Anonymous Messaging Web Application (AMWA)

https://github.com/Adamkadaban/SWE-2022

Group 30: Soft Engineers
Product Manager: Adam Hassan
Scrum Master: Harrison Stark
Development Team Member: Carl Noll
Development Team Member: Gregory Bouraoui

1. Project Description

1.1 Problem Statement

We wanted to create a product that encourages users to open up and engage with their coworkers online. Whether about minor issues and complaints or about serious workplace abuses, an online forum like ours can help resolve them. Communication is key. Indeed it is how we can support each other and address conflicts [1].

While many want to communicate online about their work, it is often that they will not do so in fear of retaliation from their employer. This pattern is one that spans skyscraper offices to the fields where we grow our food. Those particularly vulnerable are undocumented workers who cannot express concerns with their employers either due to language barriers or job insecurity because, in an exploitative job, their feedback and well-being is pushed aside [2]. Managers have a lot of power, and companies will go to great lengths to protect their brands. Our app - AMWA - enables users to express concerns anonymously.

Another motivator for making this app is to collect information about disparities in the workplace. By gathering data entered by users, it might be possible to analyze and showcase that data in a way that will demonstrate work disparities. Indeed, the tech field, for example, is so male-dominated that we often see a difference in the way men and women are being paid based on their gender. Our hope is that by making account signup and data reporting anonymous, we can gather information regarding these barriers to equity and set the groundwork for more transparent communication between workers, managers, and companies.

1.2 Solutions

AMWA is a discussion forum for workers. Unlike a typical chat app, the space is completely anonymous, meaning users can share their experiences free of status and risk.

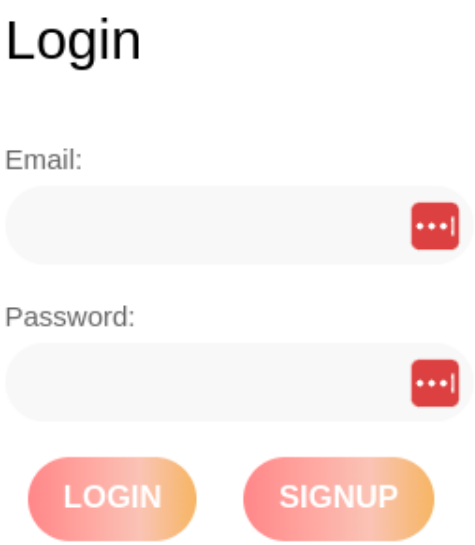
Our product gives users the space to speak their minds freely. Users will receive positive reinforcement by getting attention from others via upvotes and downvotes and a commenting feature. Users can also initiate DMs with another user to talk one-on-one about more personal issues. Thus, users can do their part to see how they are being paid, and what position is getting paid the most, and can thus get a sense of how the company fares when it comes to equitable pay.

A central function of our app is security. We maintain users' anonymity from onboarding to posting and messaging. Posts are wrapped in json files and sent to MongoDB which is maintained in the cloud. When a user logs in, their passwords are hashed and salted so they are resilient to hacks. Our app keeps the user’s identity safe throughout the user flow.

