CFG Final Project

Team members:

Jody Broad
Rada Kanchananupradit
Melissa Long
Georgina Annett
Khadija Warsama

Table of Contents

Introduction

Our self-care tracker is designed to help individuals track and monitor their physical and mental well-being by collecting and tracking data on various health metrics. Our tracker includes an easy-to-use interface, as well as a back-end system to provide valuable insights into the users overall health. Our objectives for this project were to:

- Create a user-friendly interface that allows individuals to easily track and monitor their self-care.
- Collect data on key health metrics such as sleep, exercise, mood, and water intake using an SQL database
- Using HTML, CSS, and Flask to create and connect the Front End to the Back End Python Logic.
- Conduct thorough testing to ensure the reliability and accuracy of our tracker

Throughout this report, we will provide a detailed overview of our project, including our planning process, development methodology, and testing procedures. We will also discuss the challenges we faced and the solutions we implemented to overcome them.

Background

What does it do?

Our app is a self-care tracker that is designed to provide a comprehensive solution for tracking and managing various aspects of users well-being such as exercise, mood and water intake; to encourage and monitor users to look after themselves and to examine the data behind these activities.

How does it work?

- Mood Tracking: Understanding and managing emotions is crucial for overall well-being. Using a selection of emojis, our tracker allows users to track their mood in calendar format, whether their feeling happy, sad, anxious, or any other emotion.
- Sleep Tracking: A good night's sleep is essential for health. Our tracker enables users to monitor their sleep duration and quality.
- Water Tracking: Staying hydrated is key to maintaining optimal health. Our tracker lets users track their water intake and visualise it over time.
- Activity Tracking: Monitoring physical activity is important for a balanced lifestyle. Our tracker allows users to log and track their activity levels
- User-Friendly Interface: Our wellness tracker features a user-friendly interface that makes it easy to navigate and interact with the various features ensuring a seamless user experience whilst also displaying the weather using an API.

What kind of problem does it address?

Our self-care tracker aims to address key challenges and fulfil needs of self-care tracking. We recognised the existing gaps in traditional methods of tracking and monitoring self-care, and thus, developed our solution to overcome these obstacles. Here are some of the problems our wellness tracker fills:

- Accessibility and Convenience: By offering a web-based app, our self-care tracker ensures accessibility and convenience for users. Users can access and utilise the tracker from any device with internet access, making it readily available whenever and wherever users need it..
- Health: Our self-care tracker helps users better manage their physical and mental health by providing an insight into the users patterns and behaviours.
- Cost-Effective Solution: Our web-based application is completely free to use compared to physical self-care tracking alternatives making it environmentally friendly. This cost-effectiveness ensures a sustainable future for our tracker, enabling continuous improvements and updates to benefit our users.

By addressing these challenges and offering innovative features, our self-care tracker allows users to effectively manage their health, providing a convenient, engaging, and efficient solution to meet the users wellness needs.

Specification and design

Our selfcare tracker encompasses both technical and non-technical requirements to ensure a seamless user experience and efficient system functionality. Here are the specifications and design considerations we have implemented.

Technical Requirements:

- User-Friendly Interface: Our program features a user-friendly, intuitive, and accessible interface. It provides a clear and distinctive interactive welcome page for users to log in and track their self-care.
- Visual Enhancements: To cater to different user preferences,use emojis/icons/graphs to accompany the data, helping users better understand and track their wellness. To create a weather display on the homepage to reflect current weather conditions
- Database Integration: The tracker is connected to a database that stores the users data. Users have the option to view their trackable data in an easy to understand calendar format.
- Testing: The system has undergone comprehensive unit and functional testing to ensure its functionality and reliability

Non-Technical Requirements:

