.js	Ensure users can successfully register with valid information and receive a token or user ID in response.
[Test Login Functionality]	Ensures users can log in with correct credentials.

https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/user.test.js	
[Test Update Profile information] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/user.test.js	Ensures users can change their password given they provide the correct old password
[Test User Like Mechanism] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/like.test.js	Validates a user's ability to like another user.
[Test Creation of a New Conversation] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/convo.test.js	Ensures that once users have matched, they can successfully start a conversation
[Test Sending Messages] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/messages.test.js	Confirms users can send messages to their matches.
[Test Blocking Functionality] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/like.test.js	Ensures users can block unwanted matches.
[Test Posting to Community] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/community.test.js	Checks if users can create visible posts in the community.
[Test Liking Community Posts] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/community.test.js	Ensures users can like posts in the community.
[Test Reporting Community Posts] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/community.test.js	Validates the functionality for users to report inappropriate content.

5 most important integration tests:

Test File path with clickable GitHub link	What is it testing (1 line description)
[Test Account Registration] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/user.test.js	Ensure users can successfully register with valid information and receive a token or user ID in response.
[Test Update Profile information] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/user.test.js	Ensures users can change their password given they provide the correct old password
[Test User Like Mechanism] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/like.test.js	Validates a user's ability to like another user.

[Test Creation of a New Conversation] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/convo.test.js	Ensures that once users have matched, they can successfully start a conversation
[Test Posting to Community] https://github.com/cachingFeelings/cachingFeelings/blob/main/server/__test__/community.test.js	Checks if users can create visible posts in the community.

5 most important acceptance tests with links below. If your acceptance tests are done manually, you should have detailed steps on how to run the test and the expected outcome for test.

Test File path (if you automated the test) or as comments in Github issues (if it is with clickable GitHub link)	Which user story is it testing
https://github.com/cachingFeelings/cachingFeelings/issues/7#issue-2102711952	As a user, I want to sign up, so that I can have my own account.
https://github.com/cachingFeelings/cachingFeelings/issues/14#issue-2102773427	As a user, I want to find people that match my interests and preferences, so that I can get to know them.
https://github.com/cachingFeelings/cachingFeelings/issues/17#issue-2102781008	As a user, I want to send and receive messages with my matches, so that we can know each other more.
https://github.com/cachingFeelings/cachingFeelings/issues/23#issue-2102796201	As a user, I want to discover people in the community where everyone can post thoughts, so that I can find interesting people.
https://github.com/cachingFeelings/cachingFeelings/issues/269	As a user, I want to send a random message to the person I am currently messaging, so that I can start a smoother conversation.

Regression testing

Regression testing is run on every pull request via GitHub Actions using Jest. The regression tests must pass before the change can be merged into the main branch.

The 'test.yml' file (https://github.com/cachingFeelings/cachingFeelings/blob/ab02257733432d0205e31440b096e482a91ef72e/.github/workflows/test.yml) executes the regression tests.

```
Test Suites: 6 passed, 6 total
Tests:       60 passed, 60 total
Snapshots:   0 total
Time:        9.231 s
Ran all test suites.
```