1.3 Features and Functionality

1. Login and Signup

Safely onboard the user and verify the user's identity

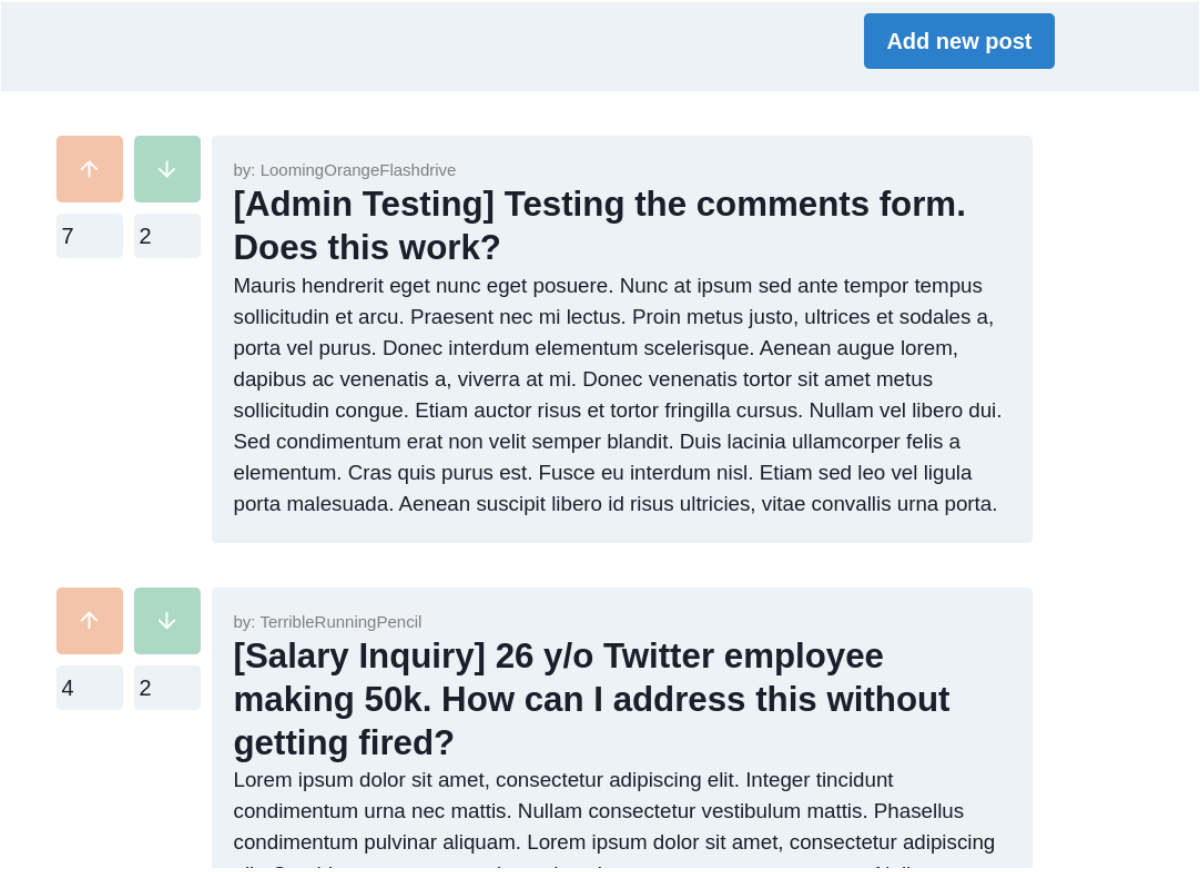


Features include:

- A "sign up" button. While entering information, the user will receive confirmation that the information entered into the sign-up fields is valid
- The user will input a name, email, and password
- When the user asks to make an account, a random username will be generated that consists of <vowel><Noun><Verb>
- Once the user has created an account, they can use their email/username and password on the front page.

2. Post Forum

Enables users to post anonymous comments

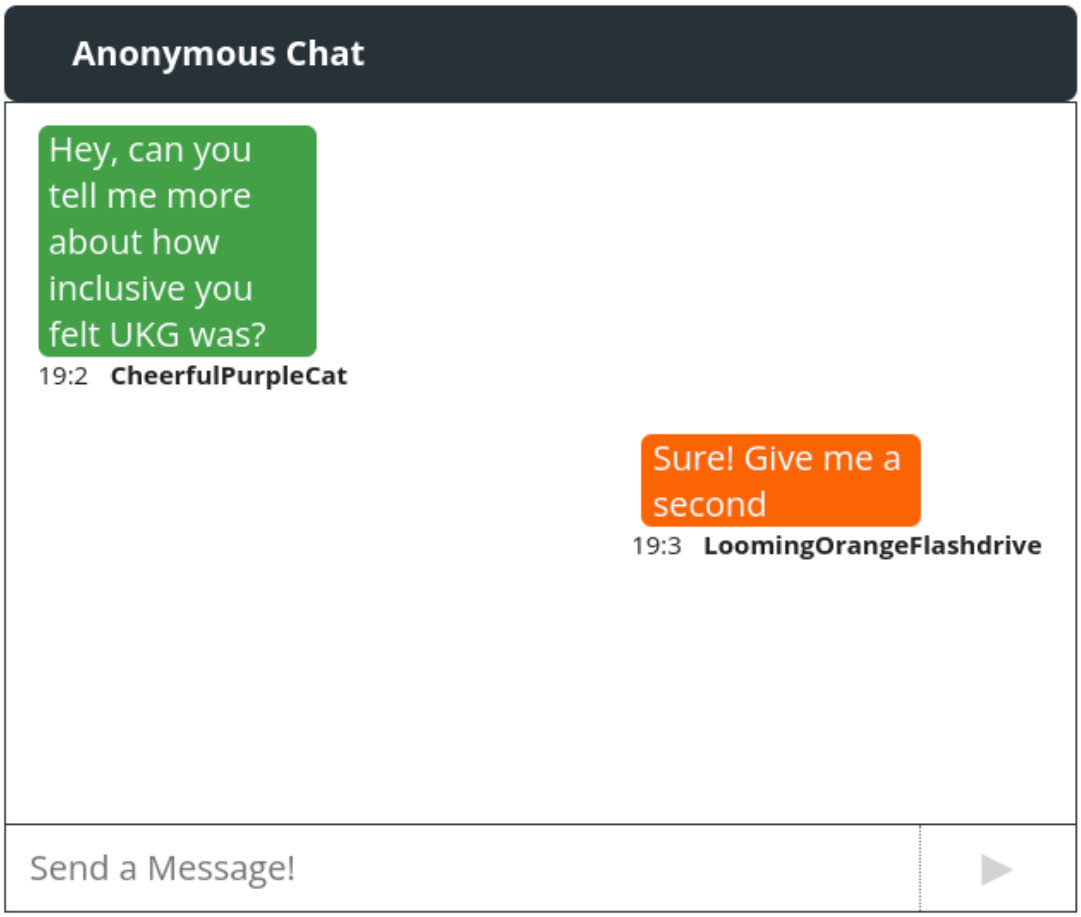


Features include:

- Comments page with a button labeled "Add New Post".
- A text box for the user to write the main contents of their comment. The user can enter tags to identify the specific company, issue, pay, etc.

3. Anonymous Messaging

Enables users to message people that may need help regarding a post.



Features include:

- User can select another user to message
- Messages are deleted once the secure messaging session is finished

1.4 System Models

Architecture: Monolithic Client-Server Pattern

1. System Components and Tech Stack:
    1. Presentation layer
        a. React Frontend
    2. Application layer
        a. RESTful Flask API
    3. Business Logic Layer
        a. Flask Backend
    4. Data access layer

a. MongoDB Database
b. Firebase Database

2. Infrastructure, Backend,and Database

Application layer

a. This includes most of our coding, database CRUD functions with api calls, GET and POST mappings, etc.