1.Documentation

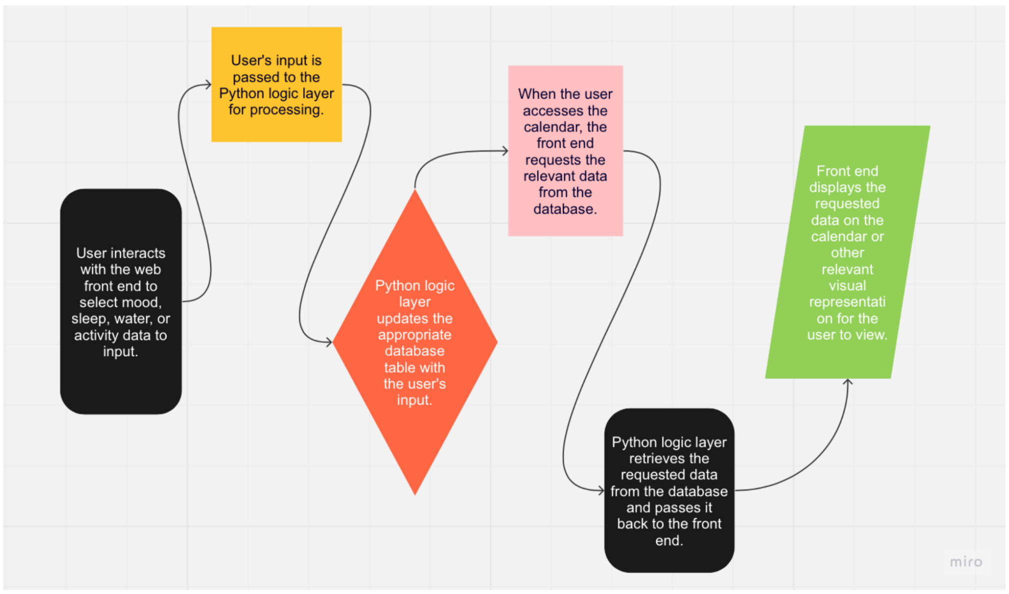
- A detailed report is available to provide deeper insights into the project, including planning, decision-making, and program creation.
- A ReadMe file on GitHub offers clear instructions on how to use the wellness tracker.
- The code is written in an efficient and readable manner, with informative comments to aid future developers.

2. Accessibility:

- Images: Visual elements, such as emojis, are included to assist visually-oriented users in in tracking their self-care
- Free Accessibility: Unlike physical alternatives, our web-based tracker is freely accessible, making it widely available
- Visual accessibility: Web page components have been spaced out clearly with clear contrasting colours to their respective backgrounds. The data tables for example have alternating row colours to allow users to easier identify corresponding data within that row and the table also highlights the current row (mouse cursor hovering over it) in orange for even greater contrast.

3. Design and Architecture:

- The user accesses the wellness tracker through the internet using the Flask application.
- The Flask application displays the HTML content and retrieves data from the API upon user request. The back-end also manages calls to and from the MySQL database.
- The MySQL database stores relevant data of the tracker, enabling effective data management and storage.
- By meeting these technical and non-technical requirements and implementing a robust design and architecture, our wellness tracker provides a user-friendly, accessible, and portable solution for tracking and managing the user's overall well-being.


User interacts with the web front end to select mood, sleep, water, or activity data to input.
User's input is passed to the Python logic layer for processing.
Python logic layer updates the appropriate database table with the user's input.
When the user accesses the calendar, the front end requests the relevant data from the database.
Python logic layer retrieves the requested data from the database and passes it back to the front end.
Front end displays the requested data on the calendar or other relevant visual representation for the user to view.
miro
Initial architecture diagram

Implementation and Execution

Development approach

We knew we wanted to work in an iterative and agile fashion. We split the time we had available into four week long sprints, with the aim of having a long sprint retro (perhaps an hour) meeting once a week; to discuss where we are all at and set priorities for the coming week. We also agreed to a couple of short standup meetings immediately after our CFG teaching to touch base in between while also using the Slack chat and the Miro board for sharing updates, issues, questions etc.

Team Member Roles

Jody	Git repo, Slack group, Miro board setup, back-end python logic (main base code), Flask forms (tracking form, log-in/log out/register), HW2, basic HTML/CSS, Navbar, SQL database, testing
Georgie	Architecture diagram, homework 2, linking to frontend, unit testing
Khadija	Database structure, graph research
Melissa	Calendar and emoji functions
Rada (Back-end & Front-end)	ERD database, main base code, connecting weather API, looking at graphs, HTML and CSS styling
Team efforts	unit and user testing, presentation, final report

We had a productive first meeting where ideas were shared and captured on Miro, and after this first meeting we already had a firm idea that we wanted to do something around the idea of a self-care app.We created a pseudocode of how we wanted the logic to work. Rada and Jody then worked on trying to break this down a bit further into specific functions before we started coding.