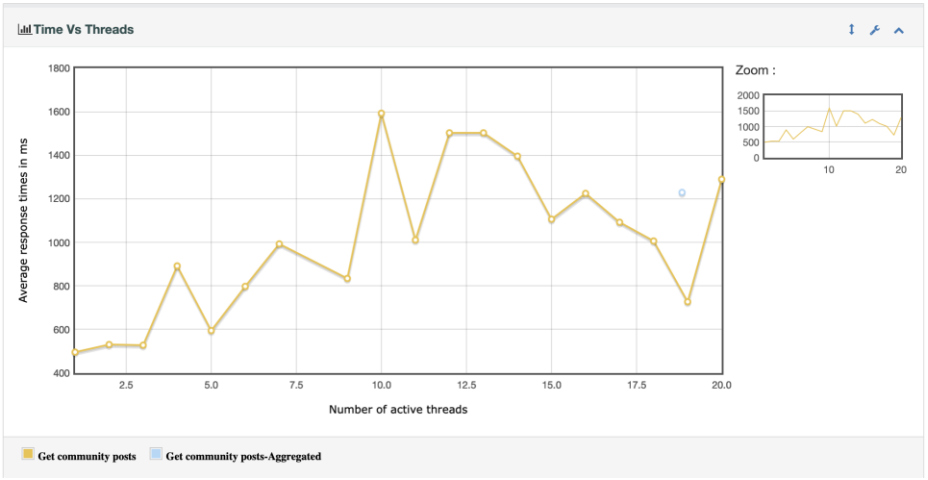
Load testing

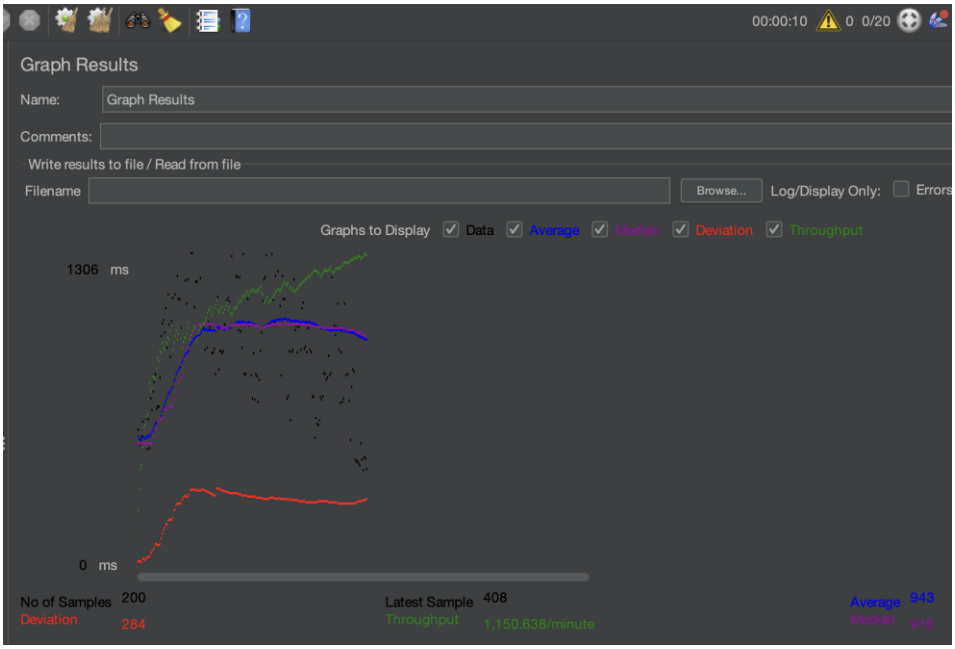
1) The load tests are done using JMeter. The test simulates 20 users making 10 requests each. The request chosen is a GET request to fetch all the posts found on the community page. Given that the GET request is fetching data that requires authentication to access, a user will need to be logged in and the web token will need to be copied into the Authorization header in the HTTP Header Manager tab.

2) Test report link:
https://github.com/cachingFeelings/cachingFeelings/tree/6a6e17971d5c001eae25b859c430a8854f013c68/Documents/LoadTests/HTMLReport

To view the report, download the repository to your local machine. Once it has been downloaded, go to the directory specified in the link and open index.html.

3) To run 200 requests takes 10 seconds. While all tests pass, there is a bottleneck observed as the number of threads reaches 10. The average response time spikes here and stays elevated as the number of threads increases. As expected, as the number of active threads increases, the response time goes up on average.





Security analysis

1) For the security analysis of our project, we utilized NodeJsScan, a static security code scanner specifically designed for Node.js applications. NodeJsScan is an open-source tool that provides comprehensive security scanning capabilities, making it an ideal choice for our project given that the majority of our source code is written in JavaScript for a Node.js environment.

To run, we executed it as part of our continuous integration (CI) pipeline, ensuring that every push to our repository was automatically scanned for potential security issues.

2) Refer to the appendix below for the results from the security scan.

The detailed scan report can be found at:
https://github.com/cachingFeelings/cachingFeelings/tree/a0d4ce05763a9fdc3aa8ff368351d6f27c1de52a/Documents/SecurityScanReport

Hardcoded Password in userconfig.js (Low Severity): This code in 'userconfig.js' contains a hardcoded password that is stored in plain text. This could lead to security vulnerabilities, as anyone with access to the source code would be able to see the password. It is advisable to use environment variables or a secure vault for storing sensitive information.

Hardcoded Password in LoginForm.js (Low Severity): A similar issue is present in 'LoginForm.js', where the password is hardcoded. This practice makes the application susceptible to attacks if the codebase is exposed. Passwords should be encrypted and stored securely, not visible within the application's code.

Hardcoded Password in SignupForm.js (Low Severity): In 'SignupForm.js', the password is again hardcoded in plain text, which poses a security risk. Secure password heading strategies, such as hashing and encryption, should be implemented to prevent the exposure of passwords.

Appendix

All Issues (3)

All (3) | Critical (0) | High (0) | Medium (0) | Low (3)

Rule	Severity	Source location	Message

	LOW	userconfig.js	A hardcoded password in plain text was identified.
			`'currentPassword': password,`
			`'password': newpwd,`

	LOW	SignupForm.js	A hardcoded password in plain text was identified.
			`'username': data['username'],`
			`'password': data[`

	LOW	LoginForm.js	A hardcoded password in plain text was identified.
			`'username': username,`
			`'password': passwo`