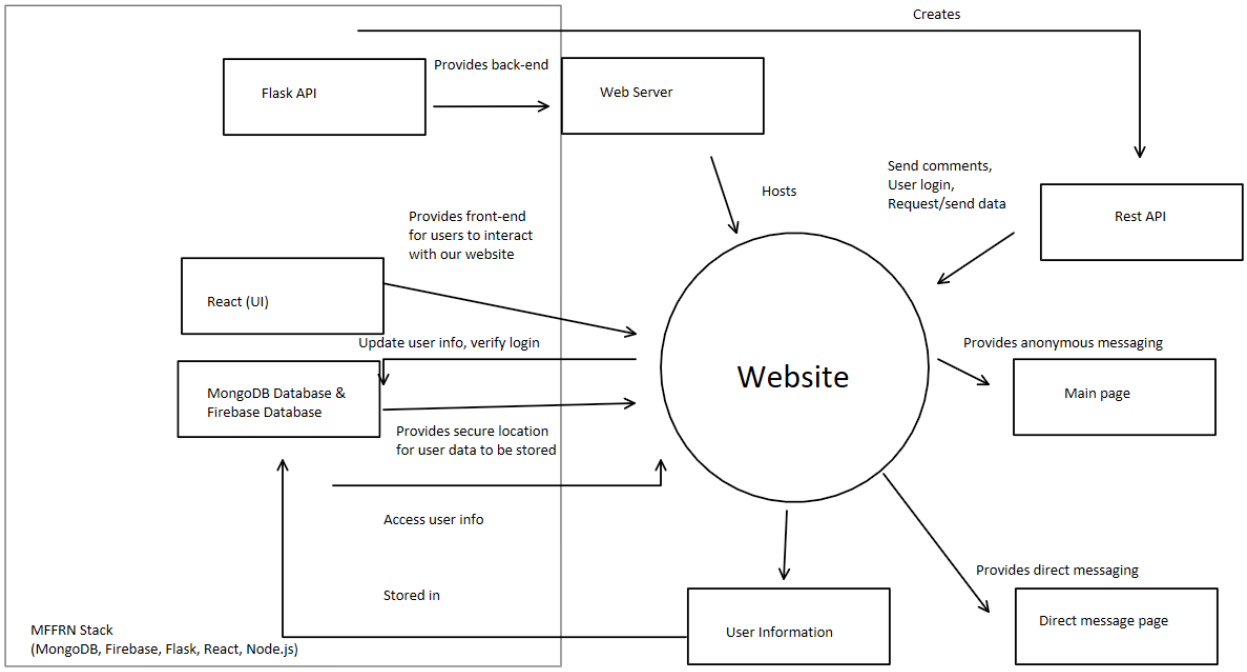
Business logic layer

a. Conditionals that describe the program flow
b. When one action finishes, it tells the application what to do next