Each sprint retro was centred on discussing what we wanted to achieve and dividing the tasks, and this was our approach going forward.

Tools and libraries

- SQLAlchemy was used as the ORM to allow Python to interact with MySQL.
- Flask was used for the web interface
- Wtforms was used to build the form fields in Flask, and also provided validation tools, to ensure that users were making valid inputs.
- Jinja for the HTML templates, displaying data pulled from the database as queries
- Datetime was used to generate today's date for daily records.
- Unittest was used for testing
- Javascript for the emoji calendar
- Matplotlib for graphs - couldn't quite get this working
- WeatherAPI - used to obtain weather data of London location in order to display certain images as website background

Implementation process

Jody began building the base code, she created a working database link, and split the code into separate Model, Create, View and Form files. The Flask instance was generating a basic website and a Flaskform working that would take registration data entered by the user on the website and commit it to the UserInfo table of the database.
While another page on the website, would then display all of the users currently registered. Jody had not been diligent in making regular commits,and broke all of her working code. After many hours trying to resolve the issue, and conscious that the team needed the base code to then branch off to work on features, she reluctantly put everything back into one file as a temporary measure - functional code was better than organised not-working code. She learned an important lesson about regularly committing work to Git and about working iteratively in general.

Once Jody had this basic iteration working, she worked with Rada to get all of the tables for our database constructed, along with the relationships between them which was fairly easy to construct as Rada had created an ERD diagram after the prior meeting to help visualise relationships. The table creation was all done in Python so all tables would be constructed and filled with data automatically, we did slightly tweak our database structure. Jody did a lot of functional user testing to ensure that everything was working properly and being committed to the database as expected. Jody then began working on building the tracking forms; we started off with separate forms for each attribute we wanted to track (Mood, Sleep, Water, Steps), but she quickly realised that it was better from the user perspective for this all to be one form, so built it into one page.

Melissa worked on the tracker calendar to display the mood, water intake, sleep (duration and quality) and steps taken from the tracking form. This required a new view and template to be added to the project files. Initially Melissa considered the possibility of building the calendar from scratch using CSS and custom Javascript. However, as it was out of scope and an inefficient method given time constraints, they researched and utilised fullcalendar.io.

At this point we had a lot of code in one file, which we knew we did not want going forward. Jody spent some time trying to split this out, but ran into issues with circular imports, which Melissa was able to assist in solving.

Georgie did some initial work on using session variables to log in. Jody took this code and expanded upon it and integrated it into all of the pages of the website. Jody set up a registration button and form, a log in button plus form and a logout button. Jody then also got the tracking page to flash up an alert if the user wasn't logged in, and wouldn't let the user submit a record unless they're logged in. A success message flashes when the user registers, logs in or logs out, and a failure message if this is unsuccessful.

After making the suggestion of utilising a weather API during an earlier group meeting, Rada had gone and worked on the weather API - getting a code for today's weather and then using a function, displaying a different background on the website to reflect the current weather conditions. The image URL had to be rendered and displayed in the html files using flask. Jody built a navbar and the basic layout.html template - all the other pages then inherited from this using Jinja and Jody also built queries for pulling data from the database and then used Jinja templating to display this on the website to the user in a sensible way on the pages user_data.html, user_list.html and then my_user_data.html (which will only display the data for the currently logged in user).

Rada also worked on the CSS styling for the website where she created, formatted and updated the html files and the CSS file. These included a variety of elements of the web page such as header, navbar, buttons and footer. This was a challenge as Rada had never worked with html or css files before, but having volunteered to take on this task, Rada spent a bit of time learning the basics and in the end she then went and did the front end styling for the group.

Khadija was tasked with the graphs , but after extensive research felt this was too challenging for her. She then assigned herself the task of the report. Rada then took over on graphs, but also felt it was too challenging considering our time restraints.

