Mood.ly

By Group 97

Name	UOL Student ID	Email
Ahmad Zaeem Bin Suhartono	210201473	ahmadzbs001@mymail.edu.sg
Chai Jia Min Shermin	200553191	jmscjcha001@mymail.sim.edu.sg
Jaztin Buenaventura	200629267	buenaven002@mymail.sim.edu.sg
Lei Jiaying	220253664	jlei003@mymail.sim.edu.sg
Nicholas Kheng Zhang Yi	200617471	nzykheng001@mymail.sim.edu.sg

Table of Contents

Chapter 1: Background

1.1 – Introduction

When it comes to the diagnosis, treatment and prognosis of mental illness, it is widely agreed upon to identify and address issues early. Therefore, our team created a mental health self-help web application to aid in the identification and early address of mental health issues emerging from society's youths, namely secondary school students.

Having access to a mental health self-app web application can encourage youths to constantly keep track of their mental health and well-being. The web application will include important features to facilitate the process of keeping youths in check while making sure that help can be easily obtained if needed.

1.2 – Aim of the Project

This project aims to help youths to identify potential mental health red flags and aid in their address of mental health issues such as anxiety, PTSD and depression. The application focuses on the early detection of mental illnesses, providing accessible mental health services to students and directing them to different resources for help. The objectives of this project are to provide these youths with a platform to monitor and track their mental health status.

1.3 – Scope of the Project

Since the agile process is centred on validating and refining products based on user feedback, the Minimum Viable Product (MVP) is essential for Agile Development. The scope helped us with determining the MVP since we can spend more time working on the features after we narrowed down the importance of a mental health self-help web application. This was done using user and market research which was presented in our previous proposal. Feedback was gathered based on our application, and is being constantly updated for our final product to be user-centred.

The features available in our product include:

Features	**Uses**
Register (Sign-Up)	Allow new users to sign-up or existing users to log in
Profile	Allow users to view their profile that they have input during registration
Visual Mood Tracker	Allow users to update their daily mood, which can be changed throughout the day
Tips (Help)	Display some tips from external sources, users can be directed to those sources
Forum	Display some forums created by different users, and for them to interact
Journal	Allow users to reflect on their daily activities, and type it out to keep track

Chapter 2: Planning and Research

2.1 – Methodology

User Centred Design (UCD) and Agile methodologies have been used throughout this project. The planning was done in the first half of the project, where we focused on the needs and wants of the user when it comes to a mental health self-help web application. From what we have gathered, we then went on to the development process, which is the second half of the project. This was where we designed and prototyped our web application based on the user requirements.

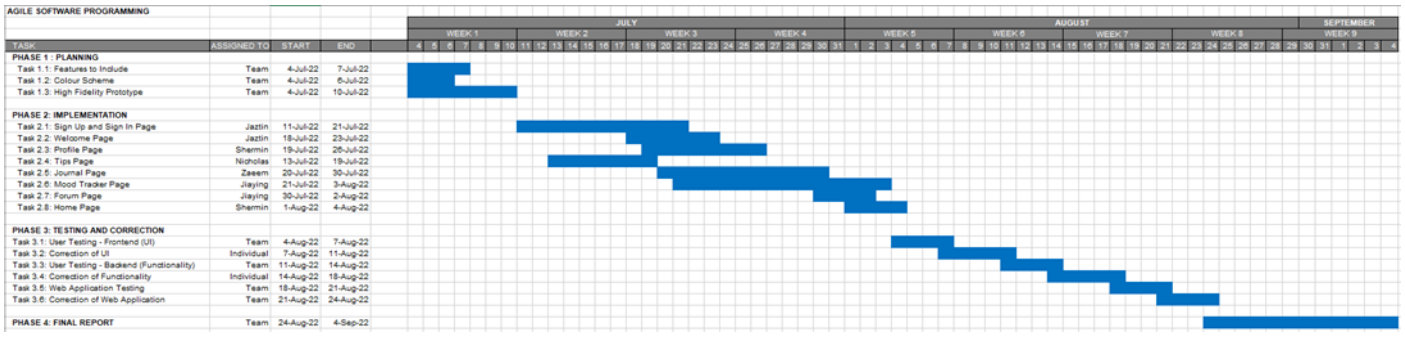
Kanban framework was used in our development stage. Given that we have done a substantial amount of background work, our planning and estimation are already well established. Since we used the technique of User-Centred Design, Kanban will work well with the incremental and evolutionary changes that we are bound to have while developing our web application.

Kanban uses the visualisation of the flow of work, where all group members have deadlines and specific features to do for the web application. A Kanban board has been shared with all the members of the group, and the progress of each member is updated weekly on the board. This way, we will be able to visualise our workflow and progress clearly, especially when we limit work in progress (WIP). Each member is only given a maximum of 2 features for them to work on, which allows them to focus on completing the task using a shorter amount of time due to our limited time and resources.

Backlogs in Kanban show all the tasks that are needed to do for the web application, which is essentially a set of user stories that we want to achieve. It also shows which features it falls under so that the member will know which tasks they are in charge of. Since there are different priorities for the features, the one with the highest priorities will have to start the development early while the rest starts to build the backbone of each feature first. The priorities of the tasks are discussed with each member of the group at the start of the weekly meeting before we try to compile all our codes and make sure that the web application runs.

2.2 – Gantt Chart

Planning our project timeline plays an important role in keeping us on track and ensuring we do not fall behind on our work. This was done using a Gantt Chart which is updated weekly and at every group meeting.



The Gantt Chart shows our team's progress throughout the next 2 months where we finalised our design, implemented the web application, and did user testing on all the backend and frontend of the web application.

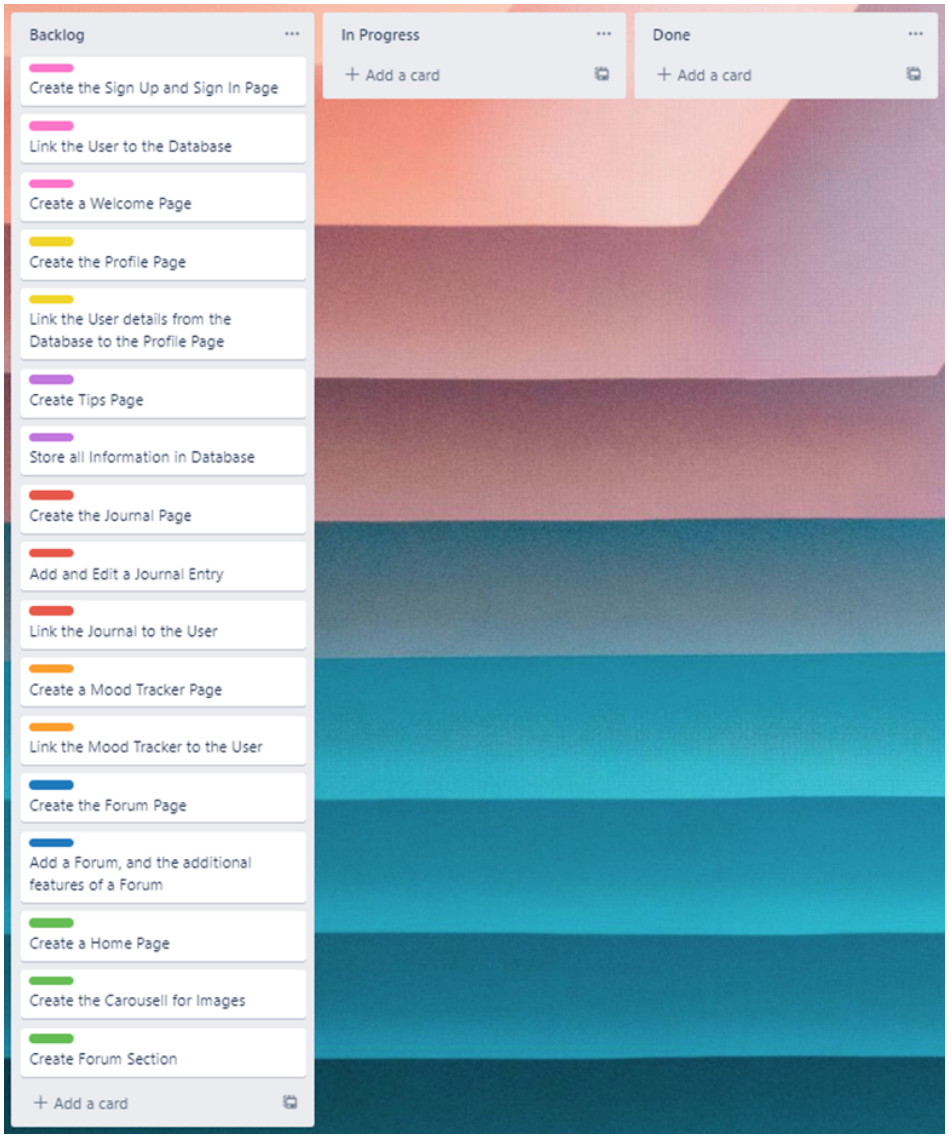
2.3 – Kanban Board

The Kanban Board was grouped into 3 main categories, namely Backlog, Progress and Completed. Each week, we will assign the tasks to be completed for that week, which we will move from Backlog to Progress once we start on the task. Next, by the end of the week, we should have the task moved to Completed before each meeting.

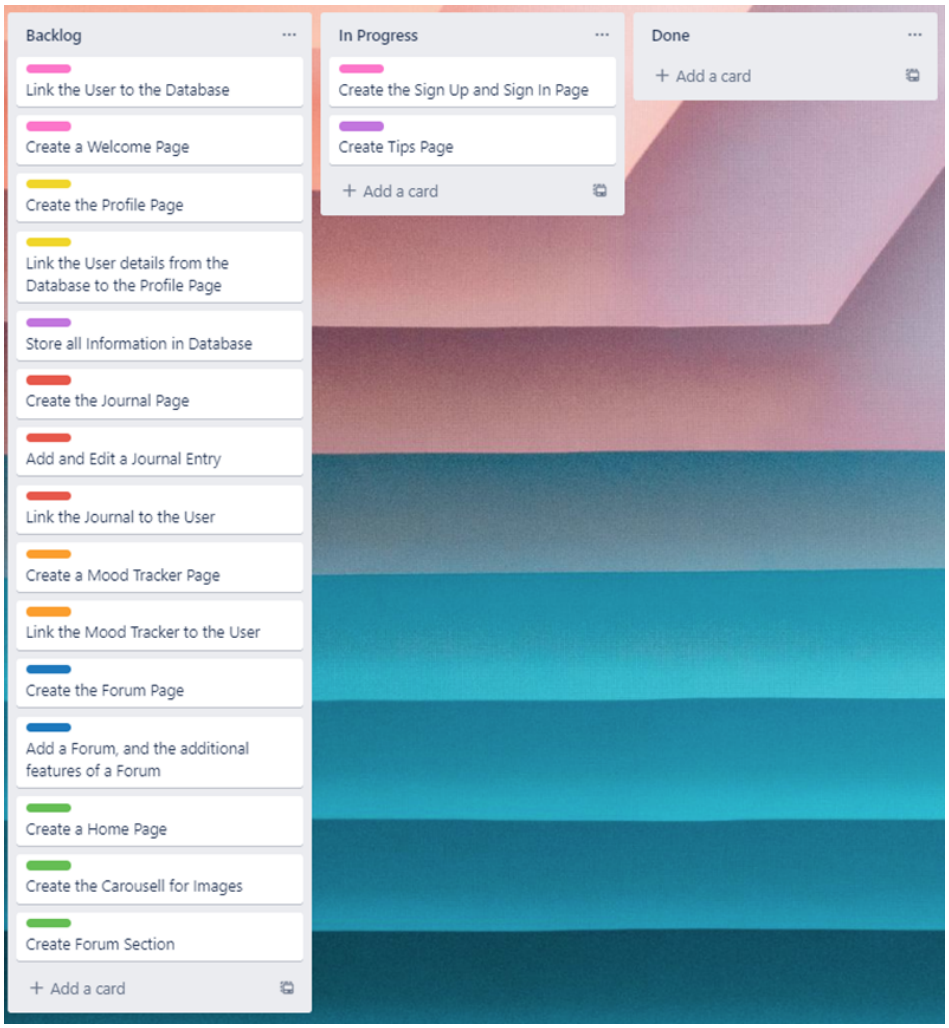
During each meeting, our progress on the web application will be checked in with one another, before we compiled all our codes in GitHub to try and run. If errors were to occur, we will fix the error on the spot as a team. When we reach the end of the meeting, tasks will be assigned again for the week. This cycle continues till we finish our web application.

The Backlog at the start of the project consists of tasks grouped according to the pages available in the web application, which includes the Home Page, Mood Tracker Page, Journal Page, Tips Page, Sign-Up Page and so on. Each member of the team is only assigned a maximum of 2 pages that they have to develop in the next 2 months.

We started our development process on the 11 of July, after finishing planning and finalising our high-fidelity prototype on Figma. The pages were done based on priorities as some pages needed to be completed before the other pages could be done. This was discussed during our planning process, and deadlines were given to each member based on the pages. This is to make sure that we can work at an efficient pace while making sure that our web application will work when put together.



14 July:

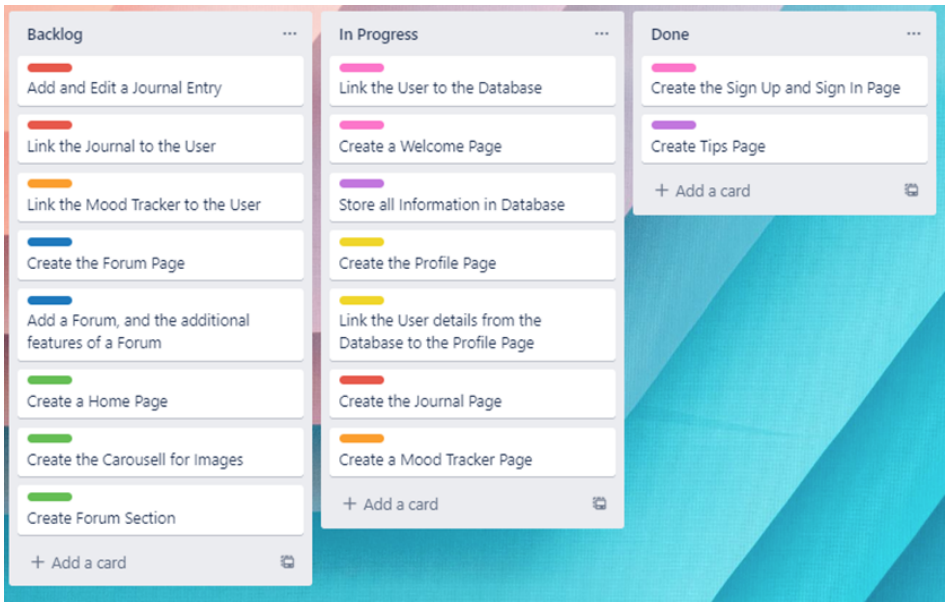


The Sign-up and Sign-In pages had to be done first. This is because our web application requires users to be logged in to use the features in the web application to the fullest. Since all our features are catered to unique users and differ for different users, we needed the sign-up page to be done.

The Tips page is not targeted to each unique user but instead applies to all users. Thus, the user was not needed when implementing the tips page.

We were on track with our proposed timeline, with the sign-in and tips page to be started.

21 July:



Once the Sign-up, Sign-in and Tips page was done with the frontend, it was time to start with the backend of the pages. The different users created using the sign-up page have to be linked to the database and linked back to the Sign-in page so that existing users will be able to sign in and access their data.

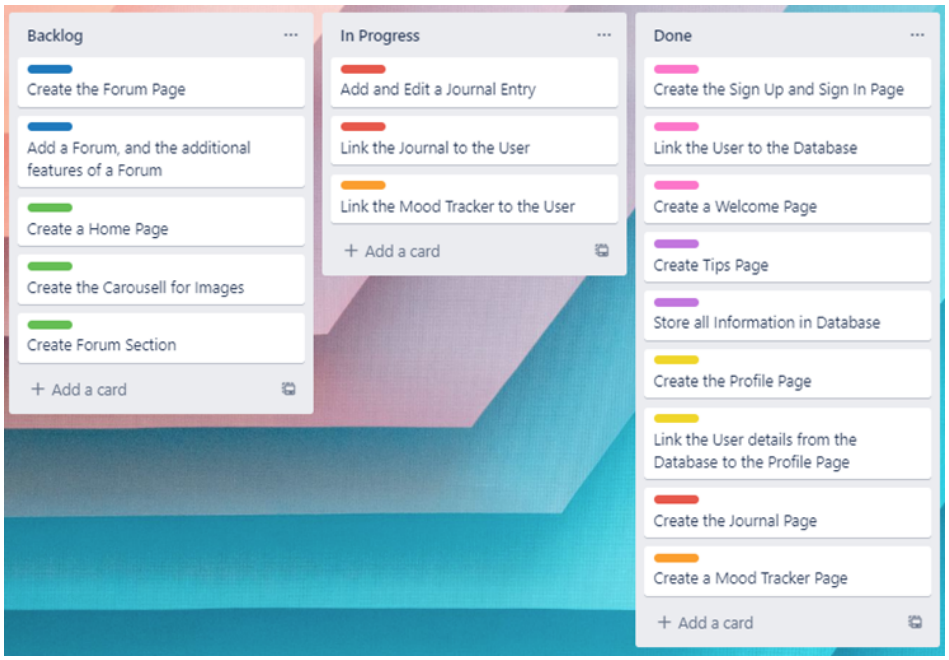
The information on the Tips page is also stored on the database, for easy access. This way, users can click on the tips which will bring them to external sources, where we took the tips from. This is preferred as external sources may change their website, and we want to have it linked to our page immediately.

A welcome page was deemed necessary after creating the Sign-up and Sign-in page so that users will have a start-up page for our web application instead of having the sign-up page as their first look. This was then implemented immediately after we were done with the Sign-up and Sign-in page.

Since our database was going to be able to store the user details, we got started with the frontend of the Profile page. Even though the backend of the Sign-up page was not done, the frontend of the Profile can be done first. It can then link to the database once the backend of the Sign-up page is done. This is to make sure that we do not fall behind on our schedule.

The Journal and Mood Tracker page also follows the unique users, and with the Sign-up page almost done, we got started with the frontend of those pages as well.

28 July:

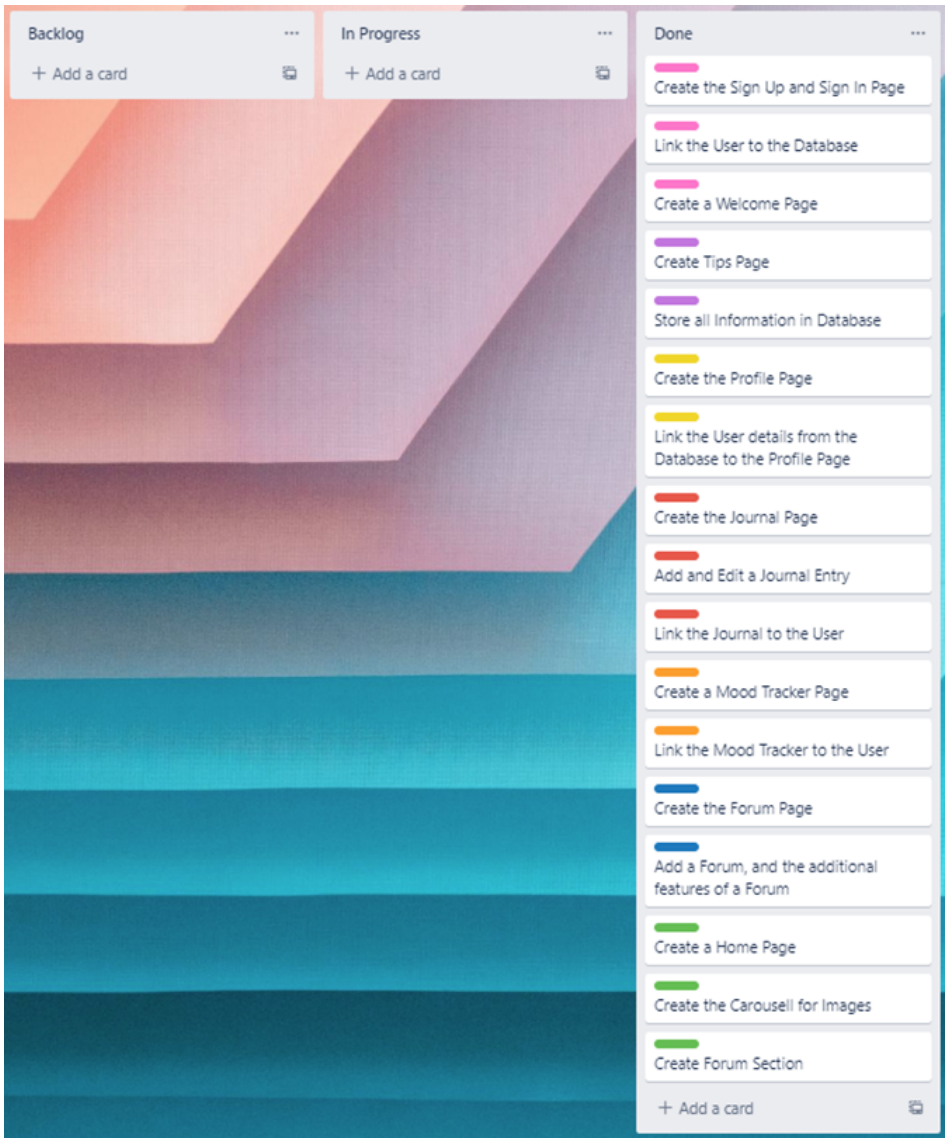


The database was able to store our user details by this time. Once this was done, our pages could slowly be built up and linked to the database. Following this, we started the journal page where each user can add their journal entry and link it to the unique user. The Mood Tracker page can also start to link to the unique user in the database. This took more time than the frontend of the pages since we had to make sure that each user has a different set of journal entries and different mood trackers from one another.