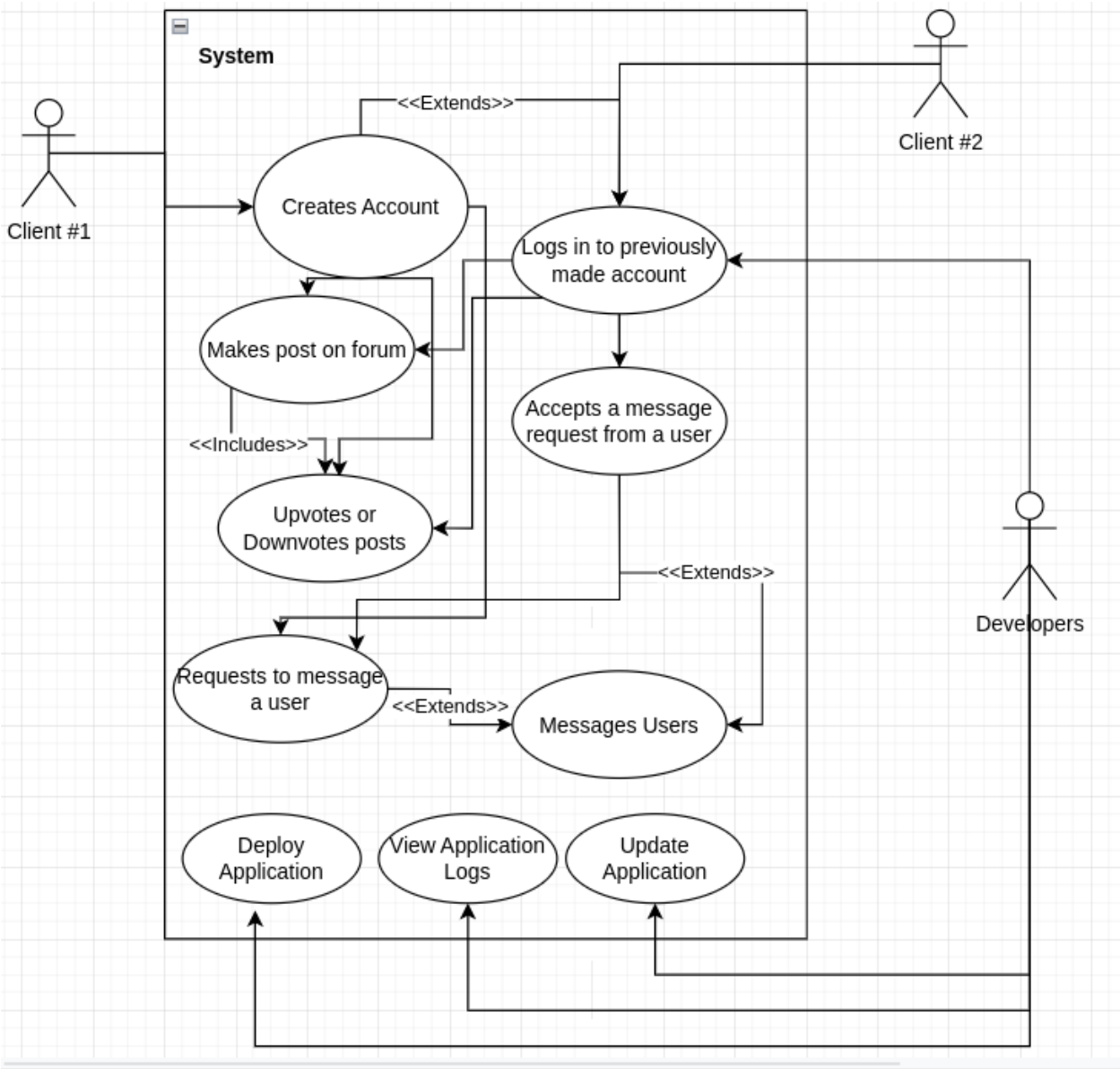
REST API will provide several services

a. User signup and login
b. Comment creation and retrieval
c. Message creation and retrieval
d. Write, Update, and Read user input from pay form to database