We had a lot of other ideas about how to expand our functionality, including adding a virtual pet element (that would have ideally been related to the self care data that the user was entering, encouraging them to look after themselves to look after their pet), and things like targets for amount of water being drunk, steps each day etc, graphical displays of trends over time and many others, but we had to keep the scope more narrow due to time constraints.

Agile development

We used an Iterative approach, with clearly defined goals for each sprint. We had a basic CRUD functional working web app by the end of week one which allowed us to build further functionality from this base. Everyone was working on their features on branches in Git, then doing Pull Requests and other members of the team reviewing the code. Jody and Rada worked particularly well together, helping each other when we had issues with specific bits of code, sharing screens and working through issues. We also did some refactoring, and would have done a lot more if time allowed.

Implementation challenges

Getting database connection working was fine, but getting all the Models (tables) and the create.py file successfully creating the data and then adding to it via the Tracking and Registration forms was quite a challenge - it took a while to determine how to get all of the relationships between tables (in terms of primary and foreign keys) working properly.

Jody found that getting the Flask forms and the website in general working went relatively smoothly. She also had some issues with splitting the code into separate files - issues with circular imports, but Melissa was able to resolve this.

For the weather API, research had to be done to find a suitable weather API to use as it had to have free access (no buying or free trials) and had to give an output that could be used when selecting images based on weather e.g. weather code. From there integration of the API was straightforward, as Rada made a request to the API to get a response in json format and from the response the specific output needed was taken e.g. weathercode from the daily data of the json response body. Rada created another function that took an input which was the code and through multiple statements returned a weather image URL to the main function. Initially everything line of code was within the find_weather function but due to better readability and reusability of the functions, Rada refactored the code so that each function was doing a single procedure which could then be called upon when needed. The body of the response_code function used to be within find_weather but as Rada wanted to test the Exception Error (because at the time was inaccessible as the request response was always 200 and we couldn't find a way to break the request or mock an input as find_weather did not take inputs), Rada went about refactoring it, including making response_code take in the required inputs (request and request response code) and return the required output. Through that Rada was able to create a test for the Exception error. Rada did have difficulty in allowing the HTML pages to render the background weather image but that was soon resolved by linking the background image using flask so that find_weather is run first and then the image URL returned from that is linked to the html file via flask.

A problem encountered during the creation and implementation of the emoji/tracker calendar was the ability to return Json from Flask. Melissa found that it was likely a complicated process that would require installing additional packages like marshmallow and Flask-marshmallow. They were able to work around this by rendering the Json directly into the template within a script tag. Due to the complexity of building the calendar using CSS and custom Javascript, a Javascript library was used instead. This led to the additional challenge of learning the API surface of fullcalendar.io due to old and outdated documentation confusing the process.

For overall styling, initially Rada had some difficulties as the first few html files were written by more than one person and so there was no consistency in structure or naming convention for individual elements. After taking on the task of the web page styling, Rada decided to reformat all the html files for better consistency and readability of the files which made CSS styling a bit easier. This included having more consistent naming conventions and indentations of the code. Given that changes had to be made to the html files and with other team members trying out different styling within the same style.css file when they created their own html files, Rada decided to start the style.css file from scratch and added in CSS code for a variety of elements such as the header, navbar, footer and different sections on different pages. She also went and did some styling for all the forms (e.g. registering, login or tracking forms) and tables (all user data and my user data) and flask messages for better accessibility.

Testing and Evaluation

Unit testing

We intended to have a lot of unit testing for all the small elements of the code and would have liked to be able to do full regression testing but unfortunately we did not get to this stage. We have a suite of unit tests and did a lot of functional user testing as we went along.

Jody wrote tests to check the connection to the database is working, that there is a valid instance of the Flask app, that we have queries from the database returning expected data (required for several of our routes on the website to view tracking data, user data etc). Working together, Rada and Jody created tests for the weather.py to make sure that it was returning the correct picture output based on the weather codes coming from the API as well as testing whether the exception was being raised or not. Georgie has further written tests that test the functionality of the log in and log out code, although unfortunately she didn't quite get these working. Melissa wrote some tests for the calendar functionality.