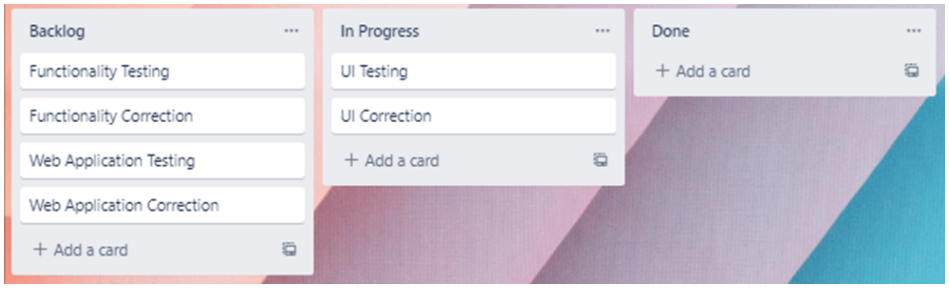
Furthermore, we started on the Forum Page once the Mood Tracker was done since it was implemented by the same person. The Forum page has almost the same concept as the Journal page, other than having the forum content similar to all the users. However, the unique user can add replies, like or share the forum, which had to be implemented after the Forum page was done.

The Home page was implemented on its own as it does not need access to the unique users' details. The carousel for the images from the Tips page was saved as assets instead for easy access. The frontend of the home page was done quickly since it did not need to wait for the other pages to be done to continue. However, the home page needed access to the Forum page as we planned to display popular forums on the home page. Thus, once the forum page is done, the home page follows along as the frontend of the home page is already done.

4 August:



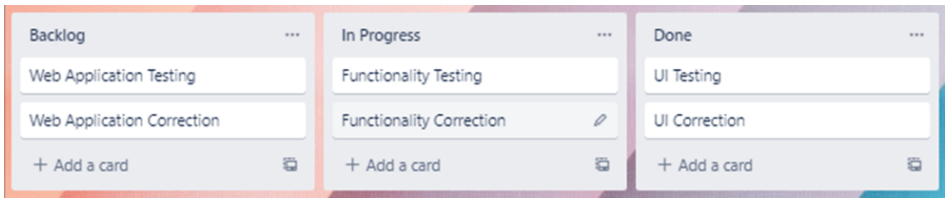
We managed to finish all the tasks allocated to us by the 4th of August, and compiled all the code together when we had our last meeting for the implementation of the web application. Following the completion of our web application, we had to start with the user testing of it.



The user testing was broken down into 3 main categories, namely the Frontend (UI), the Backend (Functionality) and the web application as a whole. The testing took 3 days while the correction took 4 days. Each category took 1 week.

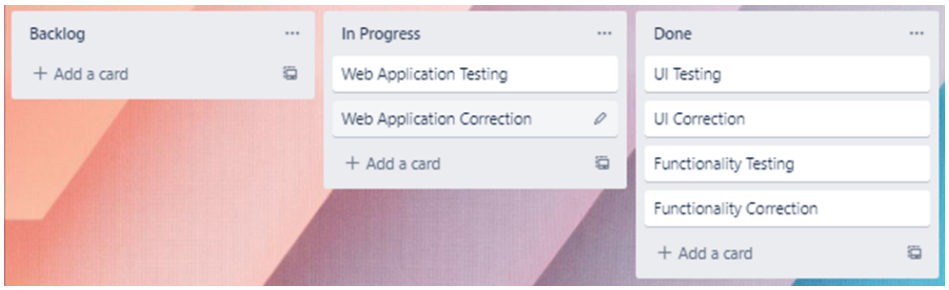
The first week, we did our Frontend testing, where we had to align the navigation bar, our content and our footer for the same look throughout the different pages. We noted down all the differences the pages had, and took the next few days to sort it out, making our web application have the same look for every page.

11 August:



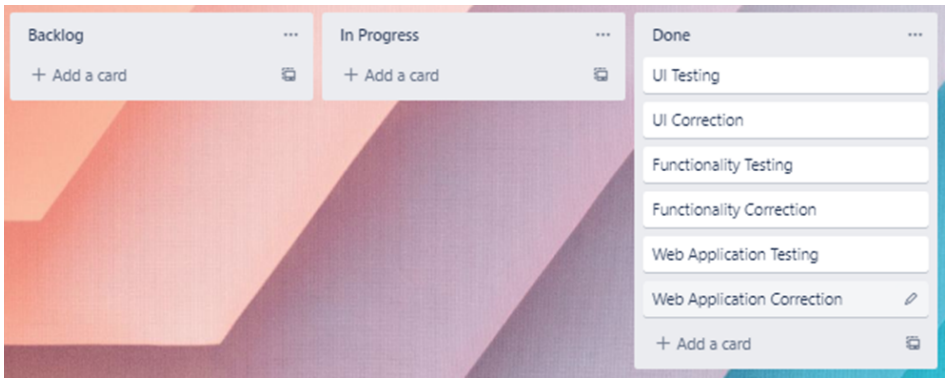
For the second week, we did Backend Testing, where we tested all the functionalities that we have implemented in each page. The functions that did not work according to the plan were noted down and passed back to the original member who was working on that page. We then corrected all the mistakes, and tested again to make sure all the functions run well.

18 August:



For the last week, we tested the web application as a whole and made sure that the web application ran smoothly with the transition between pages. We tested the whole process of the web application and noted down any errors that come along with it. We then took time to correct it, before compiling the final product. Once we made sure that the whole web application served its purpose, and ran smoothly, we reached the end of our Implementation stage.

25 August:



By the 25th of August, we were done with our mental health self-help web application.

Chapter 3: Prototyping and Iteration

3.1 - Prototyping

Before we got started on the development of the web application, we had first come up with 3 prototypes, which were the low, medium and high-fidelity wireframes. The low-fidelity wireframe was hand drawn and was a rough concept of what we aimed to create. The medium-fidelity wireframe looked to reinforce the main features of the web application and have a clear representation of it so that it could be used to get feedback from our potential users. Lastly, the high-fidelity wireframed aimed to finalize the design of the web application that we were going to develop. Both medium and high-fidelity wireframes were created using Figma, a collaborative web application for interface design. These prototypes were all documented in the midterm report and are included in the appendix below.

Before we did up the high-fidelity wireframe, we had a meeting with some of the potential users in order to gain feedback based on the medium-fidelity prototype. The topics that were covered were as follows:

- The navigation of the web application
- How user-friendly is the web application
- Possible improvements that can be made

By using Figma, we were also able to make use of the feature that allowed us to create animated prototypes. This meant that by using that feature in Figma, we were able to simulate the navigation through both the medium and high-fidelity prototypes as if they were the actual web applications that were created. Thus, this allowed us to structure the navigation of the web application more effectively and also make the process of demonstrating the navigation process to the user during the meetings much easier.

Generally, the users felt that the navigation through the web application was quite simple and intuitive, and the information that was displayed was also very easily processed.

However, they also gave feedback that some of the buttons on the forum page seemed to be rather redundant. In the medium-fidelity wireframe, for each forum post, there was a button to access the post itself and another one to add a new forum post. This was changed when we did the high-fidelity wireframe so that there was now only one button to add new forum posts and clicking on each individual post would access that particular post. They had also suggested a change in the colour scheme for the web application which was shown in blue for the medium-fidelity prototype as they felt it was a little too overpowering having too much of the same colour. This too was changed in the high-fidelity wireframe to beige and green which we felt suited the web application better.

Further user testing conducted using the high-fidelity wireframe showed that they were rather satisfied with the design of the web application and its navigation. So, we were able to move on to beginning the development process.

From all of the feedback we received to improve our web application, we had to ultimately take into consideration which specifications we wanted to implement, so the following points had to be taken into account:

Our competency as programmers

Since our team members had different levels with regards to developing a web application, we decided to choose a framework that most of the members were familiar with, which in this case was Bootstrap. Thus, members who had more prior experience with it were able to help those with lesser experience. Additionally, we had to also take into consideration the complexity of the database server system that we planned on implementing to be able to save and retrieve data.

Runway given

To complete this project, time had to be allocated to:

- Learning and familiarization of the Bootstrap framework
- Learning and familiarization of Firebase
- Debugging and resolving issues
- User testing
- Report writing

In order to prevent over-committing, 10 functional specifications were chosen by the team:

1. Allows potential users to sign up to use the web application (frontend and backend logic)
2. Allows current users to sign in to the web application (frontend and backend logic)
3. Allows users to add their mood using the mood tracker (frontend and backend logic)
4. Allows users to add and view journal entries (frontend and backend logic)
5. Allows users to edit journal entries (frontend and backend logic)
6. Allows users to add and view forum posts (frontend and backend logic)
7. Allows users to reply and like forum posts (frontend and backend logic)
8. Allows users to view tips added to the web application (frontend logic)
9. Allows users to edit their information on the profile page (frontend and backend logic)
10. Allows users to navigate between the different pages using the navigation bar (frontend logic)

Technical specifications:

1. Platform to be run on: Chrome, Firefox, Microsoft Edge
2. Framework: Bootstrap
3. Language: HTML
4. Backend: Firebase

3.2 - Unit Testing of All Functionalities in the Web Application

Our team chose to log all of our unit testing results (pass test cases) under this section. We chose failed test case logs as we felt that they were not necessary.

Test name: Unit Testing of Sign-Up Functionality

Test case #	Scenario	Steps	Expected Output	Actual Output	Pass/Fail
1.1	That users are able to sign up on the web app	1. Enter in all the necessary fields for the sign up 2. Click on submit to sign up for the web application	Once all fields have been filled up and submitted, the user will be directed to the sign-in page to sign in to the web application	As expected	Pass
1.2	That users are not able to enter blank fields when signing up	1. Leave all of the fields blank when trying to sign up	All blank fields show up red and the signing up process is unable to proceed	As expected	Pass
1.3	That invalid entries to the fields are rejected	1. Type in an invalid field entry	Error message showing field with the invalid entry will be displayed	As expected	Pass

Test name: Unit Testing of Sign-In Functionality

Test case #	Scenario	Steps	Expected Output	Actual Output	Pass/Fail
2.1	That users are able to sign in to the web app	1 Click on the sign in button 2 Enter a valid email to the email field 3 Enter the valid password to the password field 4 Click sign in	User will be directed to the profile page once signed into the web application	As expected	Pass
2.2	that users are able to log out of the web app	1 Click logout in the profile page	User will be directed to the welcome page of the web application	As expected	Pass
2.3	That users are not able to enter blank fields when signing in	1. Click sign in while leaving the required fields blank	User is not able to sign in to the web application	As expected	Pass
2.4	That if the user's information does not exist in Firebase, the user is not able to sign in to the web app	1. Enter invalid email to the email field 2. Enter the invalid password to the password field	User is not able to sign in to the web application	As expected	Pass
2.5	That if the user enters a valid email but has an invalid password, the user is not able to sign in to the web app	1. Enter a valid email to the email field 2. Enter the invalid password to the password field	User is not able to sign in to the web application	As expected	Pass

Test name: Unit Testing of Mood Tracker Functionality

Test case #	Scenario	Steps	Expected Output	Actual Output	Pass/Fail
3.1	That user is only able to add a mood to track for the current day	1. Click on the current date in the mood tracker 2. Click on the mood that the user is feeling	The date in the mood tracker is filled in with the colour representing the mood that was selected	As expected	Pass
3.2	That user is able to change the mood that was tracked for the current day	1. Click on the current date in the mood tracker 2. Click on the new mood that the user is feeling	The date in the mood tracker is filled in with the new colour of the updated mood once the page is refreshed	As expected	Pass

Test name: Unit Testing of Journal Page

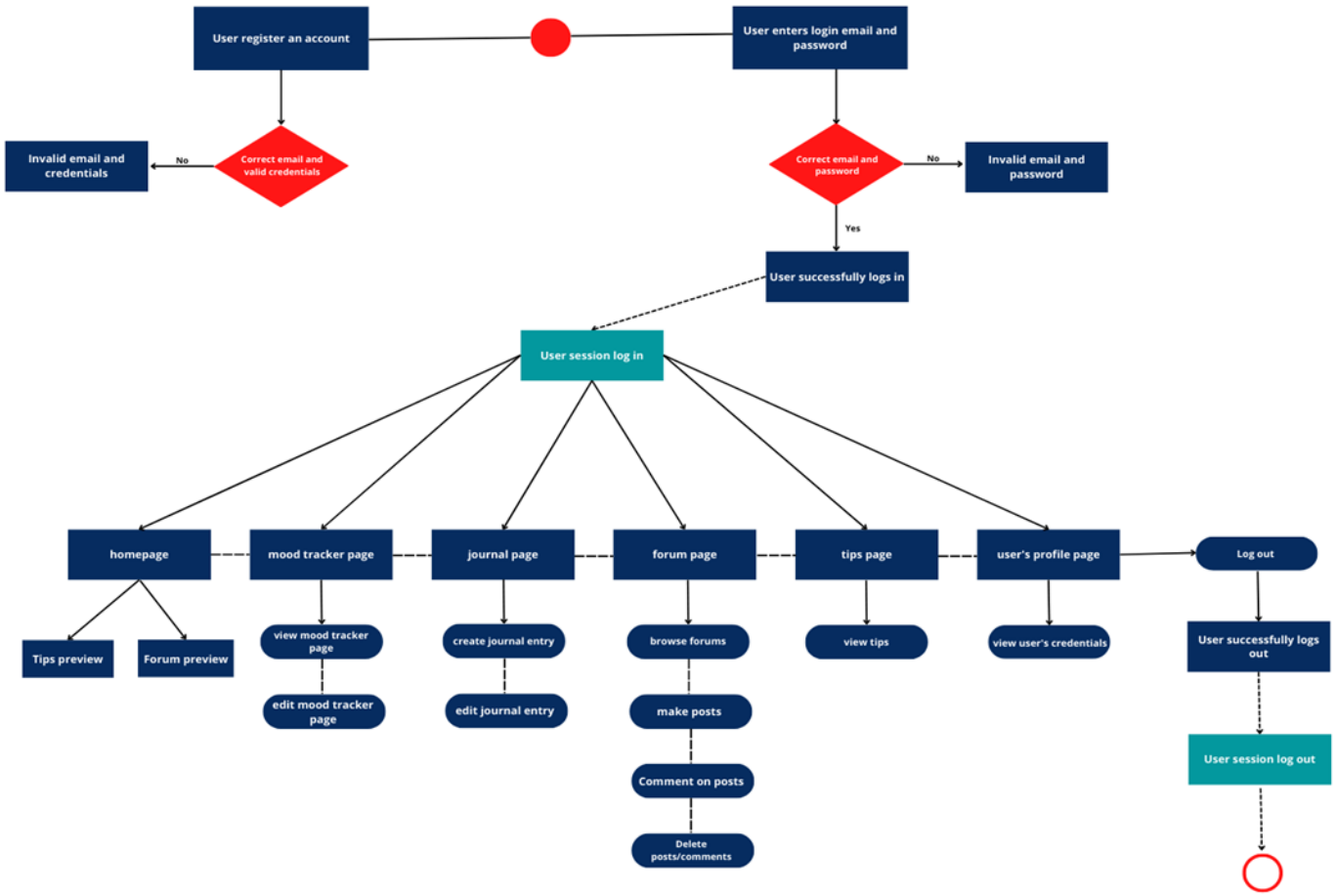
Test case #	Scenario	Steps	Expected Output	Actual Output	Pass/Fail
4.1	That user is able to add in a new journal entry	1. click on the “Add journal” entry button 2. Fill in the textbox with the content for the journal 3. Click add to publish the new journal	New journal entry is published to the journal page in the web app	As expected	Pass
4.2	That user is able to edit a journal that has already been published	1. Click on an existing journal 2. Click on the edit to begin editing the content of the journal 3. Click on done once the editing of the journal is complete	The selected journal entry will display as a pop-up when selected Once edited, the new content will be displayed for that entry on the journal page in the web app	As expected	Pass
4.3	That user is able to use the drop-down bar to select the journal entry they want to view/edit	1. Click on the drop down bar 2. Click on the number corresponding to the day the journal was published	The drop-down bar will show all of the available days for the journal entries Clicking on the number will display the selected journal entry as a pop up	As expected	Pass

Test name: Unit Testing of Forum Page

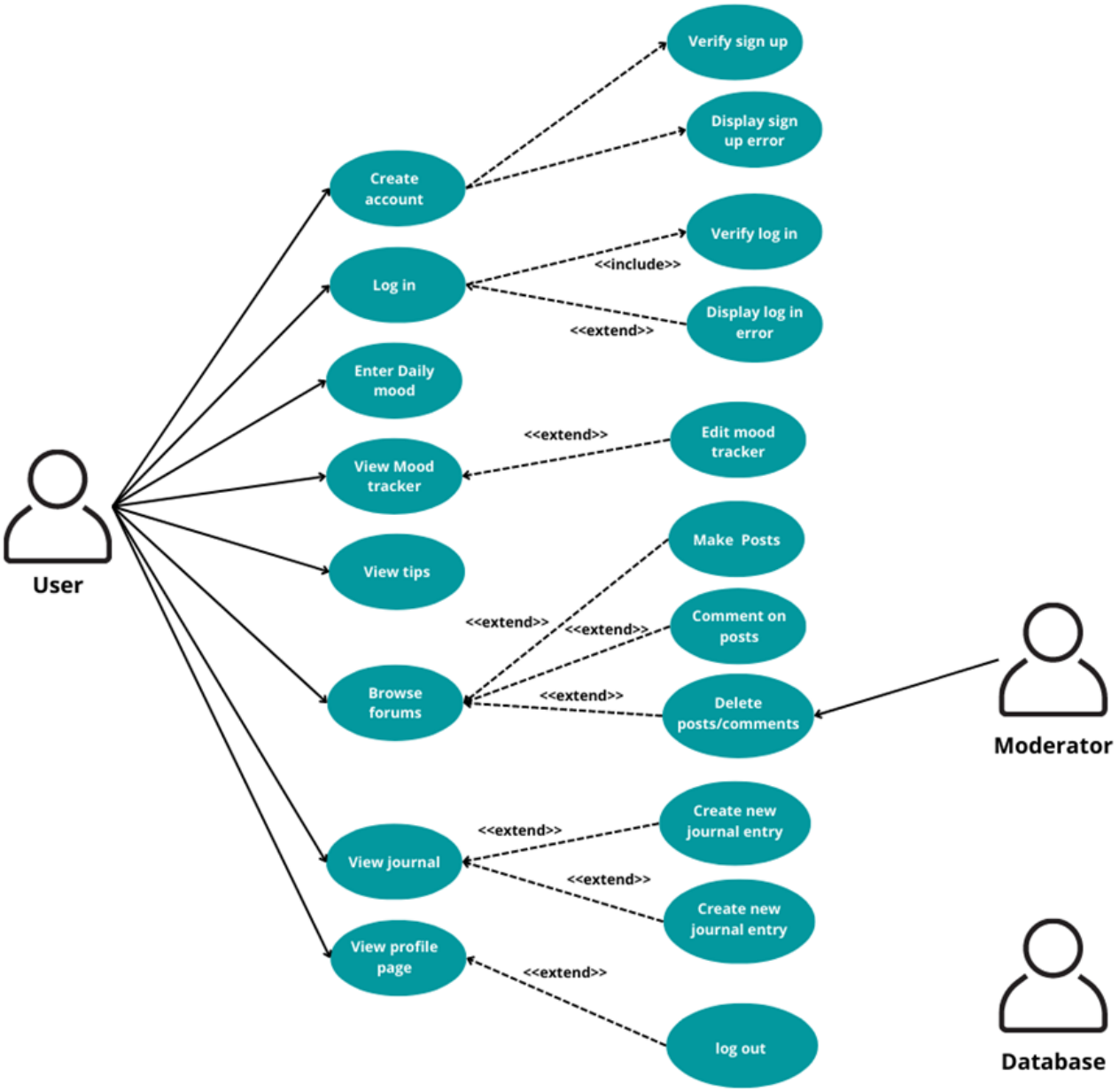
Test case #	Scenario	Steps	Expected Output	Actual Output	Pass/Fail
5.1	That user is able to add in a new forum post	1. Click on the add forum entry button 2. Fill in the textbox with the title and content for the post 3. Click create to publish the new post	New forum post is published to the forum page in the web app	As expected	Pass
5.2	That user is able to open up the selected forum post	1. Click on the forum post to open it up	User will be brought to a new page that displays that particular forum post	As expected	Pass
5.3	That user is able to like the forum post	1. Click on the heart button to like the post	When the post is liked, the number next to the heart will go up, showing the total amount of likes	As expected	Pass
5.4	That user is able to reply to the forum post	1 Enter in the reply to the textbox under the post 2 Click the add button to add a reply to that forum post	The reply will be added to the forum post The number above the replies will go up, showing the current number of replies	As expected	Pass

Chapter 4: Design Specification

4.1 - Module Design



4.2 - Use Case Diagram



4.3 - User Case Descriptions

Use Case	Access Home page	
Actors	User	
Precondition	Users must have a valid account and be logged in.	
End Result	Users can view the homepage and access tips and forum preview	
Main Flow	Step	Action
	1	User access the homepage.
	2	Users can click on the tips and forum preview links.
	3	Redirect to tips and forum page

Use Case	Log daily mood and view them.	
Actors	User	
Precondition	Users must have a valid account and be logged in.	
End Result	Users successfully log their daily moods and can view them.	
Main Flow	Step	Action
	1	User access Mood Tracker page.
	2	Users input their mood for the current day.
	3	Application verifies and updates the database with the user's mood.
	4	Mood Tracker page refreshes with updated mood for the day.