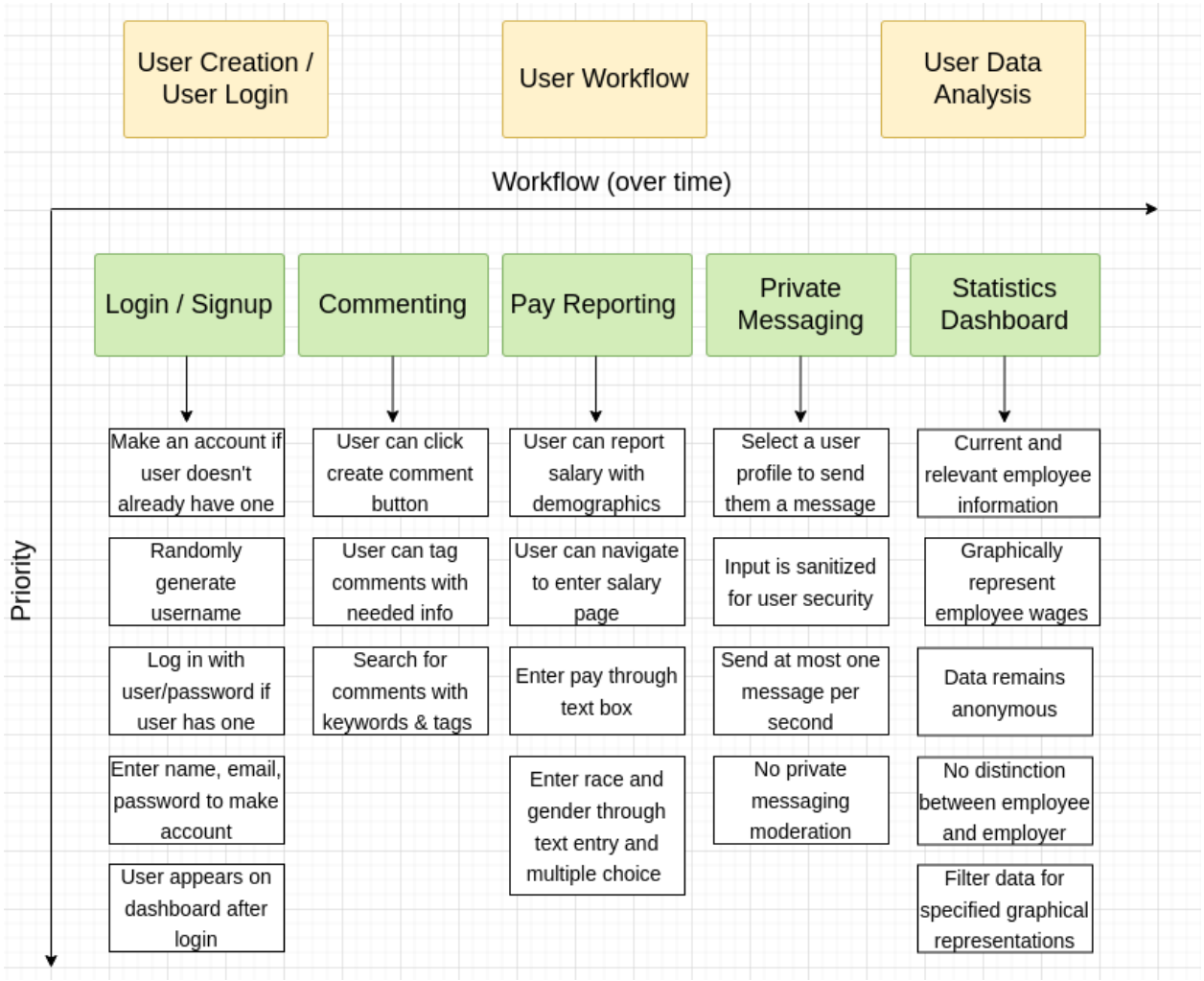
System Context Model



Use Case Model

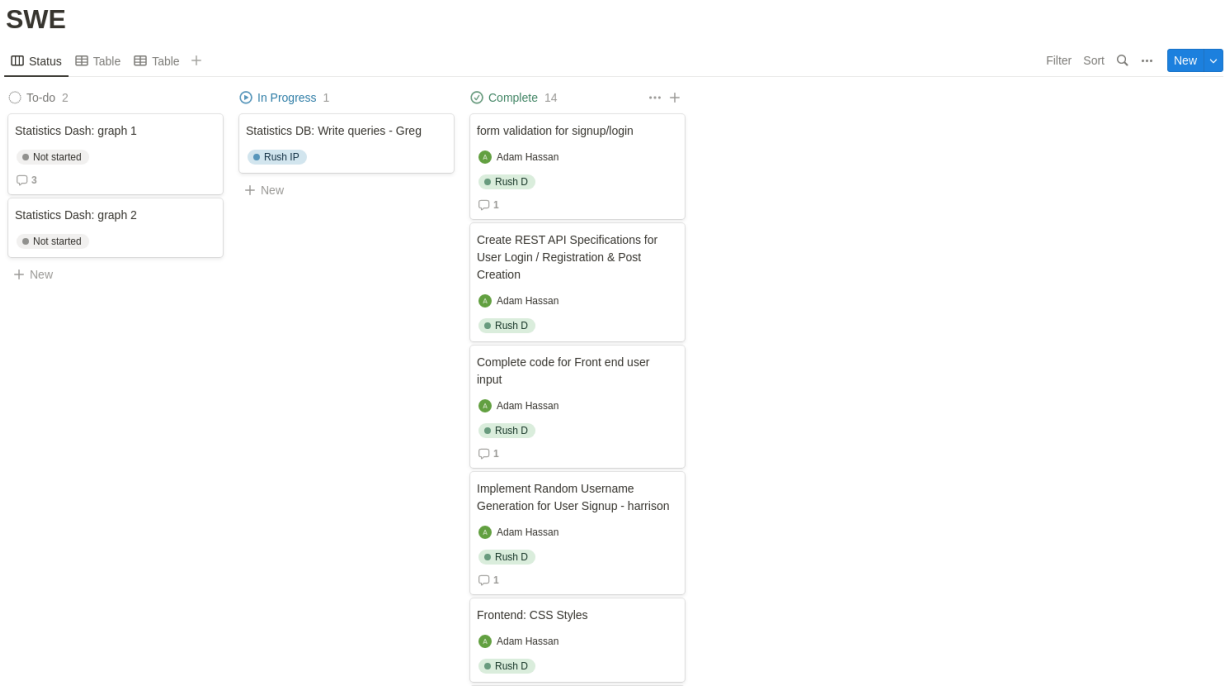


User Stories Map



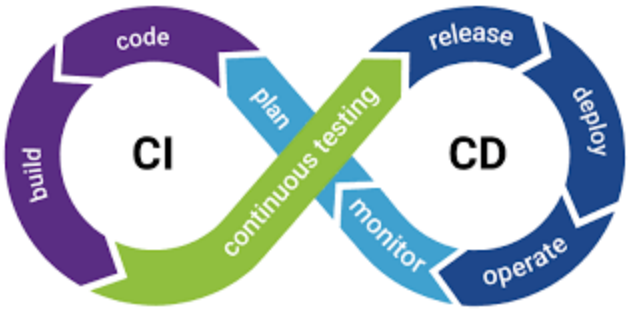
2. Code Management

We utilized a Kanban Board on Notion.so to manage app feature development and timelines.



2.1 Version Control and CircleCi

1. Version Control with Github
    a. Code is stored at a centralized repository
    b. Contributors clone a repository (copy a version onto their local machine to modify any way they please)
    c. Use terminal and git pull to pull changes made by others and keep local project up to date
    d. When someone wants to contribute a change, they stage the commit with ‘git add’, then push to github.
    e. The resulting pull request (PR) is reviewed and may be approved, rejected, or modified by others. Once approved by one of the team developers, the branch merges onto main (master branch)
2. Continuous Integration with CircleCi
    a. Automated builds and code reviews
    b. Tests software each time new code is pushed and merged
3. Pair Programming on Google Colab



2.2 Test Plan

We have enumerated several well-defined tests below

ITEM NUMBER	TEST CASE ID	TEST CASE DESCRIPTION	PRECONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	POSTCONDITION	ACTUAL RESULT	STATUS	COMMENTS
1	regUser	Register User - Enter new email and password	Go to web app	Choose "generate" to select a generated username Enter email Enter password Click Submit	Gmail: admin@gmail.com Password: pichu	Proceed to homepage	The user input is processed and a record is created within the user DB User can now login to post/dms server with these credentials	Redirect to homepage	Pass	Email invalid: "Submit error: enter valid email,"
2	inpUserData	Provide Salary Data - Enter job title, company name, location, optional: enter pay data	User has registered an account	Enter job name Company name Add Location Enter pay data Optional demographics	Software engineer Ebay Rocky Dr, Tampa, FL $80,000 annual White, non-binary	Redirect to stats dashboard	The user input is processed and a record is created within the stats DB, The user can view how their own data relates to the previously collected data & general stats.	Redirect to stats dashboard	Pass	Submit failed, missing required data. Page will display error message prompting user to complete entries instead of redirecting.
3	loginUser	Verify the login of a user	User must have an account (ie has registered) before signing in	Enter Username Enter Passsword Click Login button	Email: admin@gmail.com Password: pichu	User with the admin email should be loggied in	User homepage should be shown	User homepage is shown	Pass	If email and/or password are invalid, page disp "either your email or password is incorrect"
4	postComment	Verify that a user can post a comment	User must be logged in and on the homepage	Go to the homepage & click "post comment" User can then type comment and click "submit"	Comment: "I get paid 70k at Google as a QA Engineer" Tags: Salary question	The comment should be posted under the tag "Salary Question"	Comment is available on the homepage	Comment is available on the homepage	Pass	Unable to post comment (status code)
5	sendMessage	Verify that a user can send a private message	User must be logged in and on the homepage	User clicks a profile and is taken to a private messaging window user types a message and hits submit	Message: "hello u coming to lunch?"	The message is sent	The message is sent and viewable to the recipient, message screened by input sanitization	The message is sent and viewable to the recipient	Pass	Unable to send - server not running // Unable to send - input contains security threat