We did refactor some of our code to make tests easier to run, but some of the code was too complex for us to be able to get sensible tests running correctly.

Functional and user testing

We extensively used functional and user testing throughout, as soon as we had Flask generating the web app it was very easy to ensure by clicking around the website that everything was working and displaying as intended, so we did a lot of this as we went.

We used wtforms to do validation of user input on the website itself, this meant that the website itself was handling things before they had a chance to become issues (e.g. making sure that a valid email address was being entered, that name met a minimum length requirement, that passwords were a suitable length etc).

In the initial stages Jody was also looking at MySQL carefully to ensure that all of the data in the create.py and models.py was creating and populating the data tables correctly.

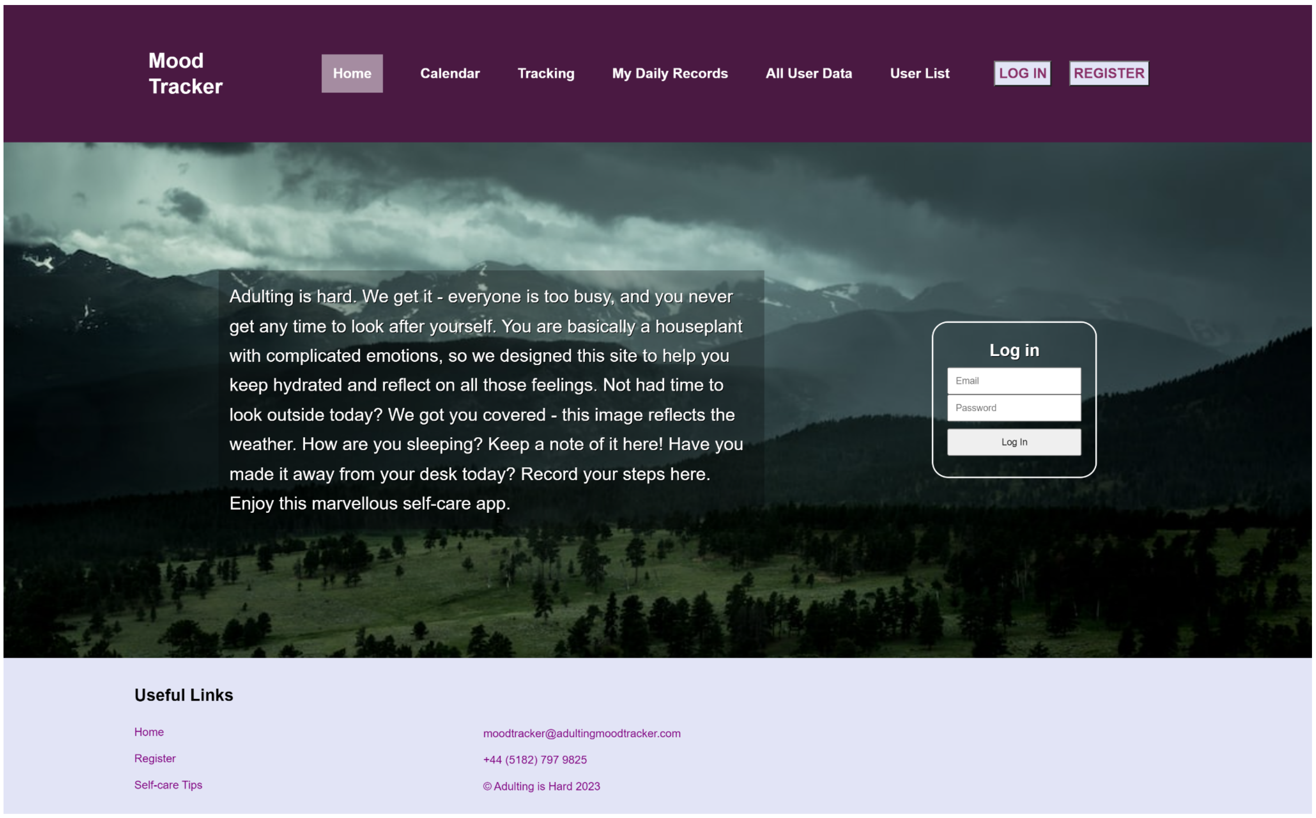
Future Work

- We would like to add a virtual pet element as a fun way of encouraging self care. So users are able to care for the pet by reaching their targets.
- We would like to add detailed graphs so users can visualise and gain insights into their health over time; for users to make informed wellness decisions. This should be used for example to analyse sleeping habits, patterns in mood, activity levels etc.
- We would like to add a convenient water bottle icon that fills up as users log their water consumption, providing a simple and intuitive way to monitor the users hydration levels - Melissa was able to add this feature to the calendar at the end of the project.
- We would like to add an achievements and goal setting portion to our tracker to keep users motivated on their wellness journey. Users can celebrate milestones and accomplishments as they progress towards their health goals by setting personal goals for sleep hours, hydration levels, and more, and track their progress as you work towards them.