Use Case	Enter a new entry into the Journal and view it.	
Actors	User	
Precondition	Users must have a valid account and be logged in.	
End Result	User successfully creates a journal entry and can view them.	
Main Flow	Step	Action
	1	User access Journal page.
	2	User writes an entry in the Journal.
	3	Application verifies and updates the database with the user’s journal entry.
	4	Journal page refreshes displaying updated journal entries.

Use Case	Browse and Post in the Forums	
Actors	User	
Precondition	Users must have a valid account and be logged in.	
End Result	Users successfully create a forum post and can view and comment on the post.	
Main Flow	Step	Action
	1	User access Forum page.
	2	User creates a post.
	3	Application verifies and updates the database with the user’s forum post.
	4	Forum page refreshes displaying the user's forum post.

Use Case	View Mental Health Tips	
Actors	User	
Precondition	Users must have a valid account and be logged in.	
End Result	Users successfully view the Tips page and can click on links.	
Main Flow	Step	Action
	1	User access Tips page.
	2	User clicks the link to view more.
	3	User is redirected to the specific tip page.

Use Case	View Profile Page	
Actors	User	
Precondition	Users must have a valid account and be logged in.	
End Result	Users successfully view the profile page and can view the user’s credentials	
Main Flow	Step	Action
	1	User access profile page.
	2	User can click logout button
	3	User is able to log out its profile

Chapter 5: System Development

5.1 - Technical Responsibility

As our team decided to utilise Kanban as a framework for the Agile methodology, we first created the user stories to visually represent on the Kanban board. We then split our responsibilities depending on each feature of the Mood.ly web application.

Application Feature	**Member In-charge**
Homepage	Shermin
Sign-up/Sign-in (Front end)	Jaztin
Sign-up/Sign-in (Back end)	Jaztin
Mood Tracker	Jiaying
Forums	Jiaying
Tips	Nicholas
Profile	Shermin
Journal	Zaeem
Technical Feature	**Member In-charge**
Firebase Database	Zaeem
Firebase Auth	Jaztin

5.2 - Code Development

5.2.1 - Language Justification

The team decided to use node.js to build the Mood.ly web application using JavaScript. We use Express as the web application framework and EJS as the templating engine. We chose to use Express and Ejs as we were familiar with them and felt that they also fit what we wanted to build with the Mood.ly web application. We chose to use Node.js, Express and EJS as they are suitable for building dynamic applications and the group as a whole has a bit of experience with it.

We also use Bootstrap for our CSS framework as it allows us to build responsive interfaces.

5.2.2 - Library Justification

The following are the dependencies used for the Mood.ly web application:

Dependencies	Version	Purpose
body-parser	1.20.0	Used for parsing incoming request bodies.
cookie-parser	1.4.6	Used to parse and populate cookies.
csurf	1.11.0	Middleware for cross-site request forgery protection.
ejs	3.1.8	Used for templating HTML pages.
express	4.18.1	Used to manage servers and routes.
firebase	9.9.1	Used to connect to our firebase real-time database.
firebase-admin	11.0.0	Used to connect to firebase authentication.
node-time-ago	1.0.0	Used to retrieve time passed.

5.2.3 - File Structure

The file structure of this project is split up into 3 separate folders which are the public, routes and views folders. The “index.js” file which contains the required functions for the application is placed in the root of the whole project.

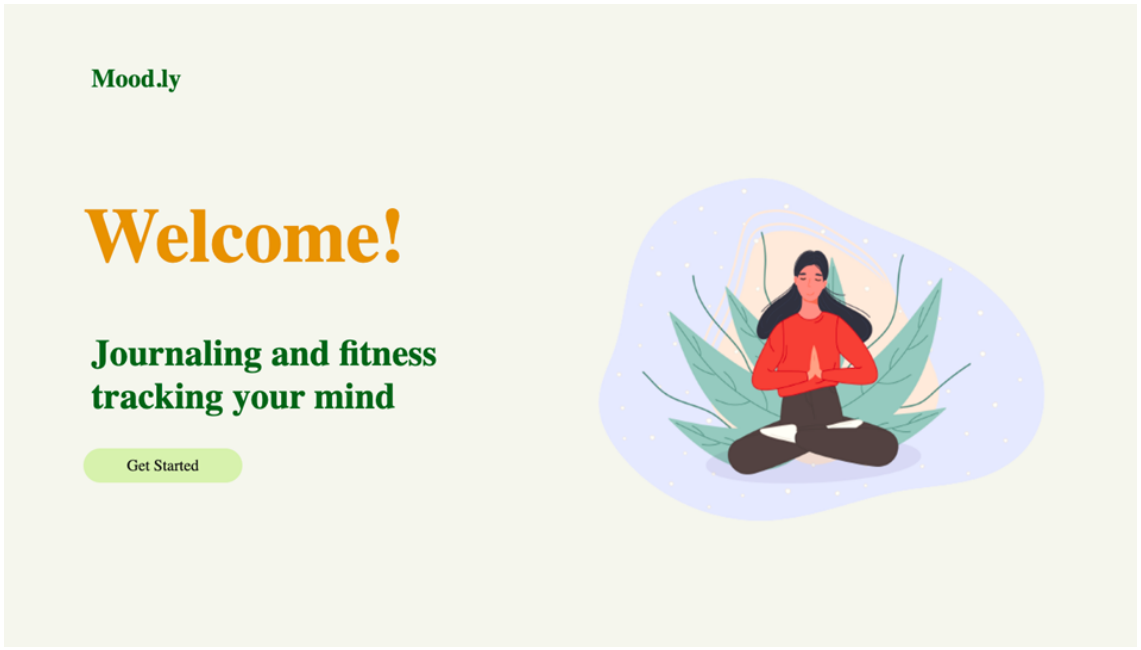
/public	Folder containing images, scripts and stylesheets which may be used by HTML
/assets	Folder containing images for the carousel in the homepage.
/css	Folder containing CSS for all pages in the project.
/js	Folder containing js scripts for individual
/routes	Folder containing the main file which handles navigation between pages.
/util	Folder containing a date utility class which is used for date logic in main.js.
/main.js	The main javascript file containing all the page navigation logic.
/views	Folder containing all the HTML pages.

5.3 - Code Description

5.3.1 - Welcome Page

The welcome page is the first landing page of the application to greet the users and displays the tagline as well as the app's logo. It also has a button to get started and direct the users to sign up and sign in.

```html
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="/css/welcomepage.css">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.mi


</head>

<body>

    <div class="welcome_image">
        <img src="/assets/welcome_page.png">
    </div>
    <div class="logo">
        <h1>Mood.ly</h1>
    </div>


    <div class="tagline">
        <h1>Welcome!</h1>
        <h2>Journaling and fitness tracking your mind</h2>
    </div>

    <form id="sign-in-page" action="/signIn" method="POST">
    <button class="get_started">Get Started</button>
    </form>

</body>
</html>
```



5.3.2 - Sign Up and Sign In Page

The sign-in and sign-up are on one page inside the signIn.html. The user can either choose the sign-in button or the sign-up button. If the user has not created an account yet, the user can click the sign-up button and fill in the input fields needed. Otherwise, the user can click the sign-in button and enter the valid email and password. Here is the HTML code for sign-in and sign-up located in the signIn.html

```
<div class="container">
  <div class="overlay" id="overlay">
    <div class="sign-in" id="sign-in">
      <h1>Welcome Back!</h1>
      <p>To keep connected with us please login with your personal info</p>
      <button class="switch-button" id="slide-right-button">Sign In</button>
    </div>
    <div class="sign-up" id="sign-up">
      <h1>Hello, Friend!</h1>
      <p>Enter your personal details and start a journey with us</p>
      <button class="switch-button" id="slide-left-button">Sign Up</button>
    </div>
  </div>

  <div class="form">
    <div class="sign-in" id="sign-in-info">
      <h1>Sign In</h1>

      <form id="sign-in-form" action="/sign_in" method="POST">
        <input type="email" placeholder="Email" id="email_sign_in" name="email"/>
        <input type="password" placeholder="Password" id="password_sign_in" name="password"/>
        <% if(locals.err) { %>
          <div class = "alert alert-danger" role="alert"><%=err %></div>
        <% } %>

        <button class="control-button in">Sign In</button>
      </form>
    </div>
```

```
<div class="sign-up" id="sign-up-info">
  <form id="sign-up-form" action="/sign_up" method="POST">

    <div class="sign_up_details" class="first_step">

      <h1>Create Account</h1>

      <% if(locals.err) { %>
        <div class = "alert alert-danger" role="alert"><%=err %></div>
      <% } %>

      <input type="text" placeholder="Full Name" id="name_sign_up" name="name" oninput="this.className = ''"/>
      <input type="email" placeholder="Email" id="email_sign_up" name="email" oninput="this.className = ''"/>
      <input type="password" placeholder="Password" id="password_sign_up" name="password" oninput="this.className = ''"/>
    </div>

    <div class="sign_up_details" id="second_step">
      <input type="text" placeholder="Username" id="user_name" name="user_name" oninput="this.className = ''"/>
      <input type="date" placeholder="Birthday" id="birthday" name="birthday" oninput="this.className = ''"/>
      <input type="radio" id="gender_male" name="gender" value="male" oninput="this.className = ''">
      <label for="male">male</label>
      <input type="radio" id="gender_female" name="gender" value="female" oninput="this.className = ''">
      <label for="female">female</label>

      <input type="tel" placeholder="Phone Number (Optional)" id="phone_number" name="phone_number" oninput="this.className = ''"
      <small></small>
      <input type="text" placeholder="Educational Level (Optional)" id="educational_level" name="educational_level" oninput="this.
      <input type="text" placeholder="Name of School (Optional)" id="school" name="school" oninput="this.className = ''"/>
    </div>

    <div style="overflow:auto;">
      <div style="float:right;">
        <button type="button" id="prevBtn" onclick="nextPrev(-1)">Previous</button>
        <button type="button" id="nextBtn" onclick="nextPrev(1)">Next</button>
      </div>
    </div>

    <div style="text-align:center;margin-top:40px;">
      <span class="step"></span>
      <span class="step"></span>
    </div>
  </form>
</div>
```

Using the firebase authentication as a backend, we are able to authenticate the email and password when the user signs in. With the help of firebase documentation, we have used the signInWithEmailAndPassword with the email and password as a parameter. The below code is to authenticate the user's email and password. If the user successfully signed in then it will create a session log-in for the signed user. If it's an invalid email or password then it will return an error.

```
<script>
    window.addEventListener("DOMContentLoaded", () => {
    var firebaseConfig = {
    apiKey: "AIzaSyB62-2cMjYCbJySOj6uFXNSQkmu1y7t0M4",
    authDomain: "moodly-9fa79.firebaseapp.com",
    databaseURL: "https://moodly-9fa79-default-rtdb.asia-southeast1.firebasedatabase.app",
    projectId: "moodly-9fa79",
    storageBucket: "moodly-9fa79.appspot.com",
    messagingSenderId: "216325732946",
    appId: "1:216325732946:web:e1f2714927b5b281267d8b"
  };

    firebase.initializeApp(firebaseConfig);

    firebase.auth().setPersistence(firebase.auth.Auth.Persistence.NONE);

    document
      .getElementById("sign-in-form")
      .addEventListener("submit", (event) => {
        event.preventDefault();
        const login = event.target.email.value;
        const password = event.target.password.value;

        firebase
          .auth()
          .signInWithEmailAndPassword(login, password)
          .then(({ user }) => {
            console.log(user);
            return user.getIdToken().then((idToken) => {
              return fetch("/sessionLogin", {
                method: "POST",
                headers: {
                  Accept: "application/json",
                  "Content-Type": "application/json",
                  "CSRF-Token": Cookies.get("XSRF-TOKEN"),
                },
                body: JSON.stringify({ idToken }),
              });
            });
          })
```

```
148                     },
149                     body: JSON.stringify({ idToken }),
150                   });
151                 });
152               })
153
154               .then(() => {
155                 return firebase.auth().signOut();
156               })
157               .then(() => {
158                 window.location.assign("/profile");
159               });
160
161             return false;
162           });
163         });
164         </script>
165
166   <script src="/js/signIn.js"></script>
167
168   </body>
169   </html>
170
```

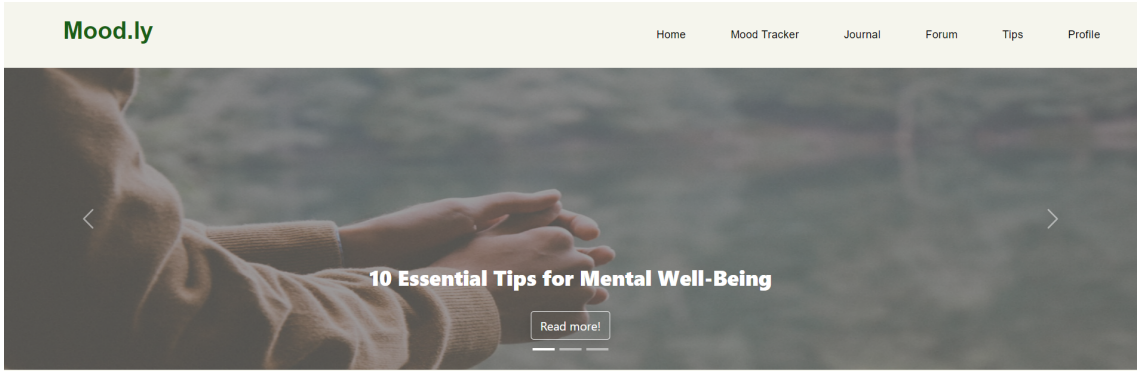
For signing up, with the firebase real-time database, the user can fill in the input fields and the user's credentials will be saved in the database. Some of the input fields are optional, therefore it could be empty or null in the database. The below code can be found in the main.js.

```
//saving sign-up to database

app.post('/sign_up', function (req, res) {

  admin
      .auth()
      .createUser({
        "email" : req.body.email,
        "password" : req.body.password
      })
      .then((userData) => {

       signUpRef.push().set({
          "userID": userData.uid,
          "name" : req.body.name,
          "email" : req.body.email,
          "password" : req.body.password,
          "username" : req.body.user_name,
          "birthday" : req.body.birthday,
          "gender" : req.body.gender,
          "phone_number" : req.body.phone_number,
          "educational_level" : req.body.educational_level,
          "school" : req.body.school

      });
      res.render("signIn.html");
    })
    .catch((error) => {
        // console.log('Error creating userr:',error);
        var errorCode = error.code;
        var err = error.message;
        console.log(errorCode)
        console.log(err)
        res.send({err: err });
    });

});
```

5.3.3 - Home Page

The Home Page displays a brief summary of what to expect from the other pages in the web application.

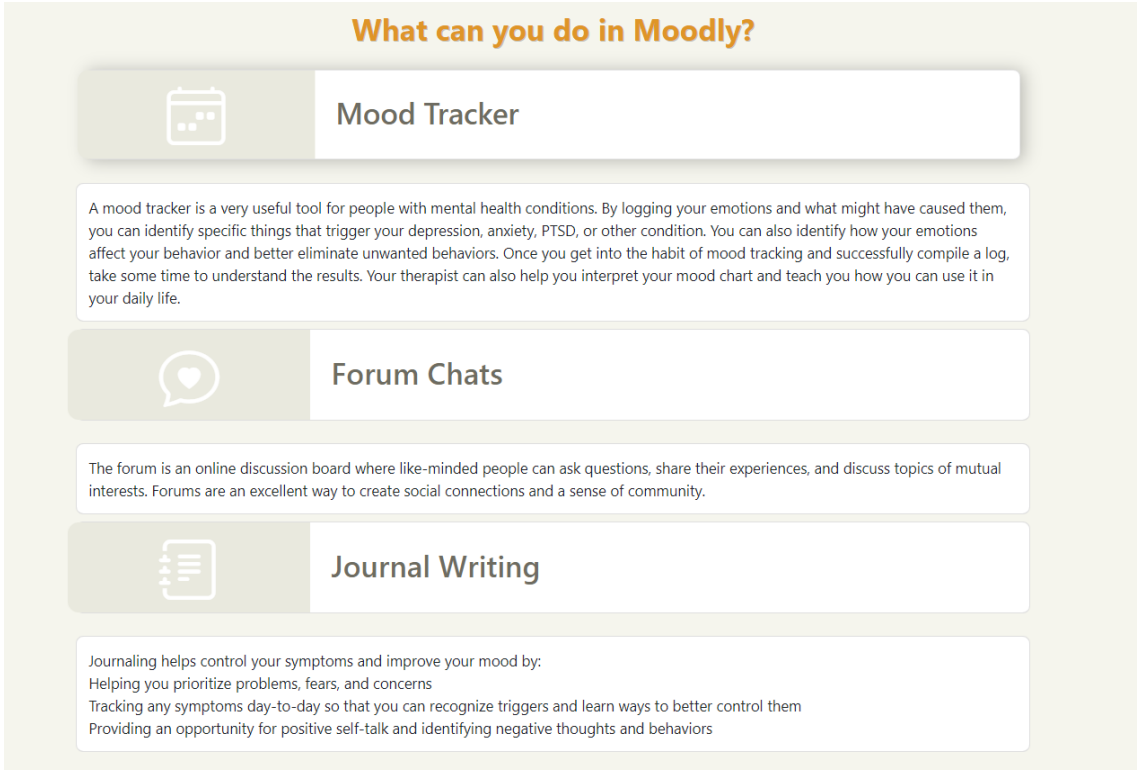
The image carousel found at the top of the page shows the title of the different tips that can be found on the Tips page. This way, users can have a glimpse of the different tips while they are on the home page.



This is done using the carousel component in Bootstrap. It cycles through a series of images, with the title of the different tips on the image. A “Read More” button is also displayed for easy navigation to external sources.

```
<div id="carouselExampleCaptions" class="carousel slide" data-bs-ride="carousel">
    <div class="carousel-indicators">
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="0" class="active"
            aria-current="true" aria-label="Slide 1"></button>
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="1" aria-label="Slide 2"></button>
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="2" aria-label="Slide 3"></button>
    </div>
    <div class="carousel-inner">
        <div class="carousel-item active" data-bs-interval="5000">
            <img class="d-block w-100"  src="/assets/carousel_1.jpg">
            <div class="carousel-caption  d-md-block">
                <h3>10 Essential Tips for Mental Well-Being</h3>
                <a class="btn text-white border-light shadow" href="https://www.healthhub.sg/live-healthy/1926/10-Essentials-for-Mental-Well-Being">Read more!</a>
            </div>
        </div>
```

The Home Page also shows the different features available in the web application, while giving a brief summary of what each feature can be used for. This way, users will be able to know which features they are more interested to use or to know more about the specific feature before using. Our main features include the mood tracker, forum and journal writing.



Following that, forums found on the Forum page will also be displayed so that users can have easy access to them, without having to go to the Forum page. This way, users can have a quick glance at what is happening on the forum page.