The following are test categories that we believe are especially important:

1. Application testing
    a. Confirm that generation of new users requires a unique email ID.
    b. Verify that user data gets inputted and represented properly in the MongoDB database.
    c. Verify that user identification remains anonymous

2. API Testing
    a. Explore edge cases and test for different parameters of the api calls that engage all code branches. Verify all functions and find any places the code fails
    b. Test if api is calling another api
    c. Verify that multiple calls work in succession. For example, POST mapping updates value and the new value retrieved from GET request has been updated.
    d. Verify return codes
        i. If the api is calling on another api
        ii. the api does not return anything
        iii. the api returns the correct HTTP Status code

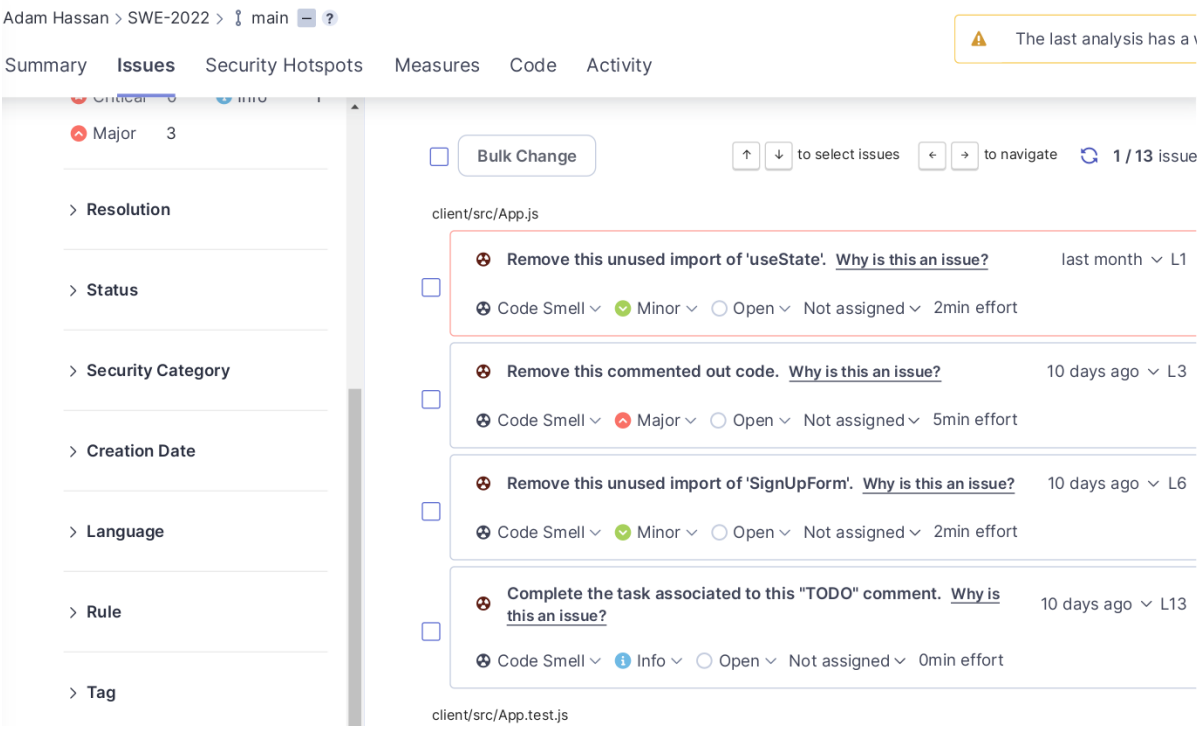
iv. the api fails due to invalid JSON formatting