- We would also like to incorporate engaging user design(ux) in our interface to make it more engaging and user friendly.
- We would also like to improve our accessibility features by providing more visual elements such as bold fonts, emojis and images.
- Would have loved to do more thorough testing like the login and writing data to the database, but this was too complicated for us to achieve in the time we had, would probably have ideally used something like Selenium for this.
- We would also like to add wider user testing, using a focus group.

Conclusion

We are proud that we achieved the basic functionality that we set out, but also had many ideas for other features that we would have added if time allowed. There are improvements that we could have made in terms of coding, but we did manage to separate the concerns well (in separate logical python files) and kept our functions as simple and small in size as possible. We also used OOP principles including classes and a variety of libraries.

We have a number of functional tests, but not the full coverage we would have liked to achieve. However, we do have a functional web app that writes and reads from a SQL database that was all built in Python including all of the tables and their relationships and fills these with data. Users can Register, Log In, Log Out, add daily tracking records, see a table of all their past records and then can also see a graphical representation of all their records using the calendar function. We also used a weather API to dynamically update the website background.



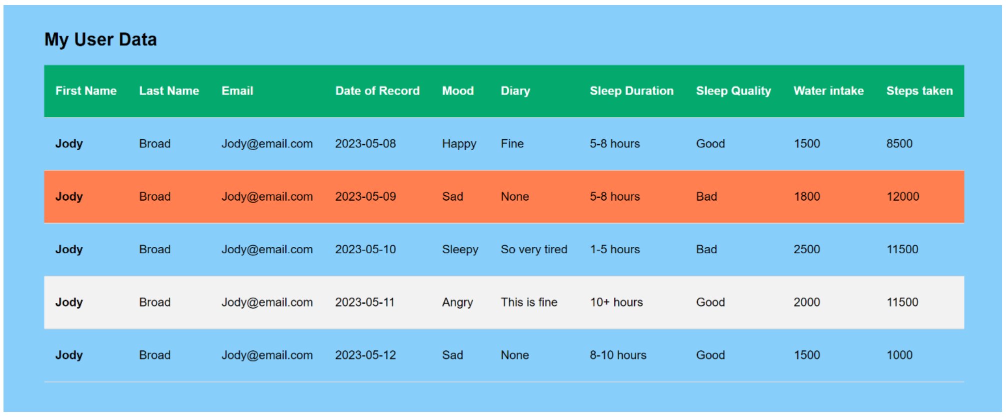
Adulting is hard. We get it - everyone is too busy, and you never get any time to look after yourself. You are basically a houseplant with complicated emotions, so we designed this site to help you keep hydrated and reflect on all those feelings. Not had time to look outside today? We got you covered - this image reflects the weather. How are you sleeping? Keep a note of it here! Have you made it away from your desk today? Record your steps here. Enjoy this marvellous self-care app.