This was done using DataSnapshot from Firebase. A DataSnapshot contains data from the Database location, which is our Firebase. When we want to read data from our firebase, we can take a snapshot, and the data in the database will get copied. We then input it into our HTML using EJS and read those data. The data then gets displayed on the HTML pages.

```
app.get("/homepage", function (req, res) {
  var forumListRef = forumRef;
  var forumList = [];

  forumListRef.on('value', (data) => {
    data.forEach(function (snapshot) {
      forumList.push(snapshot.val());
    })
  });

  res.render("homepage.html", { forumItem: forumList });
});
```

```
<button type="submit" class="btn" id="forumId" name="forumId" value="<%= item.forumId%>">
    <div class="row align-items-center">
        <div class="col-md-8 mb-3 mb-sm-0 wrap-content" style="text-align: left;">
            <h4>
                <%= item.forumTitle%>
            </h4>
            <p class="text-sm"><span class="op-6">Posted</span>
                <span class="op-6">
                    by <%= item.username%></span>

                </a></p>
            <p class="text-sm op-5">
            <p class="text-black mr-2" href="#">
                <%= item.forumContent%>
            </p>
            </p>
        </div>
        <div class="col-md-4 op-7">
            <div class="row text-center op-7">
                <div class="col px-1"> <i class="ion-connection-bars icon-1x"></i> <span
                        class="d-block text-sm">
                        <%= item.numOfLikes%>
                            Votes
                    </span> </div>
                <div class="col px-1"> <i class="ion-ios-chatboxes-outline icon-1x"></i> <span
                        class="d-block text-sm">
                        <%= item.numOfReplies%>
                            Replies
                    </span> </div>
                <div class="col px-1"> <i class="ion-ios-eye-outline icon-1x"></i> <span
                        class="d-block text-sm">
                        <%= item.numOfViews%>
                            Views
                    </span> </div>
            </div>
        </div>
    </div>
</div>
```

5.3.4 - Journal Page

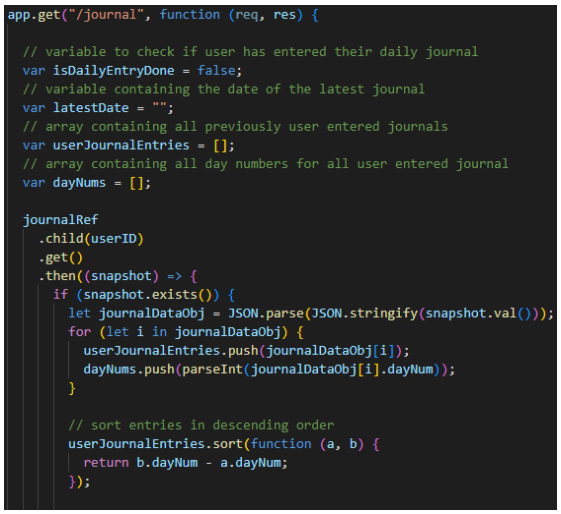
journal.html

This file handles the display for the Journal page. It contains the same navigation bar as other pages in the website. This page allows users to write down their personal feelings for each day and they can revisit them at their leisure. It also allows the user to update their journal entries.

Instead of a search function as we previously intended to implement, we implemented a dropdown of all Daily entries so that the user can access specific journals without having to scroll to find them.

/journal route

In the backend, there are 3 different routes for the journal. The first is the GET request for the journal page. When this route is called, the application will retrieve the user's journal entries from the firebase database, based on the user's ID which is set when the user first logs in.



```
app.get("/journal", function (req, res) {

  // variable to check if user has entered their daily journal
  var isDailyEntryDone = false;
  // variable containing the date of the latest journal
  var latestDate = "";
  // array containing all previously user entered journals
  var userJournalEntries = [];
  // array containing all day numbers for all user entered journal
  var dayNums = [];

  journalRef
    .child(userID)
    .get()
    .then((snapshot) => {
      if (snapshot.exists()) {
        let journalDataObj = JSON.parse(JSON.stringify(snapshot.val()));
        for (let i in journalDataObj) {
          userJournalEntries.push(journalDataObj[i]);
          dayNums.push(parseInt(journalDataObj[i].dayNum));
        }

        // sort entries in descending order
        userJournalEntries.sort(function (a, b) {
          return b.dayNum - a.dayNum;
        });
```

```
        // get latest date by getting object with the latest dayNum
        var maxDayNum = Math.max(...dayNums);
        for (let i in userJournalEntries) {
          if (userJournalEntries[i].dayNum === maxDayNum) {
            latestDate = userJournalEntries[i].dateAdded;
            break;
          }
        }

        // check if daily journal has been added for today,
        // by checking if the latest journal entry date is today
        isDailyEntryDone = dateUtil.isDateToday(latestDate);

        res.render("journal.html", {
          userJournalEntriesData: userJournalEntries,
          dailyEntryDone: isDailyEntryDone,
          latestDayNum: maxDayNum,
        });
      } else {
        console.log("No data available");
        res.render("journal.html", { latestDayNum: 0 });
      }
    })
    .catch((error) => {
      console.error(error);
      res.redirect("/");
    });
});
```

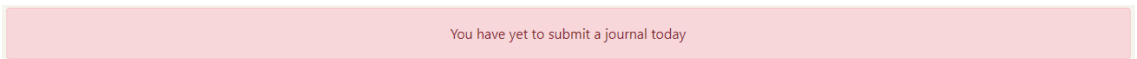
Once verified that the user has previously created journal entries, as evidenced when snapshot.exists() returns true, the application will first check if the user has done their daily journal entry and also retrieve the maximum day number from the data.

The application then renders the HTML and passes the user's journal entries, latest day number and the boolean as to whether the user has done their daily entry to journal.html which then dynamically displays information on the journal page.
If the user has no previously created journal entries, the application renders the journal page with the latest day number set to 0.

In the HTML page, the passed parameters are used to dynamically display the user's journal entries. The following are the dynamically generated components in the journal page.

Daily journal reminder

If the user has yet to submit a journal entry for the day, the page will display a reminder.

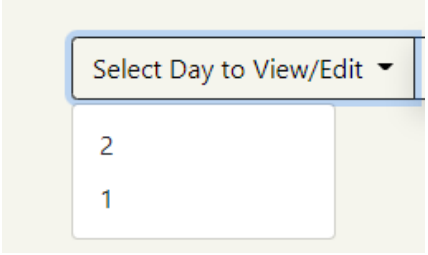


This is done by checking if the passed parameter dailyEntryDone is false, in which case it will create the element to display the reminder.

```
<% if(!locals.dailyEntryDone) { %>
    <div class="container">
        <div class="alert alert-danger noDataToday" role="alert">
            You have yet to submit a journal today
        </div>
    </div>

<% } %>
```

Journal Dropdown Selector

As mentioned earlier, this element allows the user to quickly select a specific journal to view and edit.

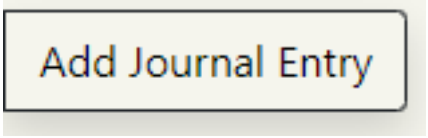


This is achieved by iterating through the parameter userJournalEntriesData from the backend and then using Bootstrap to allow the element when clicked to show the corresponding id, in this case, the corresponding id belongs to the view journal modal.

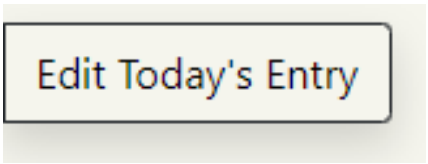
```
<ul class="dropdown-menu" style="width: auto;">
    <% if(locals.userJournalEntriesData && locals.userJournalEntriesData.length> 0) { %>
        <% userJournalEntriesData.forEach(function(parameter) { %>
            <li>
                <div class="dropdown-item" data-bs-toggle="modal"
                    data-bs-target="#openJournal<%= parameter.dayNum%>">
                    <%= parameter.dayNum%>
                </div>
            </li>
            <% }) %>

        <% } %>
</ul>
```

Add/Edit Journal Button

This element allows the user to either add a journal for the day if it has not yet been created



or edit today's journal entry if it has already been created.

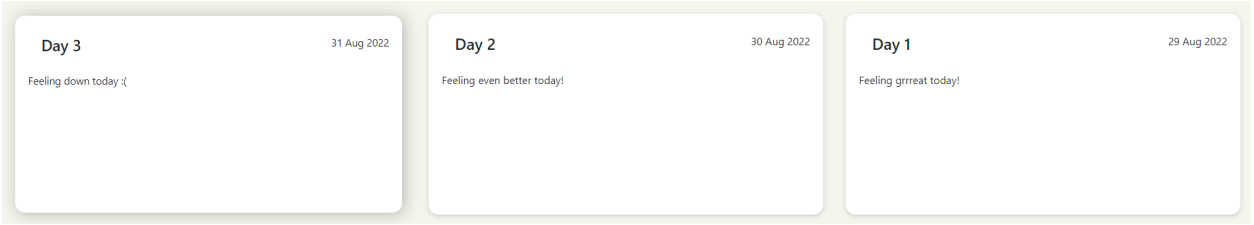


This is achieved by generating the element based on whether dailyEntryDone is true or false, it will also set the corresponding modal id as shown below. Using Bootstrap, when the button is clicked the target modal will be shown.

```
<% if(!locals.dailyEntryDone) { %>
    <button type="button" class="btn text-black border-dark shadow " style="width: auto;"
        data-bs-toggle="modal" data-bs-target="#addJournalEntry">
        Add Journal Entry
    </button>
<% } else { %>
    <button type="button" class="btn text-black border-dark shadow jounralBtn"  style="width: auto;"
        data-bs-toggle="modal" data-bs-target="#editJournalEntry<%= locals.latestDayNum%>">
        Edit Today's Entry
    </button>
<% } %>
```

Grid View of Journal Entries

Elements are created for each journal entry the user has previously added. Each element contains the journal content and the elements are sorted in descending order.

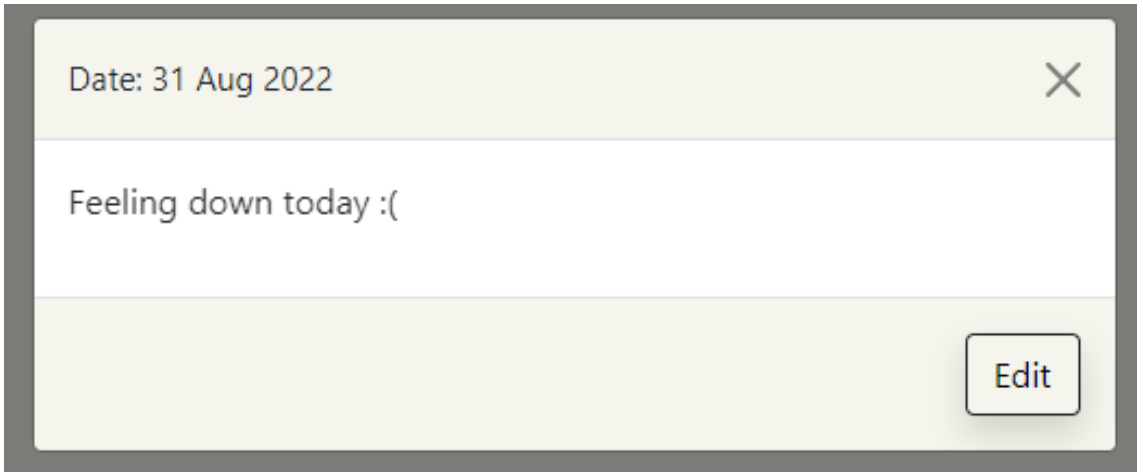


This is done by iterating through the userJournalEntriesData and generating grid-item elements to display the information.

```
<% if(locals.userJournalEntriesData && locals.userJournalEntriesData.length > 0) { %>
    <% userJournalEntriesData.forEach(function(parameter) { %>
        <div class="grid">
            <div class="grid-item journal-container" data-bs-toggle="modal" data-bs-target="#openJournal<%= parameter.dayNum%>" style="padding-left: 20px ;padding-right: 20px; padding-top:10px; border-radius: 15px; height: 300px; " >
                <div class="grid-header" style="display: flex;">
                    <h4 style="padding: 20px;">Day <%= parameter.dayNum %></h4>
                    <span style="text-align: end ; flex:auto; align-self:flex-end; padding-bottom: 34px"><%= parameter.dateAdded %></span>
                </div>

                <p class="wrap-content">
                    <%= parameter.content %>
                </p>
            </div>
        </div>
```

View Journal Modal Pop-up

This element is a Bootstrap modal which allows it to be hidden and then displayed when the corresponding grid element is clicked. It contains a button that allows the user to edit their journal entry.

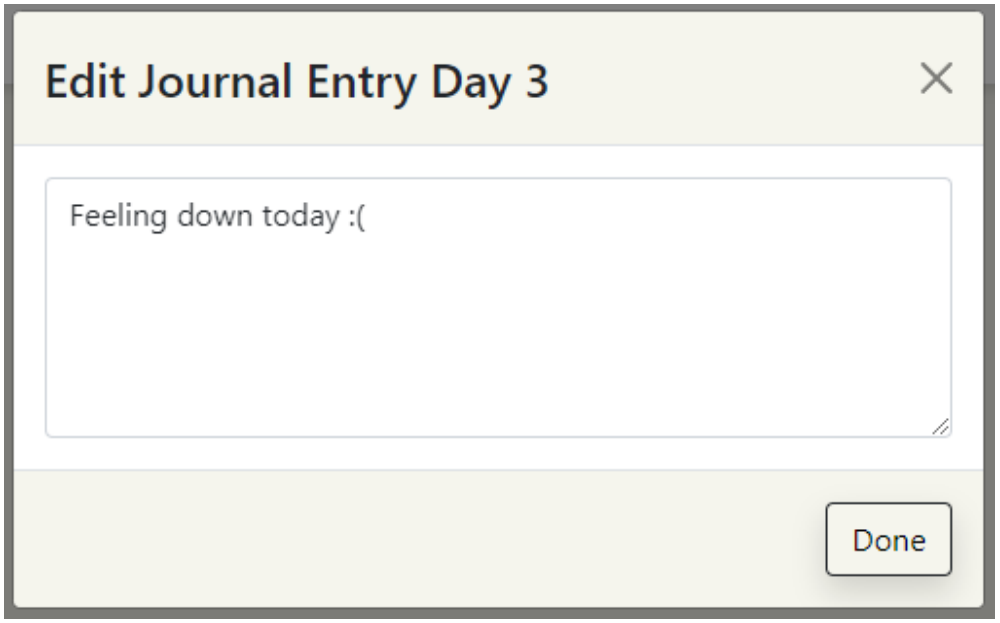


This element is generated in the same for loop iteration as in the grid display of journal contents. The Edit button will call the corresponding target edit modal.

```
<div class="modal " id="openJournal<%= parameter.dayNum%>">
    <div class="modal-dialog modal-dialog-centered mb-2 modal-dialog-scrollable">
        <div class="modal-content">

            <div class="modal-header">
                <span style="font-size: 15px;">Date: <%= parameter.dateAdded%> </span>
                <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
            </div>
            <div class="modal-body">
                <p><%= parameter.content %></p>
            </div>
            <div class="modal-footer" >
                <button class="btn text-black border-dark shadow" data-bs-toggle="modal" data-bs-target="#editJournalEntry<%= parameter.dayNum%>">
                    Edit
                </button>
            </div>
        </div>
    </div>
</div>
```

Edit Journal Modal Pop-up

This element is the same as the View Journal Modal Pop-up. It contains a button that will send data to the backend to update the Firebase database.

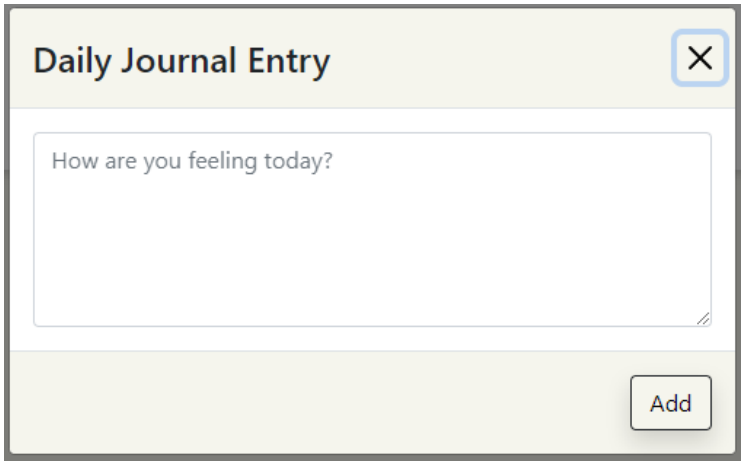


This element is also generated in the same for loop iteration as in the grid display of journal contents. This element is also a form which will submit to /editjournalentry as a POST request. It will pass the journal content, day number and date added to the backend.

```
<div class="modal" id="editJournalEntry<%= parameter.dayNum%>">
    <div class="modal-dialog modal-dialog-centered mb-2 modal-dialog-scrollable ">
        <div class="modal-content">

            <div class="modal-header">
                <h4 class="modal-title">Edit Journal Entry Day <%= parameter.dayNum%></h4>
                <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
            </div>

            <form method="POST" action="/editjournalentry" id="form">
                <div class="modal-body">
                    <textarea class="form-control w-100" rows="5" id="journalContent" name="journalContent"
                    placeholder="How are you feeling today?"><%= parameter.content %></textarea>
                </div>
                <input type="hidden" id="dayNum" name="dayNum" value="<%= parameter.dayNum %>">
                <input type="hidden" id="dateAdded" name="dateAdded" value="<%= parameter.dateAdded %>">
                <div class="modal-footer" >
                    <button type="submit" class="btn text-black border-dark shadow" data-bs-dismiss="modal">Done</button>
                </div>
            </form>
        </div>
    </div>
</div>
```

Add Journal Modal Pop-up

This element is also a Bootstrap Modal like the View and Edit pop-ups. This element allows the user to add their daily journal entry when they have yet to do so.



This element is not dynamically generated. This element contains a form that will submit to /addjournalentry as a POST request. It will pass the journal content, day number and date added to the backend.

```
<div class="modal" id="addJournalEntry">
    <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">

            <div class="modal-header">
                <h4 class="modal-title">Daily Journal Entry</h4>
                <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
            </div>

            <form method="POST" action="/addjournalentry" id="form">
                <div class="modal-body">
                    <textarea class="form-control w-100" rows="5" id="journalContent" name="journalContent"
                            placeholder="How are you feeling today?"></textarea>
                </div>

                <input type="hidden" id="dayNum" name="dayNum" value="<%= locals.latestDayNum + 1 %>">

                <div class="modal-footer">
                    <button type="submit" class="btn text-black border-dark shadow" data-bs-dismiss="modal">Add</button>
                </div>
            </form>

        </div>

    </div>

</div>
```

/editjournalentry route

This route is called when the user clicks the Edit button in the journal page which submits a form. This route will update the specific record of a day that has been edited by the user.

Firstly the application checks if the user has previous records, if true the application will add a record to the database. In this case, the record overrides the previous record thus updating it with the data from the form.

```
app.post("/editjournalentry", function (req, res) {

  journalRef
    .child(userID)
    .get()
    .then((snapshot) => {
      if (snapshot.exists()) {
        let journalDataObj = JSON.parse(JSON.stringify(snapshot.val()));

        for (let i in journalDataObj) {
          if (journalDataObj[i].dayNum == req.body.dayNum) {
            journalRef
              .child(userID)
              .child(i)
              .set({
                userID: userID,
                dateAdded: req.body.dateAdded,
                content: req.body.journalContent,
                dayNum: parseInt(req.body.dayNum),
              });

              res.redirect("journal");
          }
        }
      } else {
        console.log("User does not exist");
        res.redirect("/");
      }
    })
    .catch((error) => {
      console.error(error);
      res.redirect("/");
    });
});
```

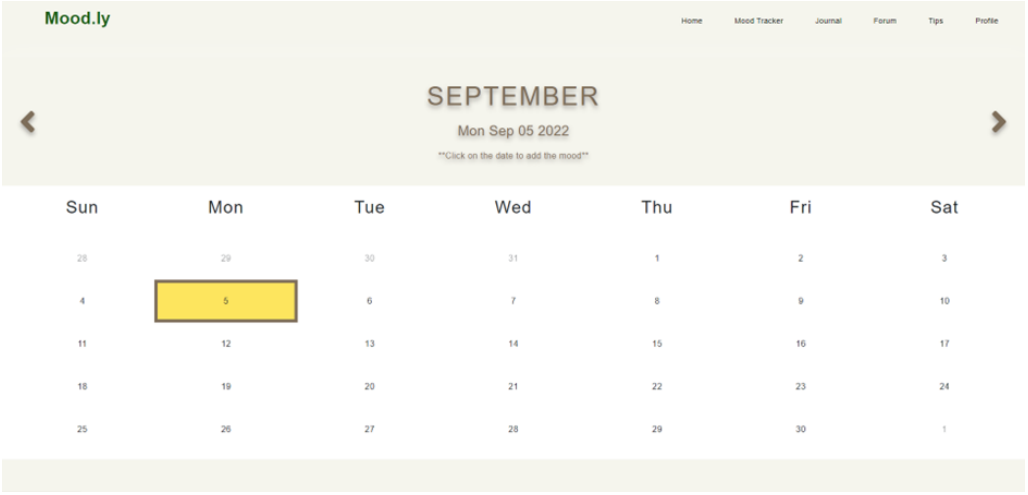
/addjournalentry route

This route is called when the user clicks the Add button in the Add Modal pop-up in the journal page. This route simply adds a new record into the database with data from the form in the journal page.

```
app.post("/addjournalentry", function (req, res) {

  journalRef
    .child(userID)
    .push()
    .set(
      {
        userID: userID,
        dateAdded: dateUtil.formatDate(Date()),
        content: req.body.journalContent,
        dayNum: parseInt(req.body.dayNum),
      },
      (error) => {
        if (error) {
          console.error(error);
          res.redirect("/");
        }
      }
    );

    res.redirect("journal");
});
```

5.3.5 - Mood Tracker Page

The Mood tracker page displays the mood calendar of the user. Users can Add their “mood of the day” into this calendar, and the date will be highlighted by the colour allocated to the specific mood. This calendar is mainly created using HTML, Javascript and Css. This mood calendar contains the Month, Today’s date, and the dates of the current month.