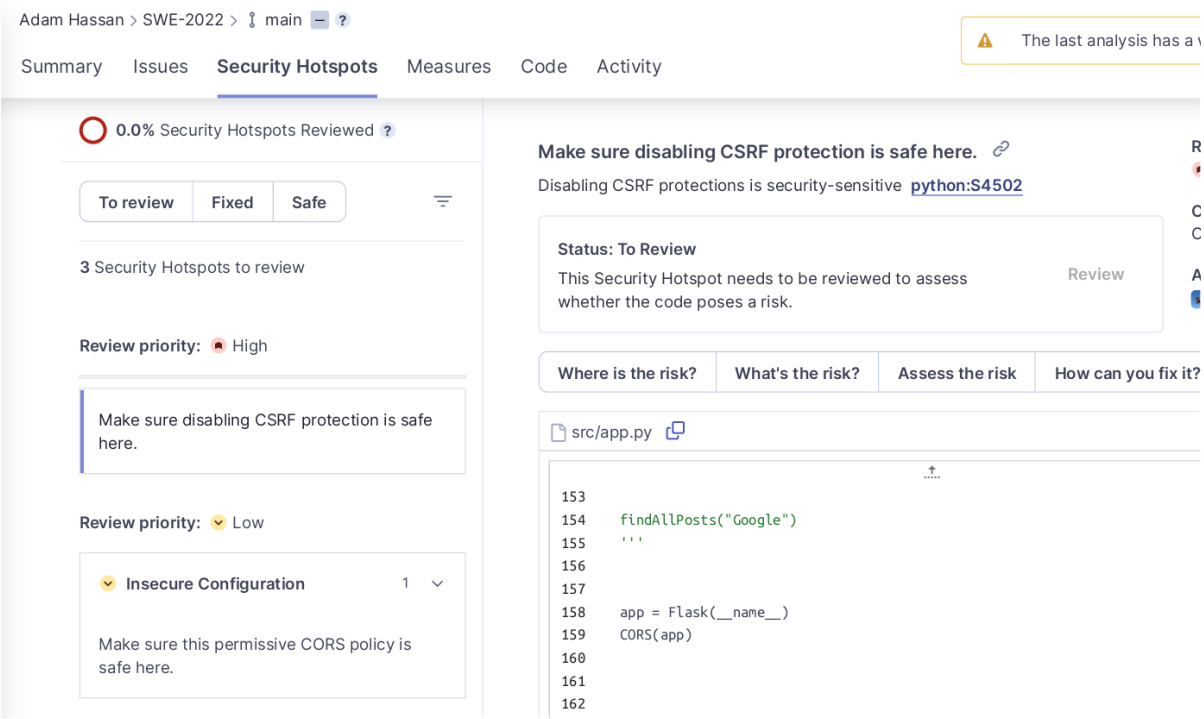
2.2 Static Code Analysis

I have included screenshots of the partial Sonarcloud reports here for brevity, but the full reports are linked:

- Security Hotspots - SWE-2022 - Adam Hassan.pdf
- Static_Analysis_Report_BEFORE.pdf
- Static_Analysis_Report_AFTER.pdf

Part 1





Issues (in order):

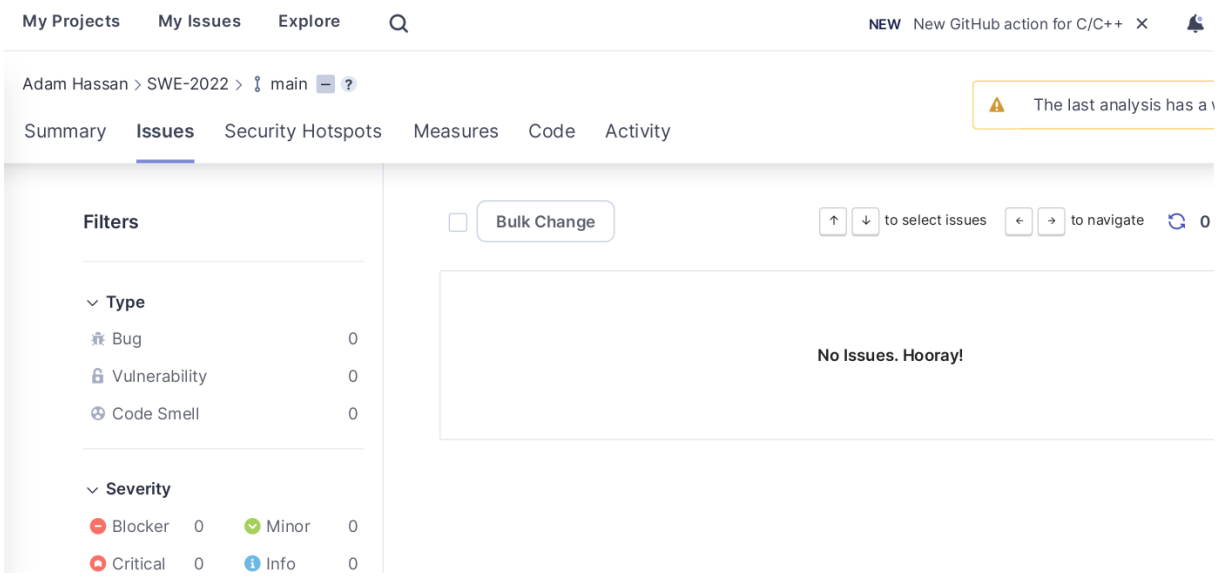
- Remove this unused import of 'useState'.
    - Because this import is unused, the relevant line can be deleted.
- Remove this commented out code.
    - Because this was old code for testing purposes, it can be deleted.
- Remove this unused import of 'SignUpForm'.
    - Because this import is unused, the relevant line can be deleted.
- Complete the task associated to this "TODO" comment.
    - This task, being error handling, was completed.
- Remove this unused import of 'screen'.
    - Because this import is unused, the relevant line can be deleted.
- Remove this commented out code.
    - Because this was old code for testing purposes, it can be deleted.
- Remove this useless assignment to variable "setError"
    - Because this variable is unused, the relevant line can be deleted.
- Remove the declaration of the unused 'setError' variable.
    - Because this variable is unused, the relevant line can be deleted.
- Add the "let", "const" or "var" keyword to this declaration of "passwordValidation" to make it explicit.
    - Although the code was still functional, we decided that the variable would be appropriate as a const type. Thus, we added the "const" keyword before the variable declaration and assignment.
- Add the "let", "const" or "var" keyword to this declaration of "regexValidation" to make it explicit.

    - Although the code was still functional, we decided that the variable would be appropriate as a const type. Thus, we added the "const" keyword before the variable declaration and assignment.
- Add the "let", "const" or "var" keyword to this declaration of "errorLogin" to make it explicit.
    - Although the code was still functional, we decided that the variable would be appropriate as a const type. Thus, we added the "const" keyword before the variable declaration and assignment.
- Add the "let", "const" or "var" keyword to this declaration of "errorPasswordLength" to make it explicit.
    - Although the code was still functional, we decided that the variable would be appropriate as a const type. Thus, we added the "const" keyword before the variable declaration and assignment.