Useful Links

Home
Register
Self-care Tips

moodtracker@adultingmoodtracker.com
+44 (5182) 797 9825
© Adulting is Hard 2023

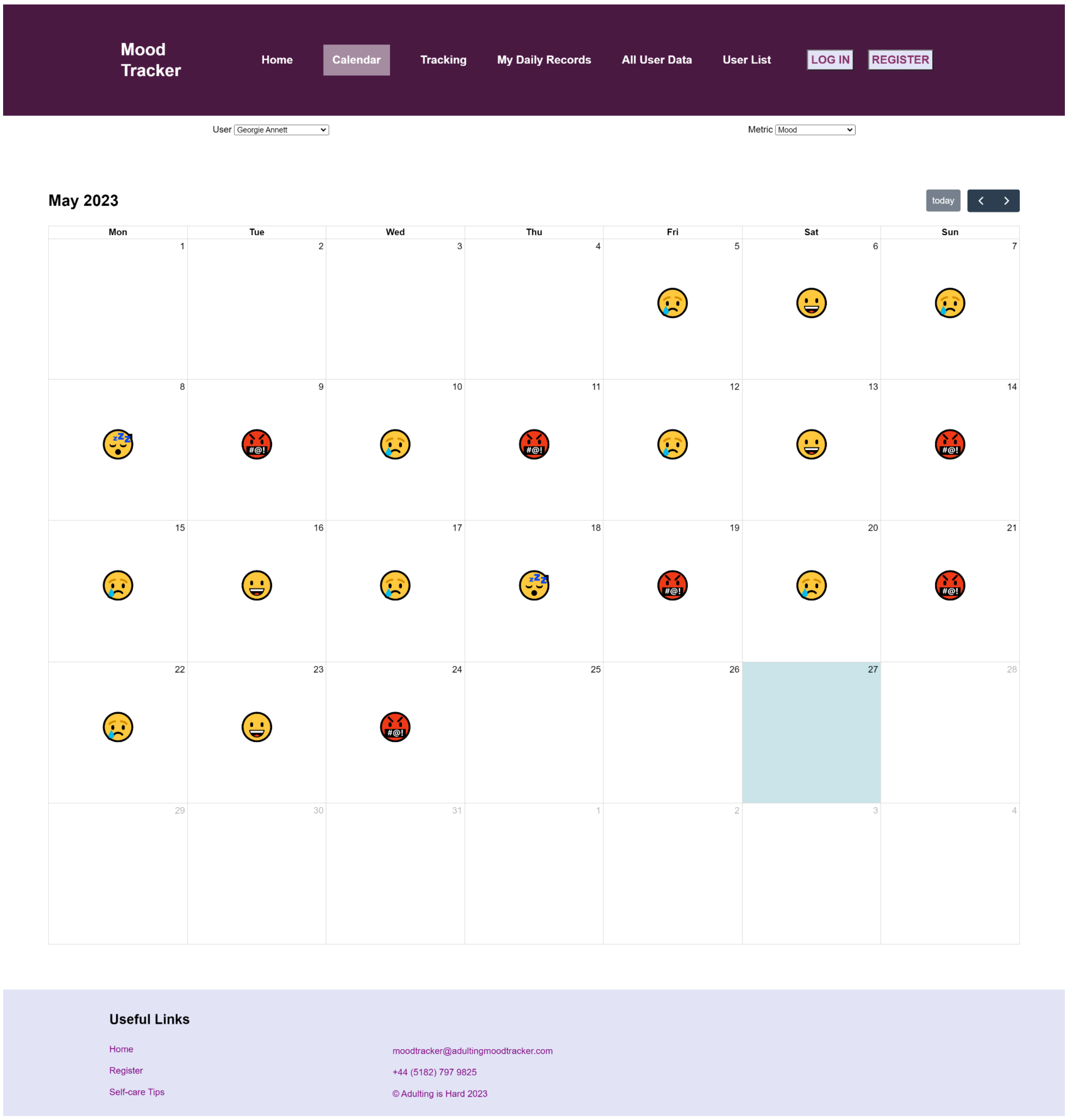
Homepage with dynamically generated background based on weather data from weather API - all elements in view such as header, navbar, body, a form and footer.