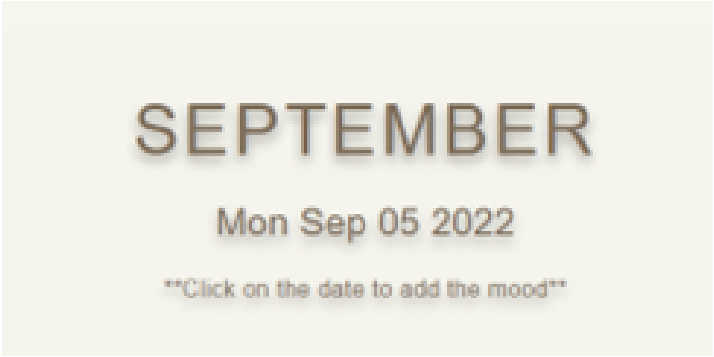
We used the Date() method, in order to get the date of the current day,

```
const date = new Date();
```

To get the specific year and month, I used the getFullYear() method to return the year of the specified date according to local time and getMonth() method returns the month in the specified date according to local time, as a zero-based value (where zero indicates the first month of the year).

```
const renderCalendar = () => {
    date.setDate(1);

    const monthDays = document.querySelector(".days");

    const lastDay = new Date(
        date.getFullYear(),
        date.getMonth() + 1,
        0
    ).getDate();

    const prevLastDay = new Date(
        date.getFullYear(),
        date.getMonth(),
        0
    ).getDate();

    const firstDayIndex = date.getDay();

    const lastDayIndex = new Date(
        date.getFullYear(),
        date.getMonth() + 1,
        0
    ).getDay();

    const nextDays = 7 - lastDayIndex - 1;

    const months = [
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December",
    ];
```

After getting the year and month, we used the document.queryselector().innerHtml to display the month and current date in the calendar.



HTML:

```
<div class="month">
```

```
<div class="date">
```

Javascript:

```
document.querySelector(".date h1").innerHTML = months[date.getMonth()];

document.querySelector(".date p").innerHTML = new Date().toDateString();
```

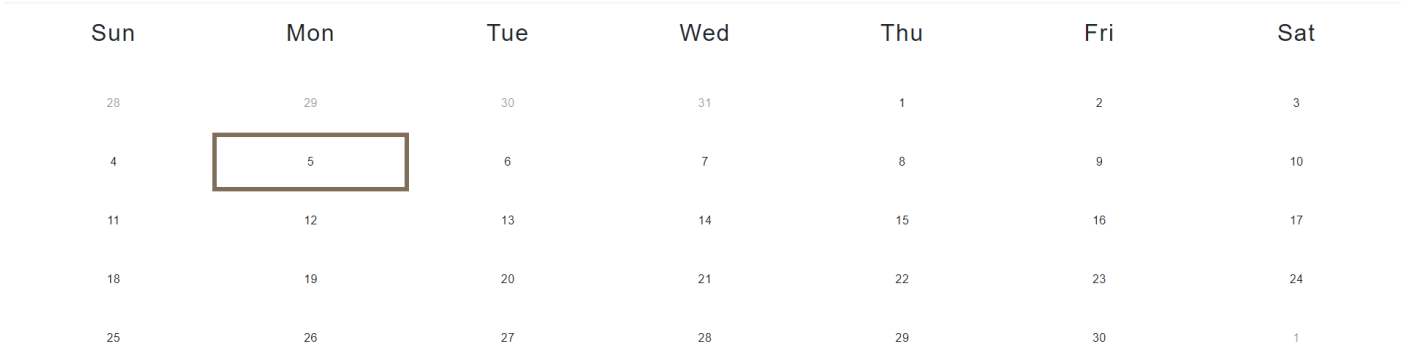
The right and left arrow icon buttons shown below are for the user to navigate to the next and previous month. using the getMonth() + 1 method to get the previous month and the getMonth() -1 method to get the next month.


```
<i class="fas fa-angle-left prev"></i>
```

```
<i class="fas fa-angle-right next"></i>
```

```
document.querySelector(".prev").addEventListener("click", () => {
    date.setMonth(date.getMonth() - 1);
    renderCalendar();
});

document.querySelector(".next").addEventListener("click", () => {
    date.setMonth(date.getMonth() + 1);
    renderCalendar();
});
```

This is the Dates of the current month, the page will get the current date,month and year to produce the calendar. By getting the last date of the current month [`lastDay`], the month before [`prevLastDay`], first day of the current month[`firstDayIndex`], and the last day of the month [`lastDayIndex`] to determine how the calendar will look like



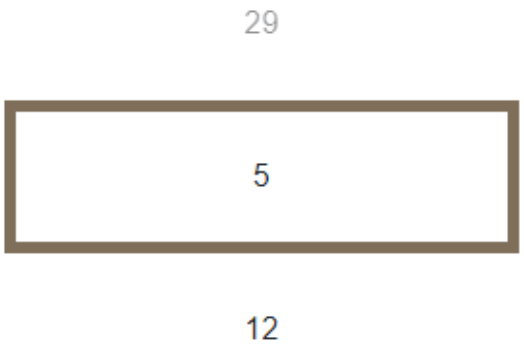
HTML:

```html
<div class="days" ></div>
```

Javascript:

```javascript
const date = new Date();


    const renderCalendar = () => {
        date.setDate(1);

        const monthDays = document.querySelector(".days");

        const lastDay = new Date(
            date.getFullYear(),
            date.getMonth() + 1,
            0
        ).getDate();

        const prevLastDay = new Date(
            date.getFullYear(),
            date.getMonth(),
            0
        ).getDate();

        const firstDayIndex = date.getDay();

        const lastDayIndex = new Date(
            date.getFullYear(),
            date.getMonth() + 1,
            0
        ).getDay();

        const nextDays = 7 - lastDayIndex - 1;

        const months = [
            "January",
            "February",
            "March",
            "April",
            "May",
            "June",
            "July",
            "August",
            "September",
            "October",
            "November",
            "December",
        ];

        document.querySelector(".date h1").innerHTML = months[date.getMonth()];

        document.querySelector(".date p").innerHTML = new Date().toDateString();

        let days = "";

        for (let x = firstDayIndex; x > 0; x--) {
            days += `<div class="prev-date">${prevLastDay - x + 1}</div>`;
        }

        for (let i = 1; i <= lastDay; i++) {
            if (
                i === new Date().getDate() &&
                date.getMonth() === new Date().getMonth()
            ) {
                days += `<div class="today" id="today" data-bs-toggle="modal" data-bs-target="#myModal">${i}</div>`;

            } else {
                var dayId = i.toString() + (date.getMonth() + 1).toString(10) + date.getFullYear();
                days += `<div id=${dayId}>${i}</div>`;
            }
        }

        for (let j = 1; j <= nextDays; j++) {
            days += `<div class="next-date">${j}</div>`;
            monthDays.innerHTML = days;
        }
```

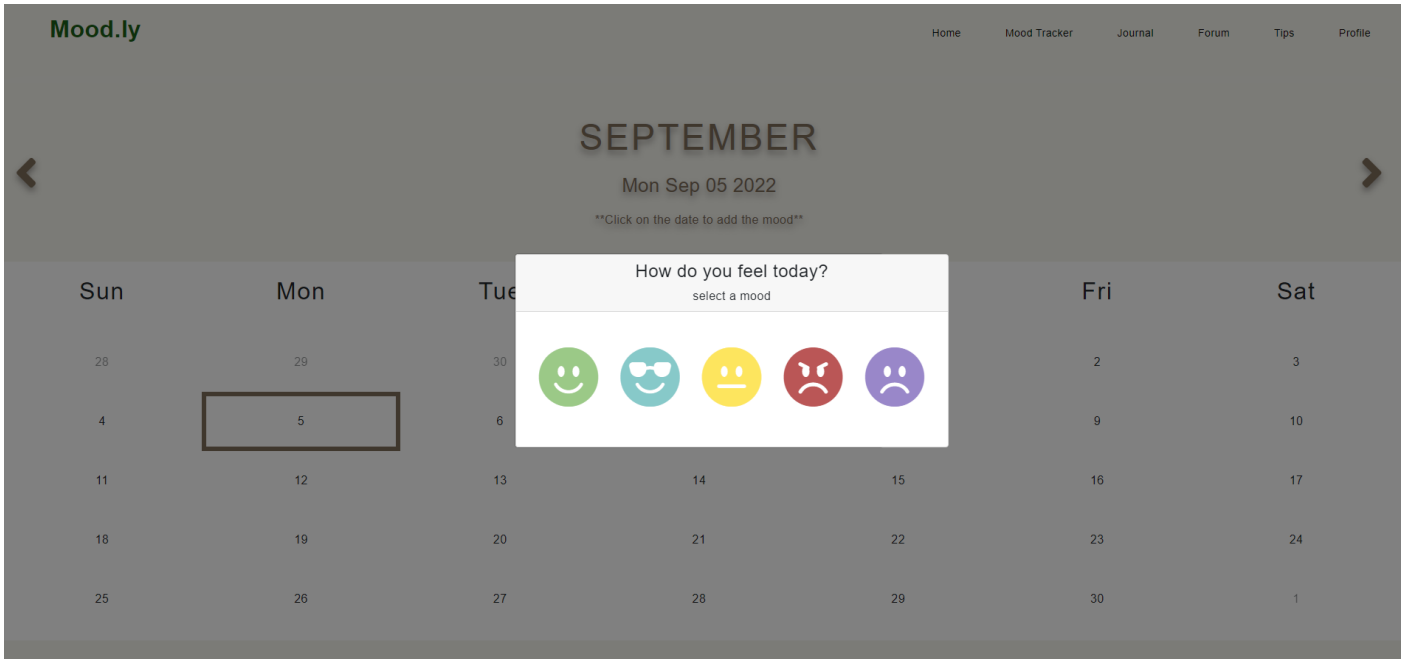
The current date have a border around it, it is to make it easier for the users to recognise which day is the current date and will be easier for them to add a mood into the calendar without needing to refer to another calendar to check the date of the current day



Calendar.CSS:

```
.today {
  border:0.4rem solid #7e6e5a
}
```

To add a mood, User will have to click on the current date and the modal will pop up on the screen for the user to select the most suitable mood to represent their day.



HTML:

```
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="exampleModalCenterTitle" aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered" role="document">
      <div class="modal-content">

        <div class="card" style="width: fit-content;">
            <div class="card-header" >
                <h4 style="justify-content: center; display:flex" >How do you feel today?</h4>
                <span style="justify-content: center; display:flex">select a mood</span>
            </div>
            <div class="card-body" style="justify-content: center; align-items: center;">
                <div class="moodPack" style="justify-content: center; display:flex">
                    <a href="/happy" style="padding:15px;">
                        <i class="bi bi-emoji-smile-fill" style="color: rgb(155, 201, 135);font-size: 5rem; "></i>
                    </a>
                    <a href="/cool" style="padding:15px;">
                        <i class="bi bi-emoji-sunglasses-fill" style="color: rgb(135, 201, 201); font-size: 5rem;"></i>
                    </a>
                    <a href="/neutral" style="padding:15px;">
                        <i class="bi bi-emoji-neutral-fill" style="color: rgb(253, 229, 94); font-size: 5rem;"></i>
                    </a>
                    <a href="/mad" style="padding:15px;">
                        <i class="bi bi-emoji-angry-fill" style="color: rgb(186, 86, 86); font-size: 5rem;"></i>
                    </a>
                    <a href="/sad" style="padding:15px;">
                        <i class="bi bi-emoji-frown-fill" style="color: rgb(153, 135, 201); font-size: 5rem;"></i>
                    </a>
                </div>
            </div>
        </div>

      </div>
    </div>
  </div>
```

Javascript: Allocate colors to the mood, when it is clicked and updated the display on the calendar

```
function highlightMood(id, mood) {
    if (mood == "neutral") {
            var element = document.getElementById(id);
            element.style.backgroundColor = "rgb(253, 229, 94)";
        } else if (mood == "happy"){
            var element = document.getElementById(id);
            element.style.backgroundColor = "rgb(155, 201, 135)";
        } else if (mood == "mad"){
            var element = document.getElementById(id);
            element.style.backgroundColor = "rgb(186, 86, 86)";
        } else if (mood == "sad"){
            var element = document.getElementById(id);
            element.style.backgroundColor = "rgb(153, 135, 201)";
        } else if (mood == "cool"){
            var element = document.getElementById(id);
            element.style.backgroundColor = "rgb(135, 201, 201)";
        } else {
            var element = document.getElementById(id);
            element.style.backgroundColor = "white";
        }
}
```

When the user clicks on the specific mood, it will post the update into the firebase database.

For “happy”, we used the Green color to represent the mood 

HTML:

```
<a href="/happy" style="padding:15px;">
    <i class="bi bi-emoji-smile-fill" style="color: rgb(155, 201, 135);font-size: 5rem; "></i>
</a>
```

Main.js:

```
//insert happy mood
app.get("/happy", function (req, res) {
  console.log("mood = happy")
  username = userID;
  mood = "happy";

  currentTime = new Date();
  let date = ("0" + currentTime.getDate()).slice(-2);
  let month = ("0" + (currentTime.getMonth() + 1)).slice(-2);
  let year = currentTime.getFullYear();
  let currentDate = date + + month + year;

  var userMood = moodRef.child(username).child(currentDate).child("today_mood");
  userMood.once('value')
    .then((querySnapshot) => {
      var a = querySnapshot.exists();          //querySnapshot.exists(); -- when query is
        if (a == true) {
            userMood.set({
                usernaany ser 1",
                mood: mood,
                currentTime: currentDate
            });
        } else if (a == false) {
            userMood.set({
                username: "user 1",
                mood: mood,
                currentTime: currentDate
            });
        } else {
          console.log("error adding mood")
        }
      });
  res.redirect('back');
});
```

For “cool”, we used the blue color to represent the mood 

HTML:

```
<a href="/cool" style="padding:15px;">
    <i class="bi bi-emoji-sunglasses-fill" style="color: rgb(135, 201, 201); font-size: 5rem;"></i>
</a>
```

Main.js:

```
//insert cool mood
app.get("/cool", function (req, res) {
  console.log("mood = cool")
  username = userID;
  mood = "cool";

  currentTime = new Date();
  let date = ("0" + currentTime.getDate()).slice(-2);
  let month = ("0" + (currentTime.getMonth() + 1)).slice(-2);
  let year = currentTime.getFullYear();
  let currentDate = date + + month + year;


  var userMood = moodRef.child(username).child(currentDate).child("today_mood");
  userMood.once('value')
    .then((querySnapshot) => {
      var a = querySnapshot.exists(); //-- when query is empty -- false || when query is
        if (a == true) {
          userMood.set({
            username: "user 1",
            mood: mood,
            currentTime: currentDate
        });
        } else if (a == false) {
          userMood.set({
              username: "user 1",
              mood: mood,
              currentTime: currentDate
          });
        } else {
          console.log("error adding mood")
        }
      });

  res.redirect('back');
});
```

For “neutral”, we used the yellow color to represent the mood

HTML:

```
<a href="/neutral" style="padding:15px;">
    <i class="bi bi-emoji-neutral-fill" style="color: rgb(253, 229, 94); font-size: 5rem;"></i>
</a>
```

Main.js:

```
app.get("/neutral", function (req, res) {
  console.log("mood = neutral")
  username = userID;
  mood = "neutral";

  currentTime = new Date();
  let date = ("0" + currentTime.getDate()).slice(-2);
  let month = ("0" + (currentTime.getMonth() + 1)).slice(-2);
  let year = currentTime.getFullYear();
  let currentDate = date + + month + year;

  var userMood = moodRef.child(username).child(currentDate).child("today_mood");
  userMood.once('value')
    .then((querySnapshot) => {
      var a = querySnapshot.exists();//-- when query is empty -- false || when query is
        if (a == true) {
          userMood.set({
            username: "user 1",
            mood: mood,
            currentTime: currentDate
        });
        } else if (a == false) {
          userMood.set({
              username: "user 1",
              mood: mood,
              currentTime: currentDate
          });
        } else {
          console.log("error adding mood")
        }
    });

  res.redirect('back');
});
```

For “mad”, we used the red color to represent the mood 

HTML:

```
<a href="/mad" style="padding:15px;">
    <i class="bi bi-emoji-angry-fill" style="color: rgb(186, 86, 86); font-size: 5rem;"></i>
</a>
```

Main.js:

```
app.get("/mad", function (req, res) {
  console.log("mood = mad")
  username = userID;
  mood = "mad";

  currentTime = new Date();
  let date = ("0" + currentTime.getDate()).slice(-2);
  let month = ("0" + (currentTime.getMonth() + 1)).slice(-2);
  let year = currentTime.getFullYear();
  let currentDate = date + + month + year;

  var userMood = moodRef.child(username).child(currentDate).child("today_mood");
  userMood.once('value')
    .then((querySnapshot) => {
      var a = querySnapshot.exists(); //querySnapshot.exists(); -- when query is empty -- f
        if (a == true) {
          userMood.set({
            username: "user 1",
            mood: mood,
            currentTime: currentDate
          });
        } else if (a == false) {
          userMood.set({
              username: "user 1",
              mood: mood,
              currentTime: currentDate
          });
        } else {
          console.log("error adding mood")
        }

    });
  res.redirect('back');
});
```

For “sad”, we used the purple color to represent the mood 

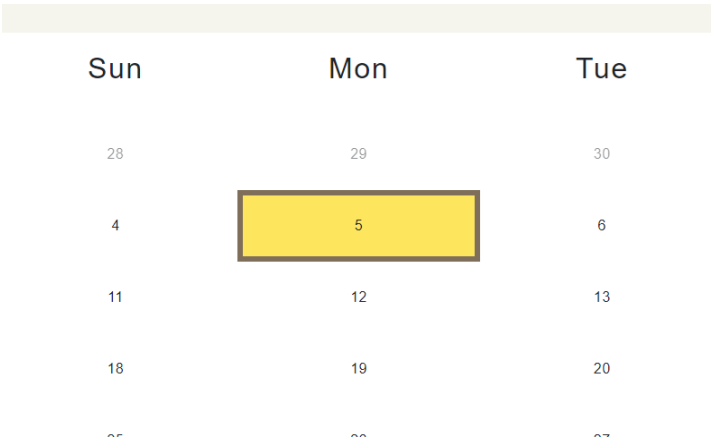
HTML:

```
<a href="/sad" style="padding:15px;">
    <i class="bi bi-emoji-frown-fill" style="color: rgb(153, 135, 201); font-size: 5rem;"></i>
</a>
```

Main.js:

```
app.get("/sad", function (req, res) {
  console.log("mood = sad")
  username = userID;
  mood = "sad";

  currentTime = new Date();
  let date = ("0" + currentTime.getDate()).slice(-2);
  let month = ("0" + (currentTime.getMonth() + 1)).slice(-2);
  let year = currentTime.getFullYear();
  let currentDate = date + + month + year;

  var userMood = moodRef.child(username).child(currentDate).child("today_mood");
  userMood.once('value')
    .then((querySnapshot) => {
      var a = querySnapshot.exists(); //querySnapshot.exists(); -- when query is empty -
      console.log(a);
        if (a == true) {
          userMood.set({
            username: "user 1",
            mood: mood,
            currentTime: currentDate
        });
        } else if (a == false) {
          userMood.set({
              username: "user 1",
              mood: mood,
              currentTime: currentDate
          });
        } else {
          console.log("error adding mood")
        }
      });
  res.redirect('back');
});
```