Security Hotspots:

- Make sure disabling CSRF protection is safe here
    - Here, we are using CORS (Cross-Origine Resource Sharing) to ensure that the frontend can communicate with the Flask backend, which Soncarcloud complains about because CSRF (Cross-Site Request Forgery) vulnerabilities can be introduced when it is possible for one website to reference another, which can lead to user impersonation. Because we are only using the API on the website to access MongoDB, this vulnerability is not an issue and the code does not need to be changed.

Part 2



- All issues have been resolved.

3. Technical Details

3.1 Technical Details

Each service (login/signup, commenting, and messaging) of the application is running in its own process, which allows for better security and more resilience in the case that something breaks. User information is stored securely on MongoDB and is sent from the app to the database with a Flask REST API. All passwords are hashed and salted.

The frontend is all made using the React JS framework, while the backend is written in python using Flask. One notable exception to the usage of MongoDB for storage is the comments page, which uses Firebase for ease of access.

3.2 Technical Debts

- Naive Technical Debt:
    - No member of our team had previous experience with React JS, MongoDB, or Firebase prior to the start of the project.
- Strategic Technical Debt:
    - Currently, a naive approach to routing is employed due to the variety in the number of features that the application contains.
    - Separate services is secure and less prone to breaking, however, using a single routing system would allow for ease of setup and make the app more lightweight.
- Unintentional Technical Debt:
    - No search feature was implemented as filtering of posts was quite slow.

3.3 Login and Access Credentials

Set up a MongoDB cluster with a user that has read/write permissions.

In `src/creds.config`, enter information in the following format:

```
mongo_db_username
mongo_db_password
password_salt
```

In `client/comments/.env`, enter information in the following format:

```
REACT_APP_FIREBASE_API_KEY=""
REACT_APP_FIREBASE_AUTH_DOMAIN=""
REACT_APP_FIREBASE_PROJECT_ID=""
REACT_APP_FIREBASE_STORAGE_BUCKET=""
```

```
REACT_APP_FIREBASE_MESSAGING_SENDER_ID=""
REACT_APP_FIREBASE_APP_ID=""
REACT_APP_FIREBASE_MEASUREMENT_ID=""
```

3.4 Installation Instructions

Setup

Note that the following requires login and access credentials[3.2] to have been set up.

This requires Node js

On Unix and Windows:

```
git clone https://github.com/Adamkadaban/SWE-2022
cd SWE-2022
cd ./src

python3 install -r requirements.txt

npm --prefix ./client/login install
npm --prefix ./client/comments install
npm --prefix ./client/messaging/client install
npm --prefix ./client/messaging/server install
npm --prefix ./client/about install
```

Usage

Run the following commands in the terminal:

```
cd ./src
python3 app.py
cd ../

cd ./client/login
./start.sh

cd ../comments
./start.sh

cd ../messaging/server
./start.sh
```

```
cd ../client
./start.sh

cd ../../about
./start.sh
```

The app can now be accessed at http://localhost:3000

4. Risk Management Plan

4.1 Risk Management Plan

Risk	Probability	Effect of Risk	Strategy
Node has package inconsistencies	High	Insignificant	Developers can meet and refactor the code to ensure everything is up to date and consistent.
Browser blocks CORS on website	Low	Serious	Code can be implemented to alert users and developers that their browser is blocking Cross-Origin Requests.
A significant vulnerability is found on the webpage	Low	Serious	Sonarcloud can be configured to give live updates on vulnerabilities in the codebase such that things can be fixed as soon as possible.
Team members are away and cannot work on the project	Moderate	Tolerable	Developers can work together to offset the lost time working on the code.
Bugs might be found in deployment	Moderate	Tolerable	Users can report issues on GitHub, which will alert developers to fix them.
Sonarcloud runs out of free hourly scans	Low	Low	We can switch to an open-source alternative like SonarQube to run scans periodically.
Not enough time has been allocated for some features	High	Serious	The most important features should be prioritized and developers should meet to determine a plan for reorganizing the sprint backlog.

4.2 Software Quality Attributes and Explanations

Several steps (both before and after deploying the app) have been taken to increase code quality. These include the following:

- The code has no heavily nested statements, which makes it easier for developers to work on and update in the future.
- None of the code uses any deep or convoluted inheritance, which also promotes readability, and thus clean and safe development.
- All input forms are validated to prevent vulnerabilities that may be caused by code injection or anything similar.
- The backend does range checks on numerical inputs to ensure they are valid. This prevents unexpected requests from being made, which could have adverse effects on the server if unregulated.
- Code smells, Issues, and Security Hotspots as reported by SonarCloud have been refactored and resolved to ensure that code is not only readable and up to standards, but is also secure.
- Features of the app have been segmented to ensure that if one feature is down for development or otherwise, the rest of the website can still work as needed.
- Variables that can be used as constants are defined as such, while others are defined as dynamic variables. This promotes better readability, which will make future development easier.
- Passwords are hashed and salted for security in case they are ever leaked.
- The backend has exception handling for config errors and keyboard interrupts, and the frontend includes error handling for login/signup, connection issues on the messaging server, and Firebase connection issues. This ensures that it is clear we know what errors occurred when the app runs if something is not working.

References

[1] https://time.com/5353848/salary-pay-transparency-work/
[2] https://www.dol.gov/sites/dolgov/files/general/labortaskforce/docs/508_union-fs-8.pdf