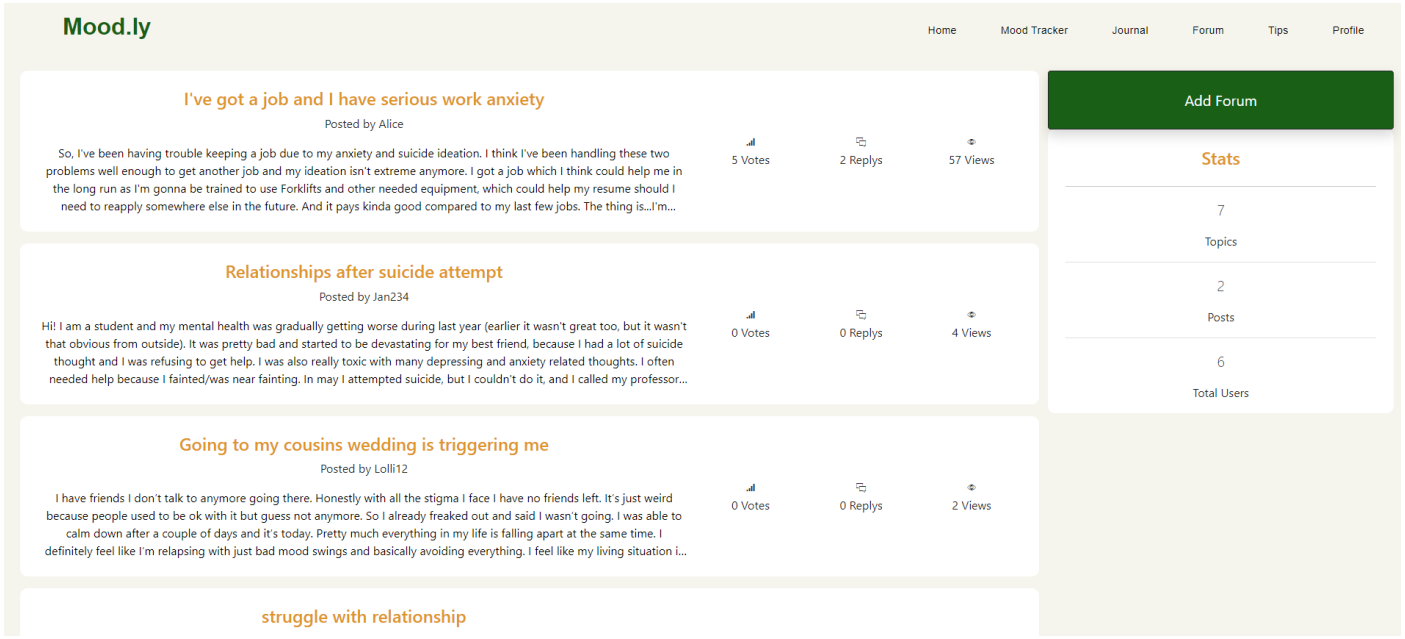
Once the mood is collected, the mood tracker page will refresh and will update the mood of the day with the allocated colour.



```
app.get("/mood_tracker", function (req, res) {
  //username = "user 1";
  username = userID;
  currentTime = new Date();
  let date = ("0" + currentTime.getDate()).slice(-2);
  let month = ("0" + (currentTime.getMonth() + 1)).slice(-2);
  let year = currentTime.getFullYear();
  let currentDate = date + + month + year;

  var today_mood = "empty";
  var prev_moods = {};

  moodRef.child(username).get().then((snapshot) => {
    if(snapshot.exists()) {

      let moodDataObj = JSON.stringify(snapshot.val());
      moodDataObj = JSON.parse(moodDataObj);



      for(let i in moodDataObj) {

        if(i != currentDate) {
          prev_moods[i] = moodDataObj[i].today_mood.mood;
        }
        else {
          var moodObj = moodDataObj[i].today_mood;
          today_mood = moodObj.mood;
        }
      }

      res.render("mood_tracker.html", {
        title: "Dynamic title",
        today_mood : today_mood,
        prev_moods : prev_moods
      });

    }
    else {
      console.log("No data available");
      res.render("mood_tracker.html", {
        title: "Dynamic title",
        today_mood : today_mood,
        prev_moods : prev_moods
      });
    }
  })
  .catch((error) => {
    console.error(error);
    res.redirect("/");
  });
```

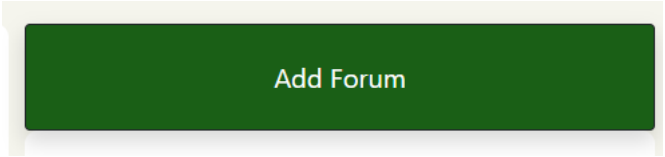
5.3.6 - Forum Page

The forum page displays the list of different forums created by different users, to talk about the topic that they created. Each forum topic, users are able to see the title, some parts of the content, the number of likes, replies and views. There is also a Stats to show how many users are there using mood-ly, the number of forum posts and the total number of replies collected. Users can also add their own forum by clicking on the “add forum” button.

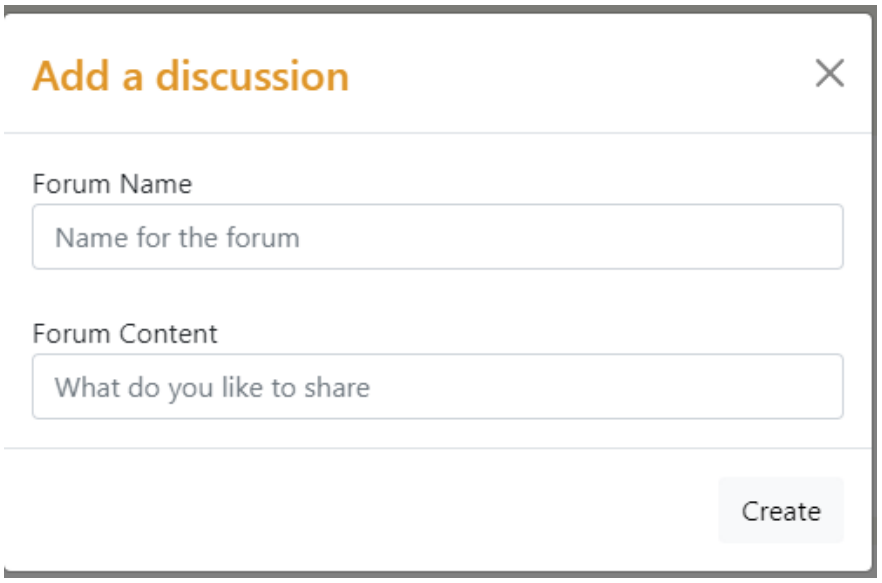



```
<div class="col-lg-9 mb-3">

        <% forumItem.forEach(function(item) { %>


    <div  class="card-btn row-hover pos-relative py-3 px-3 mb-3 forum-container" id="forumId"
      name="forumId" value="<%= item.forumId%>">
          <form method="get" action="./eachForum">

            <button type="submit" class="btn" id="forumId" name="forumId" value="<%= item.forumId%>">
              <div class="row align-items-center">
                <div class="col-md-8 mb-3 mb-sm-0 wrap-content">
                  <h4><%= item.forumTitle%></h4>
                  <p class="text-sm"><span class="op-6">Posted</span> <span class="op-6">
                      by <%= item.username%></span></p>
                  <p class="text-sm op-5">
                    <p class="text-black mr-2" href="#">
                      <%= item.forumContent%>
                    </p>
                  </p>
                </div>
                <div class="col-md-4 op-7">
                  <div class="row text-center op-7">
                    <div class="col px-1"> <i class="ion-connection-bars icon-1x"></i> <span class="d-block text-sm">
                      <%= item.numOfLikes%>
                        Likes
                      </span> </div>
                    <div class="col px-1"> <i class="ion-ios-chatboxes-outline icon-1x"></i> <span class="d-block text-sm">
                      <%= item.numOfReplies%>
                        Replies
                      </span> </div>
                    <div class="col px-1"> <i class="ion-ios-eye-outline icon-1x"></i> <span class="d-block text-sm">
                      <%= item.numOfViews%>
                        Views
                      </span> </div>
                  </div>
                </div>
              </div>
            </button>
          </form>
        </div>
    <% }) %>
</div>
```

When the user clicks on the “add forum” button, there will be a pop up modal for the user to add the forum name and the content of it.




```
<button type="button" class="btn btn-success border-dark shadow addJournalBtn" style="align-items: flex-end;" data-bs-toggle="modal"
  data-bs-target="#myModal">
  Add Forum
</button>
```




```
<div class="modal" id="myModal">
  <div class="modal-dialog">
    <div class="modal-content">

      <!-- Modal Header -->
      <div class="modal-header">
        <h4 class="modal-title">Add a discussion</h4>
        <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
      </div>

      <!-- Modal body -->
      <div class="modal-body">
        <form id="form" method="POST" action="/addForumItem">
          <div class="form-group">
            <label for="forum name">Forum Name</label> <span id="nameAlert"></span>
            <input type="text" class="form-control" id="forum_name" name="forum_name" placeholder="Name for the forum" required>
          </div>
          <br>
          <div class="form-group">
            <label for="forum content">Forum Content</label> <span id="nameAlert"></span>
            <input rows="3" type="text" class="form-control" id="forum_content" name="forum_content"
              placeholder="What do you like to share" required>
          </div>
      </div>

      <!-- Modal footer -->
      <div class="modal-footer">
        <button type="submit" id="submitBtn" class="btn btn-light">Create</button>
      </div>
      </form>

    </div>
  </div>
</div>
```

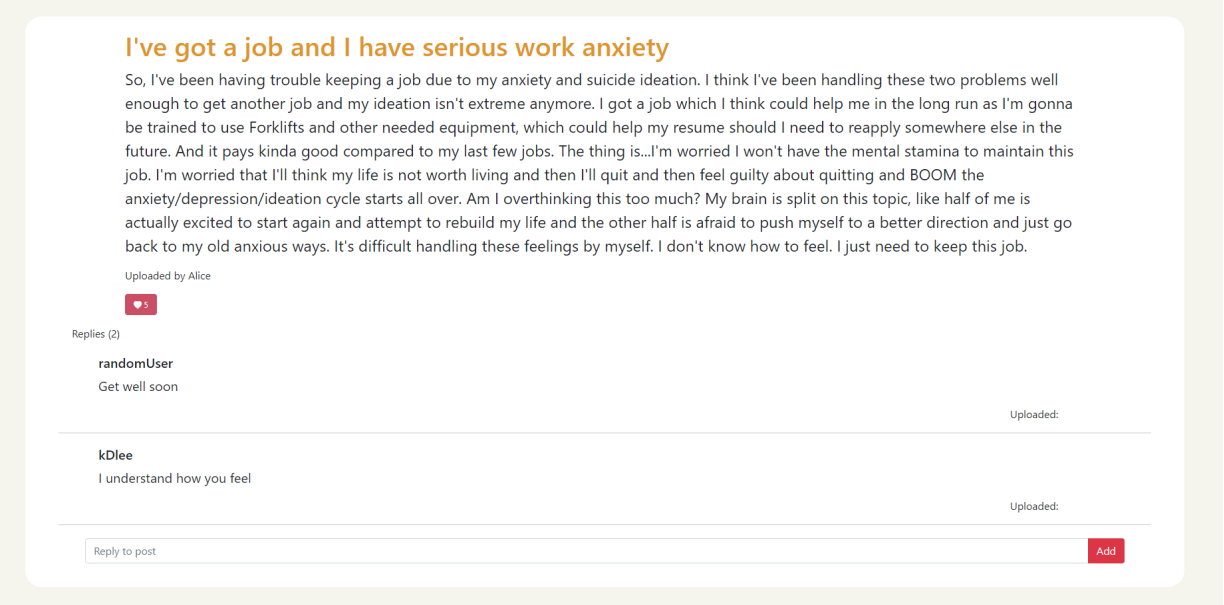
/addForumItem to create a new forum and add it into the firebase database, it will create a new forum.

```
var newforumList = forumRef.push();
newforumList.set({
  forumId: newforumList.key,
  username: username,
  forumTitle: req.body.forum_name,
  forumContent: req.body.forum_content,
  numOfLikes: 0,
  numOfViews: 0,
  numOfReplies: 0,
  currentTime: new Date()
});
```

And it will also update the number of forums, to display in the main_forumPage statistic list.

```
  updateTotal = totalForum + 1;

 forumDataRef1.update({
   totalForum: updateTotal
 })
```

When a user clicks into a specific forum, the forum title, content and the replies will be displayed in each forum.

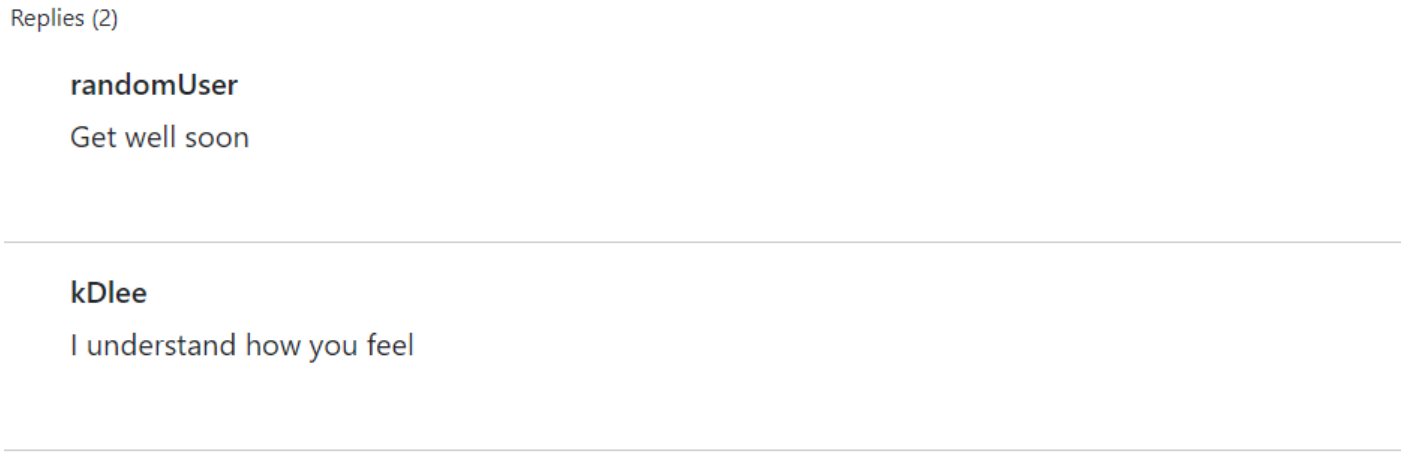


Users can click the like button if they want to show some love and care to the creator of this forum, the number will increase as the user clicks on it.

every click on the like button, will update the number of like for this specific forum in the firebase database

```
var numOfLikes = querySnapshot.val().numOfLikes;
var forumLikes = numOfLikes + 1;


forumRef.child(forumId).update({

  numOfLikes: forumLikes,

});
```

Users can see the replies that they post or posted by other users from the replies section below the post.



eachforum.html:

```
<% replyPosts.forEach(function(reply) { %>
  <div>
    <!-- <%= reply.forumReplyId %> -->
    <div class="replies">
      <h5>
        <%= reply.username %>
      </h5>
      <p style="font-size: 20px;">
        <%= reply.reply %>
      </p>

      <span style="padding-left: 90%;">Uploaded: <%= reply.currentTime %></span>
    </div>
    <hr>


    <% }); %>
```

```
var repliesRef = forumRef.child(forumId).child("forumReplies");
var replyPost = repliesRef;
var forum_replies = [];


replyPost.on('value', (data) => {
  data.forEach(function (snapshot) {
    forum_replies.push(snapshot.val());
  })
});
```

There is an input box below the replies for users to add their own replies and submit.

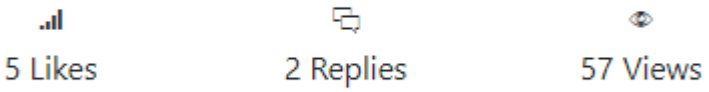

```
var ref = forumRef.child(forumId).child("forumReplies");
var newforumList = ref.push();
newforumList.set({
  forumReplyId: newforumList.key,
  username: username,
  reply: reply,
  currentTime: currentTime
});
```

When a new reply is added, it will update the statistic in the database and add increase the number of replies accordingly

```
var Totalcomments = querySnapshot.val().Totalcomments;

updateTotal = Totalcomments + 1;

forumDataRef1.update({
  Totalcomments: updateTotal
})
```

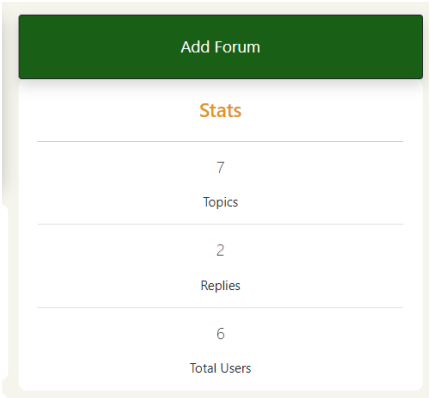
```
var repliesCount = numOfReplies;
updated_replies_count = repliesCount + 1;
var forum_post_ref = forumRef.child(forumId);

forum_post_ref.update({
    numOfReplies:  updated_replies_count
});
```

Beside each forum topic at the main_forumpage, there will be statistics shown for each forum, for users to see the number of likes, replies and views. The number of likes will change based on how many users click on the like button in each forum page. The number of replies will change as the users add their posts in each specific forum. The number of views will change when each user clicks into the forum.




```
var viewCount = numOfViews;
updatedViewCount = viewCount + 1;
var ref = forumRef.child(forumId);

ref.update({
  numOfViews: updatedViewCount
});
```

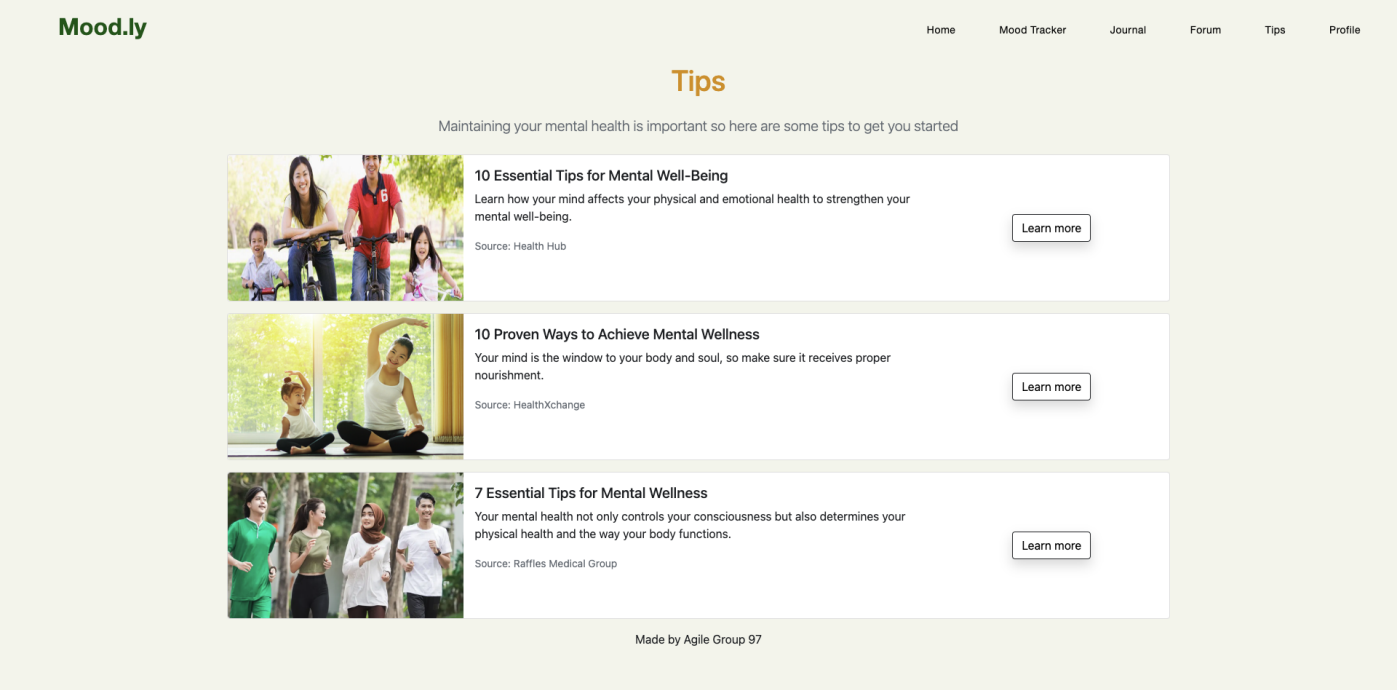
And this stats box will show the overall statistic of how many replies, posts and users there are in Mood.ly.




```
<div class="row mx-0 stats-content" >
  <div class=" flex-ew  py-3 text-center border-bottom border-right"> <p
      class="d-block lead font-weight-bold" ><%= totalForum %></p> Topics </div>
</div>
<div class="row d-flex flex-row op-7 mx-0 stats-content">
  <div class="col flex-ew text-center py-3 border-bottom mx-0"> <p
      class="d-block lead font-weight-bold"><%= totalComments %></p> Replies </div>
</div>
<div class="row d-flex flex-row op-7 mx-0 stats-content">
  <div class="flex-ew py-3 text-center border-right mx-0"> <p class="d-block lead font-weight-bold"><%= totalUsers %></p>
    Total Users </div>
</div>
```

5.3.7 - Tips Page

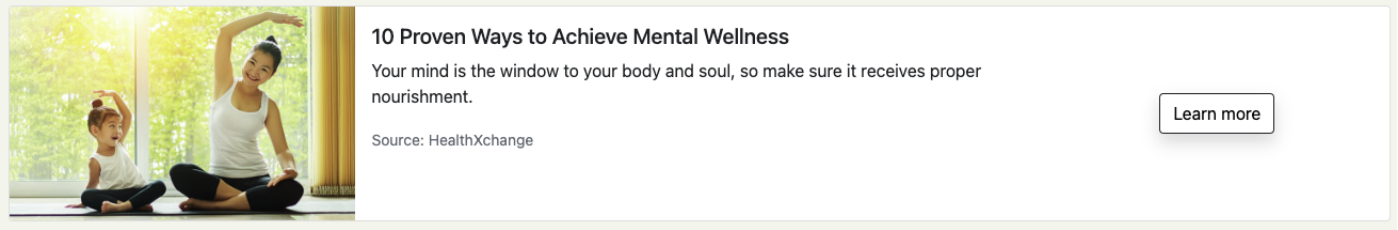
The tips page displays useful tips which the users may find helpful if they would like some additional information that they can follow in order to further improve on their mental health.



The top of the page contains the title of the page and what the page aims to offers.

As for each tip, they are placed in a card that contains an image, the title of the tip, a brief description of it, its source, and a link to the website where the user can find out more about said tip. Since we felt that we may not be the best people to give proper advice to the users, by linking them to proper websites we would be able to provide them with the best possible help.

Each card was done using the card component in Bootstrap and there is a “Learn more” button that the users can click on to take them to the respective websites for more information.



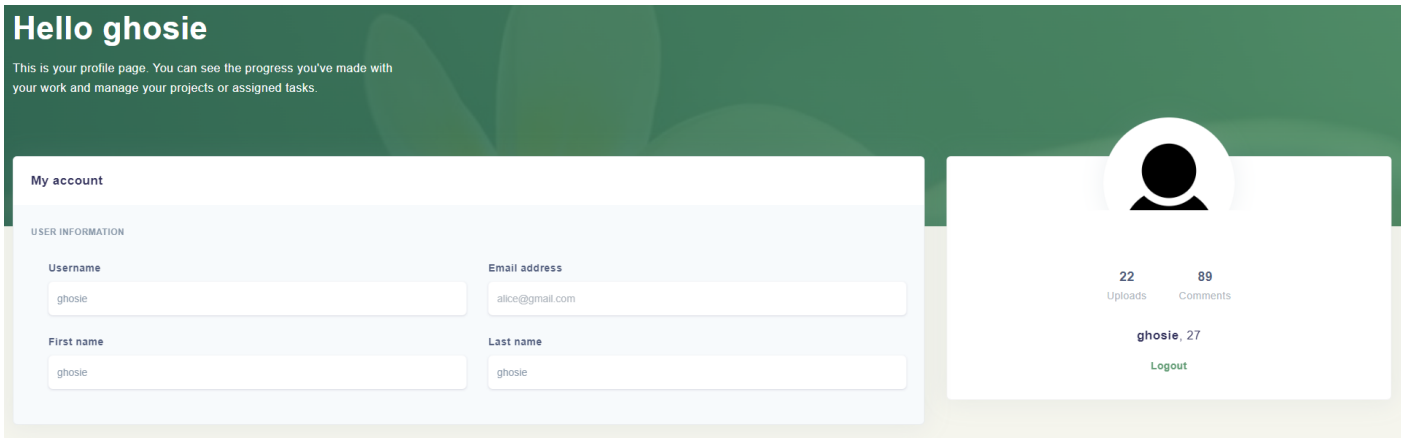
Similar to the other pages, a DataSnapshot from Firebase was used in order to receive the necessary information that we want to display onto the tips page, which are the information needed for each tip on each card component.

```
app.get("/tips", function (req, res) {
  tipsRef
    .once("value", (snapshot) => {
      if (snapshot.exists()) {
        var tips = snapshot.val();
        console.log(tips);
        res.render("tips.html", { tipsData: tips });
      } else {
        console.log("No data available");
      }
    })
    .catch((error) => {
      console.error(error);
    });
});
```

And by using EJS in line with HTML, we are able to read and display the information. Furthermore, by making use of an if statement and the forEach() method in the EJS code, we are able to check that Firebase contains the relevant data and automatically creates a card for each tip. Thus, when we add in new data for additional tips to Firebase, new cards will be created and displayed containing the new tip that was added.

```
<section id="tips">
    <div class="container">

        <% if(tipsData.length > 0) { %>
            <% tipsData.forEach(function(tips){ %>
                <div class="card mb-3" >
                    <div class="row g-0">
                        <div class="col-md-3">
                            <img class="img-fluid rounded-start tipsImage" alt="Placeholder" src=<%=tips.image %> >
                        </div>
                        <div class="col-md-6">
                            <div class="card-body">
                                <h5 class="card-title">
                                    <%= tips.title %>
                                </h5>
                                <p class="card-text">
                                    <%= tips.intro %>
                                </p>
                                <p class="card-text"><small class="text-muted">Source: <%= tips.author %></small></p>
                            </div>
                        </div>
                        <div class="col-md-1 "></div>
                        <div class="col-md-2 ">
                            <a class="btn text-black border-dark shadow " href=<%=tips.link %> role="button">Learn more</a>
                        </div>
                    </div>
                </div>
            <% }) %>
        <% } %>
    </div>
</section>
```

5.3.8 - Profile Page



The Profile displays the different details that the user has filled up during the sign-up process. Based on the information given on the sign-up page, only selected data will be displayed while the rest will be stored in the database. Fields like passwords will not be stored in the database, nor be displayed on the profile page. This is to keep users' detail confidential when it comes to security.

A "Logout" button is also included for users to log out if they do not want to be kept signed in all the way even when they are not using the web application.

```
thisUserRef.once('value')
  .then((querySnapshot) => {
      if (!querySnapshot.numChildren()) {
                  throw new Error('expected at least one result');
      }

      if (!querySnapshot.exists()) {
        throw new Error(`Entry ${userID} not found.`);
      }

    var username = querySnapshot.val().name;
    var email = querySnapshot.val().email;
    var name = querySnapshot.val().username;
    var educational_level = querySnapshot.val().educational_level;
    var age = querySnapshot.val().age;
    var phone_number = querySnapshot.val().phone_number;
    var school = querySnapshot.val().school;

    res.render("profile.html", {
              userFullName: username,
              userEmail: email,
              userUsername: name,
              userEducationalLevel: educational_level,
              age: age,
              userPhoneNumber: phone_number,
              userSchool: school
            });
```

DataSnapshot is used in the Profile page to obtain the data stored in the Firebase. EJS was used to display it in our HTML. We have obtained the snapshot of the different fields that the user has input during sign-up. Even though we only choose to display a few on the Profile page, the rest of the data in the different fields can also be displayed using EJS if we wanted to.

```
<h6 class="heading-small text-muted mb-4">User information</h6>
<div class="pl-lg-4">
  <div class="row">
    <div class="col-lg-6">
      <div class="form-group focused">
        <label class="form-control-label" for="input-username">Username</label>
        <input type="text" id="input-username" class="form-control form-control-alternative"
          placeholder="Username" value="<%= userUsername %>">
      </div>
    </div>
```

Chapter 6: Unit Testing

6.1 - UI Testing

After we had completed the web application, we presented the final product to our stakeholders, who were the people who would most likely use our app. We conducted this meeting online through platforms like Zoom and Discord due to the ongoing pandemic and social distancing measures.

As we were using a user-centred design approach, our aim was to be able to develop a usable frontend for our web application, which we were able to achieve. We scheduled these kinds of online meetings as they were easier to arrange and we were able to get feedback from the users so there would not be any misunderstandings between us as the developers and the users.

Below are questions that were asked with regards to UI Testing:

1. Is the web application interface simple and easy to navigate?
2. Is the information displayed on the web application organized and easy to process?
3. Are the features of the web application intuitive to use?
4. Is there anything that you think we can further improve on?

Feedback received:

1. The colour and design of some of the pages were not consistent.
2. A back button would be useful to exit a forum post back to the forum page.

6.2 - User Testing

Unit testing (user testing: usability of application)
For the user testing, a survey was provided to the stakeholders to get feedback with regard to the usability of the web application. The questions that were created were based on Davis, F. D. (1989) *Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology.* **MIS Quarterly**, 13:3, 319-340. The questions were also split into two categories of perceived usefulness and perceived ease of use where users would rank each question from 1 to 5, where 1 is most unlikely and 5 is most likely. Below are the questions that were crafted:

Perceived Usefulness

1. Using the web application would allow me to track my mental health status more easily and effectively.
2. Using the web application would help improve my mental health.
3. Using the web application would create greater awareness for mental health.
4. Using the web application would make me more inclined to share my issues/open up more to others.
5. I would find the web application useful on a daily basis.

Perceived Ease of Use

6. Learning to use the web application would be easy for me.
7. I would find it easy to navigate my way around the web application.
8. I feel that the aspects of the web application are well integrated together.
9. There were aspects of the web application that were overly complex.
10. I would feel confident using the web application.
11. Other comments with regard to the usability of the web application (open-ended question).

Feedback received:

1. It would be ideal if the user could add in their own custom moods for the mood track in the case that their current mood does not fall into one of the moods currently provided.
2. For the forum and journal, users should be able to add images to improve on the content they are able to show.

Chapter 7: Conclusion

7.1 - Evaluation

Our team decided to be involved in both the software development and the report writing instead of splitting the workload as everyone is trying to get familiar with all the extra libraries that we have decided to use. This way, everyone will have a chance to develop the software, and note down their experiences in the report as well. Therefore, all of us had roles in both software development and report writing. Our work was split equally between each member, with everyone taking a maximum of tasks per component. This is to make sure we do not overload ourselves with work and run out of time to focus on other commitments.

Our team has decided to adopt the Kanban methodologies when it comes to software development as we have already done our research and analysis in the first part of the project. This way, we can split the tasks based solely on software development and then take the time to implement each task.

At the beginning of creating the application, all members of the team did not have prior knowledge about the Firebase authentication and Firebase real-time database, which is needed for our application's database. At first, we were having trouble with how to apply various functions from the Firebase. However, we took the time to learn and research more about it by sharing some basic tutorial videos and Firebase documentation. We were able to solve any problems that we faced when we come together as a team and worked around the errors. Since we are all new to Firebase, we took a long time to figure out how to use and implement the database into our project. This did not stop us from giving up but instead, help one another to overcome all the challenges faced. Whenever one of the members is having an issue on their part, we will share the issues that we face, explain the problems and share what we have done to prevent it with one another so that we can all find a solution together by recommending a way on how to make the code robust.

Meeting were held weekly to discuss our progress, and be open to any new ideas or changes that were beneficial to our project. We will evaluate our work and set the tasks for the following week in every meeting so that everyone can stay on task during the tight timeline. The Kanban board was used to track each other progress and make sure that everyone is on task. With the consistent updates and improvements that were discussed during these meetings, the team have added changes to adopt a more user-centred design as compared to our previous ideas.

During the software development, there were some challenges faced as we realised the ideas that we have are harder to implement. For example, we were not able to implement some of the features such as the search option in the tips pages for easier usability. This was then discussed within the team, and we deemed it better for us to continue implementing it as our further development idea since the important part was to make sure that there were different tips displayed on the tips page.

We also had other commitments on top of this Agile Project, which also defers us from focusing on the project at all times. Because of this, our team had to work on the project at different timing and only come together once a week to discuss our progress. However, even though we had weekly meetings to compile everything, we constantly keep in touch with one another to ask about each other progress. We also spilt the tasks where each member took a page of the web application. It was harder to generalise the syntax we used and the variables we used. The design of the web application and its exact position for the different divisions were different for us. We had to take the time to gather all our positioning and colours for the different pages to have a constant look. For example, all the pages had a different position for our navigation bar due to the differences in margins and paddings. In the end, we decided to create a separate HTML file for our navigation bar and add it to the individual pages on its own so that everyone will have the same navigation bar. This allows for a better look for our web application as the user navigates across the different pages of our web application.

All in all, despite all the challenges we face, be it as a team or individually, we managed to help one another through all the setbacks and deliver our user-centred software within a tight timeline. This was only possible since we all worked together to achieve what we have set out to do at the start of the project.

7.2 - SWOT Analysis of Mood.ly

We have done a SWOT Analysis of our final product, Mood.ly.

Strengths

Mood.ly is a mental health self-help web application with all the important and necessary features required based on the user requirement research that we have done at the start of the project. Those existing mental health help application only provides certain features per application. In our web application, we build an all-rounded web application for users to track their mental health efficiently. Our web application can also be used on mobile devices as we have implemented the media queries technique. This allows users to access their application anytime even if they are on the move.

Some personal features such as Journal and Moos Tracker are dynamic. They follow the database to specific users, allowing users the data to their own accounts. The remaining features such as Tips and Forum are static since they offer the same set of information to all the different users. Firebase was used to achieve this, and we successfully implemented the different sessions for each specific user.

Weaknesses

There are still limitations to what each feature can do. These features implemented serve their purpose as a basic interaction with the users. However, there are still more complicated functions that we did not implement for each feature. This would take more resources and time which can be done if given so.

Since we are using the Test Mode in Google Firebase, the Realtime Database is left completely open to the Internet and thus, the database security rules were configured to stop allowing requests after a period of time. Therefore, client access was denied after the said period of time since it makes our web application vulnerable to attacks.

Opportunities

Since we had limited time and resources, we only implement a few important features required in a mental health web application. New features can be added to the web application such as having an online consultation with an actual counsellor instead of a chatbot since bots tend to give the same replies to most messages due to the lack of understanding.

Threats

There are other platforms offering the same services as our web application. Many health organizations are able to put in the time and resources to implement such an idea in their own platform since they already have a dedicated user community.

7.3 - Improvements (Non-Technical)

Features that can be added to improve the applications:

- Online consultation to be given to users who need to talk to someone immediately
- Made it into a mobile application, be it on Android or IOS for all the different users

7.4 - Further Development (Technical)

Technical features not implemented currently:

- Journal Search function to allow users to search specific keywords in their journal.
- Tips Search function to allow users to search for specific Tips.
- Implement firebase to be production-ready by implementing proper database rules.
- Ensure user data is secure by restricting access to the firebase to only key admins.
- Allow images to be added to journal entries and forum posts.

7.5 - Timeline for Further Development

Subsequently after the analysis and evaluation of Mood.ly, the following timeline was created as a means to document the future development of Mood.ly beyond the final project submission. Currently, the objectives for Q3 2022 have already been achieved.

Q3 2022

1. Complete all of the frontend logic planned for the final project submission
2. Complete all of the backend login planned for the final project submission

Q4 2022

1. Complete backend logic for search functionality of the journal and tips pages
2. Complete backend logic for adding images to journal entries and forum posts
3. Implement proper database rules to ensure that Firebase is production-ready
4. Ensure that user data is secure and is only accessible by the relevant parties

Q1 2023

1. Create a prototype that allows users to have online consultation through the web application
2. Conduct user testing with the new prototype
3. Include user feedback on the new feature
4. Complete the frontend and backend logic for the new online consultation functionality

Q2 2023

1. Create a prototype for a mobile application version of the web application
2. Conduct user testing with the new prototype
3. Include user feedback on the new iteration
4. Complete the frontend and backend logic for the new mobile application version

7.6 - Personal Reflection

Throughout this project, from research to software development to user testing, I have been actively involved in the whole process. I have also taken the initiative to set meeting dates and times for everyone to track one another progress. This way, we are able to know what stage of development are we in, and if there is a need for us to catch up with our tasks. Since we are given a limited time, we have to keep improving on the software almost every other day in order to reach our aims and objectives for the final product.

I also had to ask about their progress on days without meetings so as to know that they are on par with the tasks they have. This is to also check if they have any issues on their end so that we can help one another as a team. I had the easier pages to work on which were the Home Page and the Profile Page so I had more time to help others with their issues and do more parts of the report as compared to the rest. Even though we have assigned specific chapters and tasks to each member, we end up touching every part of the software and report since we are always helping one another.

Working in a team has taught me that communication is key. It is not easy for everyone to meet constantly due to our different schedules and commitment but it is important to communicate with one another about the project. This helps me to judge the progress of our work and adjust our timeline accordingly so that no one will be far behind on their task allocation.

We worked exceptionally well together in terms of our work progress and communicating with one another. The software development may be hard at times when we are unsure of the Firebase usage and the features that are harder to implement but as long as someone asks about their issues, every member is willing to help them with it even if they are not done with their own work. This is what teamwork is all about.

Even though we had a very short timeline to do the software development and the report writing, we still tried our best to deliver our final product and documentation. Our final product may have deviated from our initial plan, but we were able to adapt to the setbacks and difficulties almost right away so that we do not get stuck on our issues for too long.

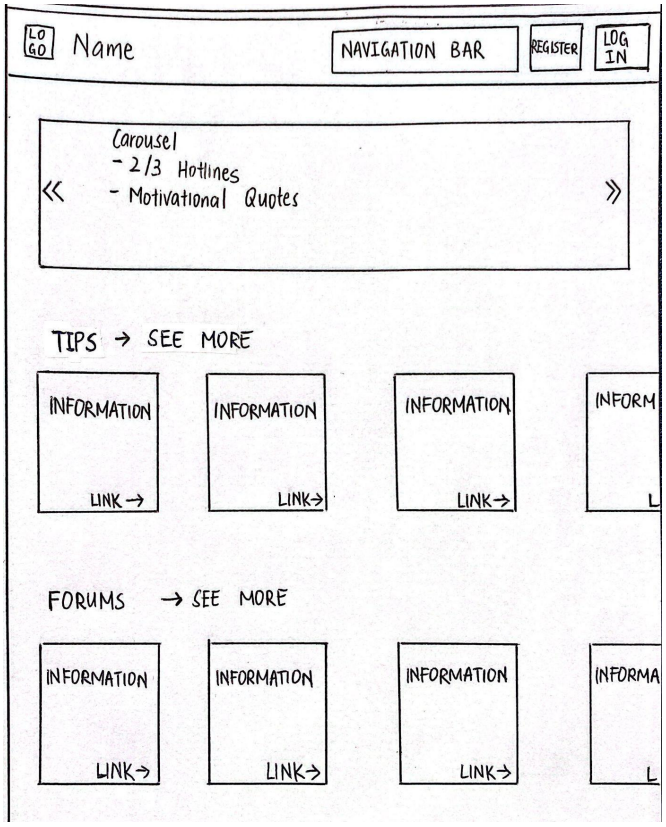
This concludes the end of our final project, with the software and report.

Chapter 8: References

8.1 - Appendix

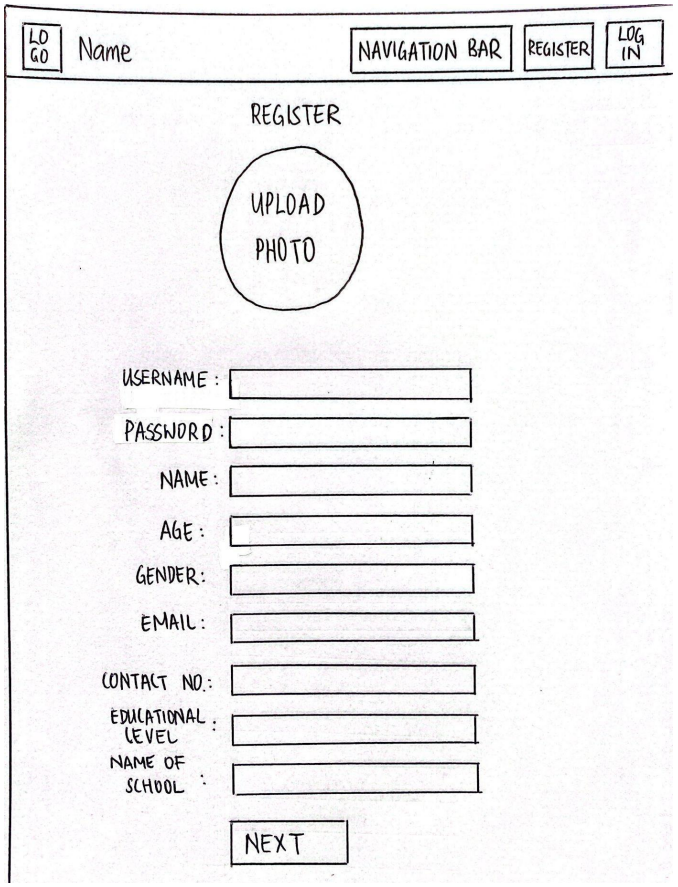
8.1.1 - Low Fidelity Prototype

At the start of our designing process, we sketched out a few different pages to display our idea on paper, before we start doing our wireframes.



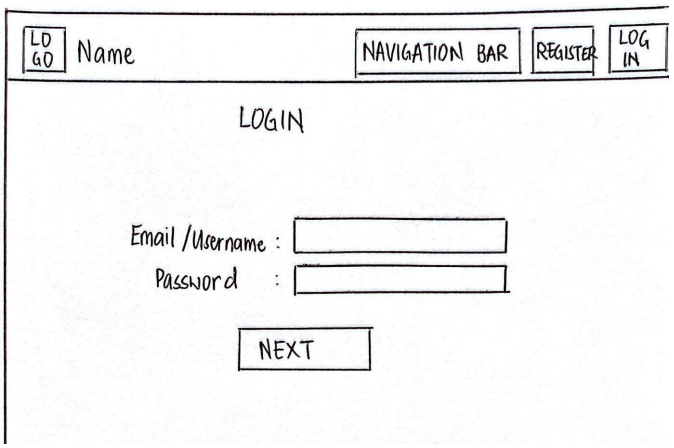
Home Page

- Carousel displaying a few hotlines that are 24/7 available in case if the user decides to call in
- Carousel will also display a few different motivational quotes that will be on the loop
- Popular tips and forum will be shown on the home page with links at the bottom of the panel for easy access to more information
    - Hyperlinks to other pages



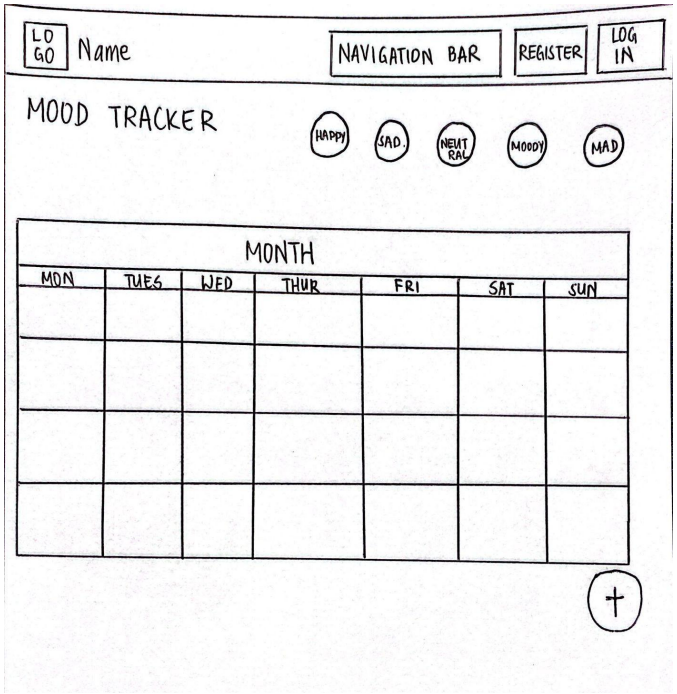
Register Page

- Requires user to fill in all the input fields in order to gather enough information to help the user if necessary
- Details will be confidential, only admin have rights to the information



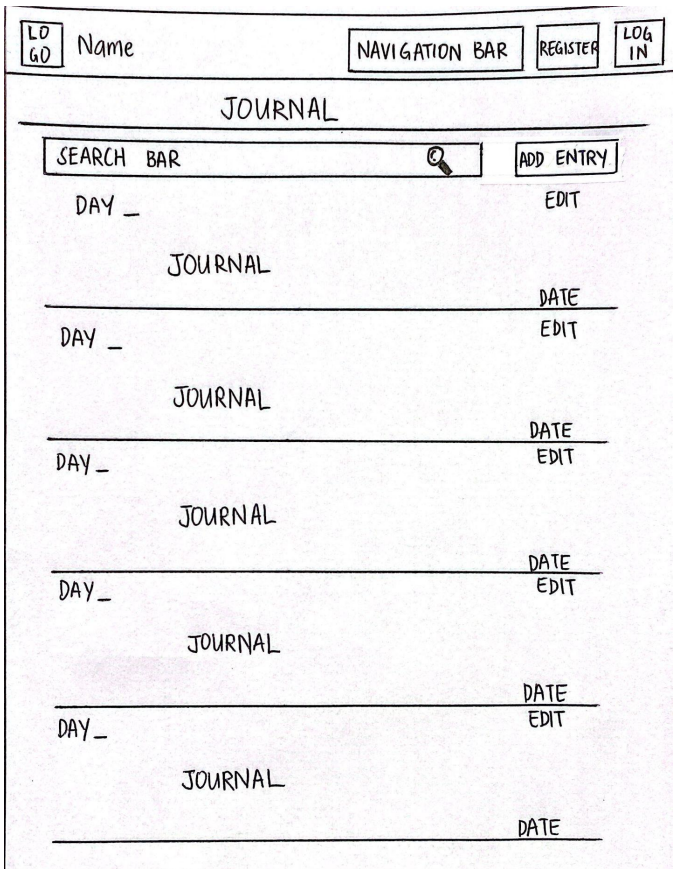
Login Page

- Allow users to log in to their account to keep track of their mood, journal and forum



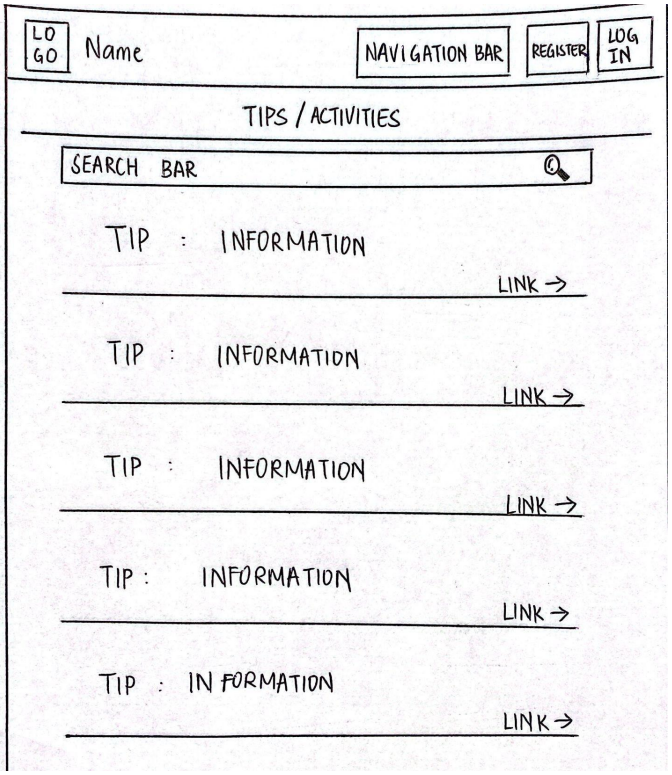
Mood Tracker Page

- Allow users to keep track of their mood for that day
- Each mood has a different colour and once chosen for the day, the particular day will turn into the assigned colour on the calendar
- Users are able to add their mood everyday using the “Add” button on the bottom



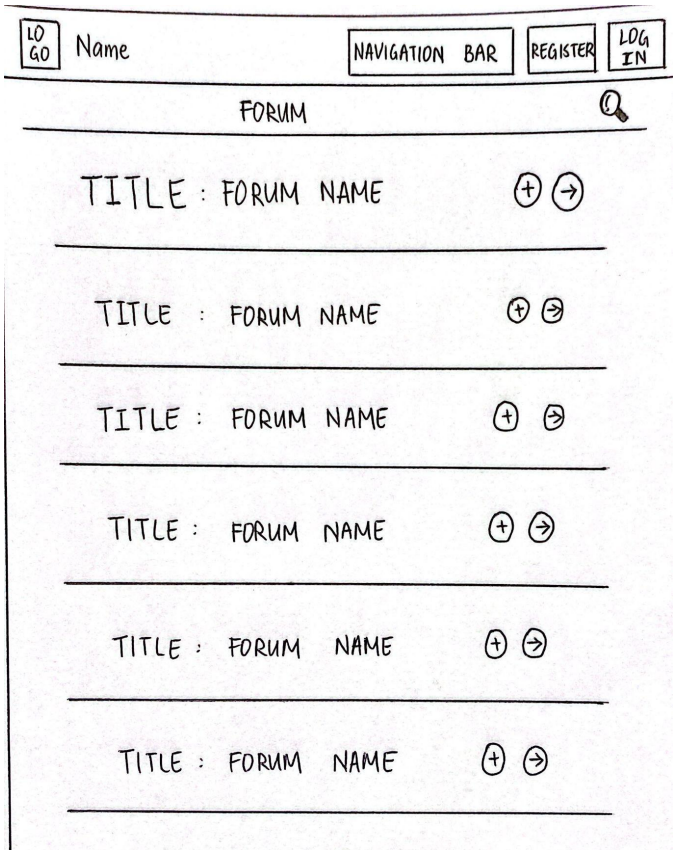
Journal Page

- Allow users to type in a short journal of their day
- The date will be displayed at the bottom of each entry for easy access
- Search bar implemented for users to search for a particular journal
- Newest entry will be at the top, while the older ones will be at the bottom
- Each journal has the option to be edited at any time



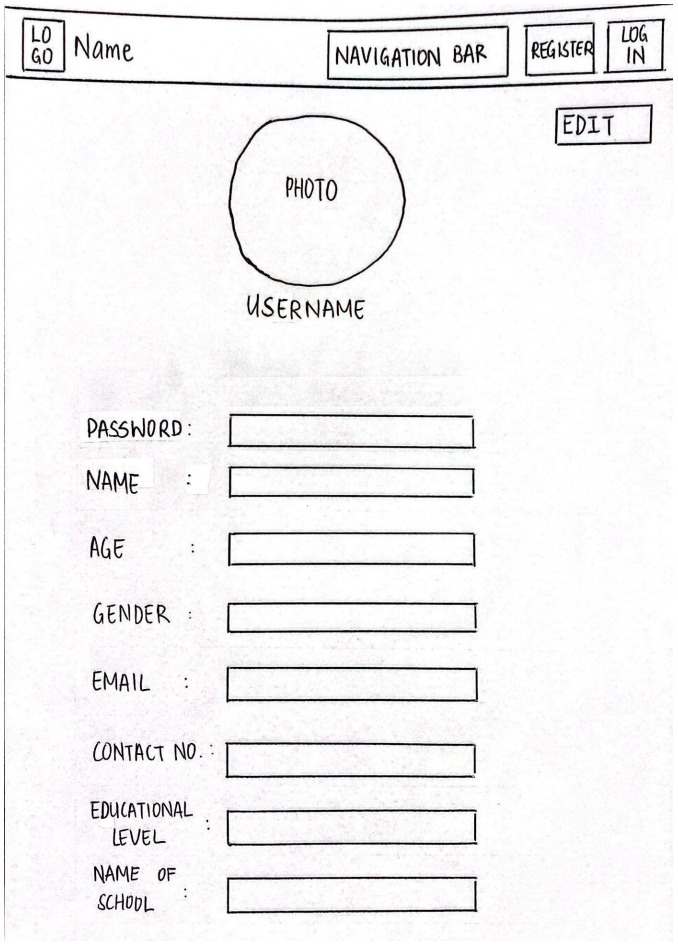
Tips Page

- Suggested tips will be shown on this page
  - hyperlinks will be provided for the user to get more information
- Search bar to search for keywords that the user may be interested in



Forum Page

- An online support group for like-minded individuals to communicate with one another
- User have the option to be anonymous in the forum to protect their own privacy and personal issues
- Users are allowed to add entry to the forum or just read through the entry available in each forum



Profile Page

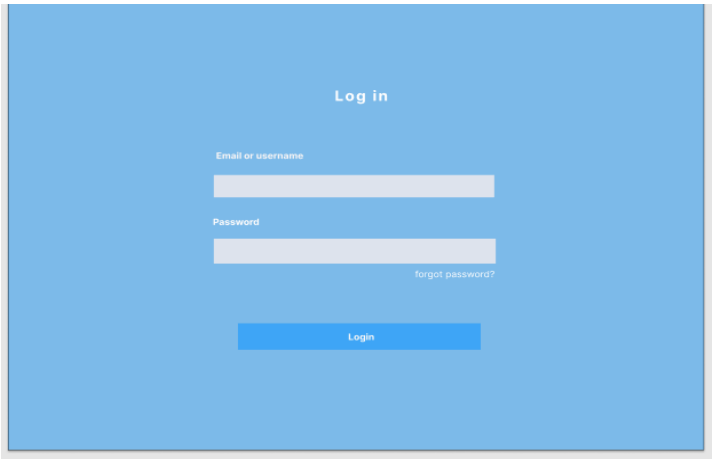
- Allow user to access the current information they have provided during the registration
- Users can update or change any information about them in the profile page

8.1.2 - Medium Fidelity Prototype

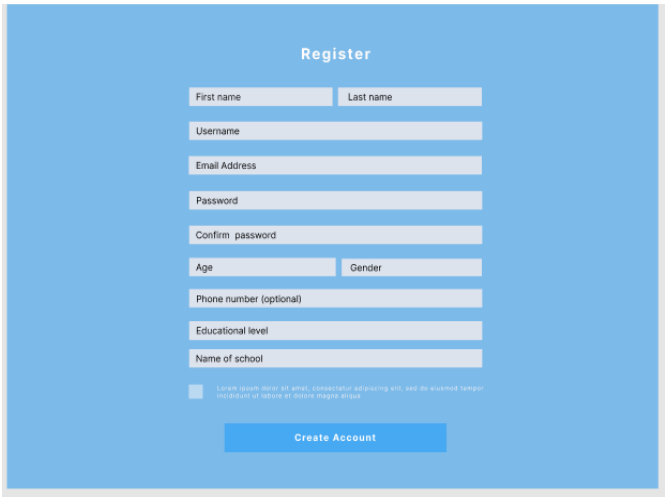
Figma Medium Fidelity Prototype Link:
https://www.figma.com/file/NT8IwOnxPIA0oV29sqldrl/Agile-Prototype?node-id=0%3A1

Using Figma, a collaborative interface design tool, we then converted the sketches into more presentable and legible iterations. This was vital for us to have a clear representation of the application so that it would be easier to get feedback from potential users to make further improvements before moving on to the prototype development. Included below are screenshots of the improved wireframes with the same functions on each page already described in the low-fidelity prototype:

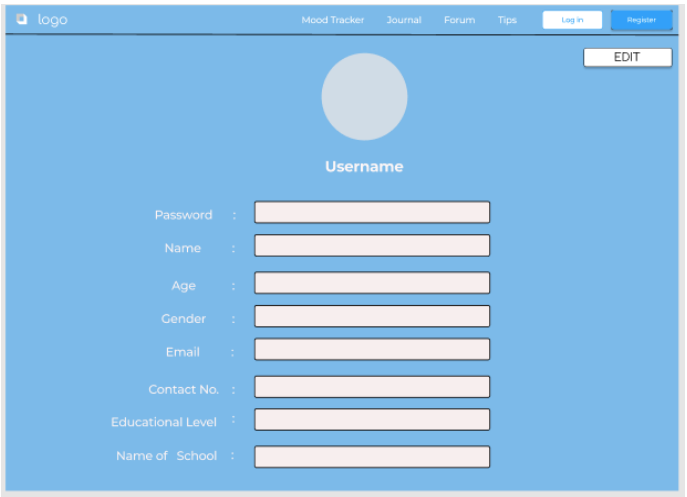
Wireframe 1 - Home Page



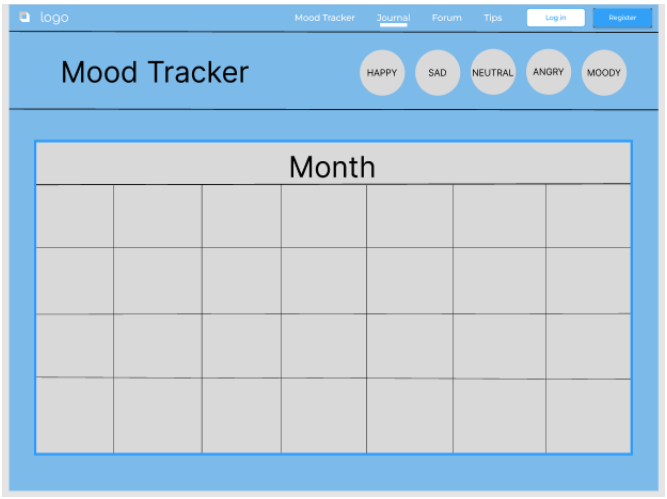
Wireframe 2 - Log in Page



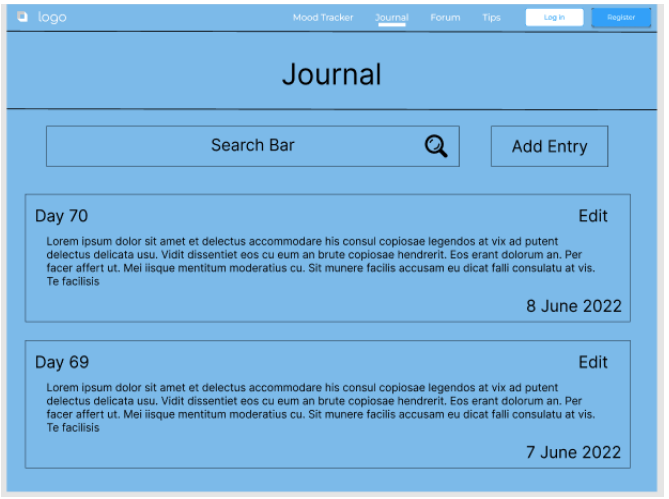
Wireframe 3 – Register Page



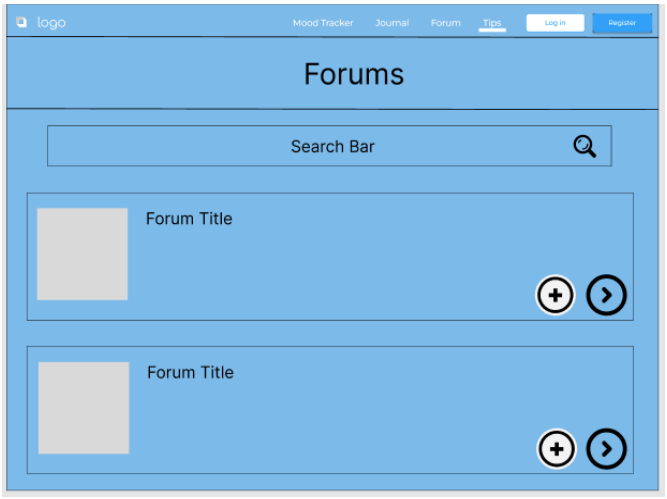
Wireframe 4 - Profile Page



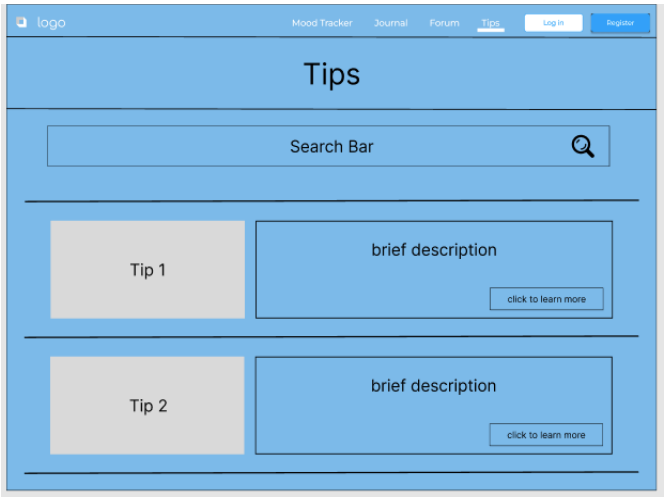
Wireframe 5 - Mood Tracker Page



Wireframe 6 - Journal Page



Wireframe 7 - Forums Page



Wireframe 8 - Tips Page

8.1.3 - High Fidelity Prototype

Figma High Fidelity Prototype Link:
https://www.figma.com/file/NT8IwOnxPIA0oV29sqldrl/Agile-Prototype?node-id=115%3A338

8.2 - GitHub Repo

GitHub Repo Link: https://github.com/Jiaying1216/moodly

8.3 - Final Report Contributions

Report Chapters	Member In-Charge
Background	Shermin
Planning and Research	Shermin
Prototyping and Iteration	Jiaying
Design Specification	Jaztin
System Development	Zaeem
Unit Testing	Nicholas
Conclusion	Team