My User Data

First Name	Last Name	Email	Date of Record	Mood	Diary	Sleep Duration	Sleep Quality	Water intake	Steps taken
Jody	Broad	Jody@email.com	2023-05-08	Happy	Fine	5-8 hours	Good	1500	8500
Jody	Broad	Jody@email.com	2023-05-09	Sad	None	5-8 hours	Bad	1800	12000
Jody	Broad	Jody@email.com	2023-05-10	Sleepy	So very tired	1-5 hours	Bad	2500	11500
Jody	Broad	Jody@email.com	2023-05-11	Angry	This is fine	10+ hours	Good	2000	11500
Jody	Broad	Jody@email.com	2023-05-12	Sad	None	8-10 hours	Good	1500	1000

Data page with the table of daily records for the logged in user (there is a similar set up for the All User Data page). There is added functionality which is that the specific row gets highlighted in orange when the mouse hovers over it. This was part of the overall CSS styling that Rada produced.



User Georgie Annett

Metric Mood

May 2023

Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5 😢	6 😀	7 😢
8 😴	9 🤬	10 😢	11 🤬	12 😢	13 😀	14 🤬
15 😢	16 😀	17 😢	18 😴	19 🤬	20 😢	21 🤬
22 😢	23 😀	24 🤬	25	26	27	28
29	30	31	1	2	3	4

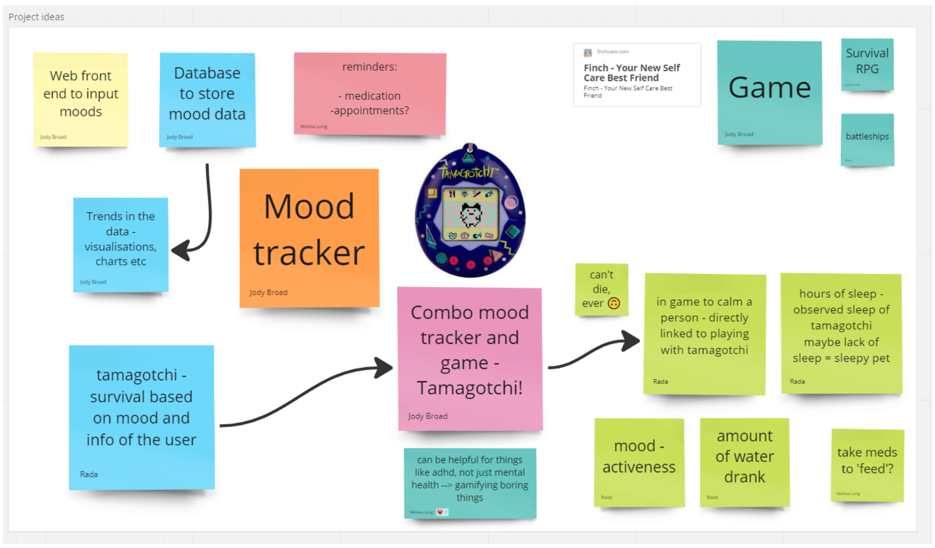
Useful Links

Home
Register
Self-care Tips

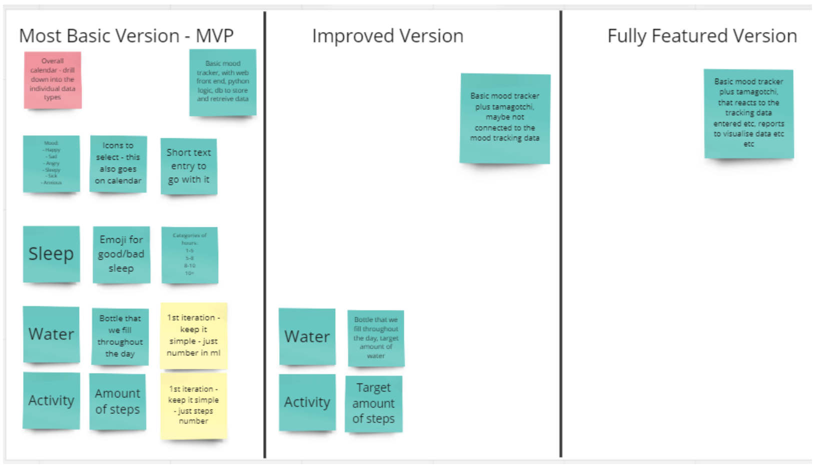
moodtracker@adultingmoodtracker.com
+44 (5182) 797 9825
© Adulting is Hard 2023

Calendar to display mood history using Javascript and the records from the database

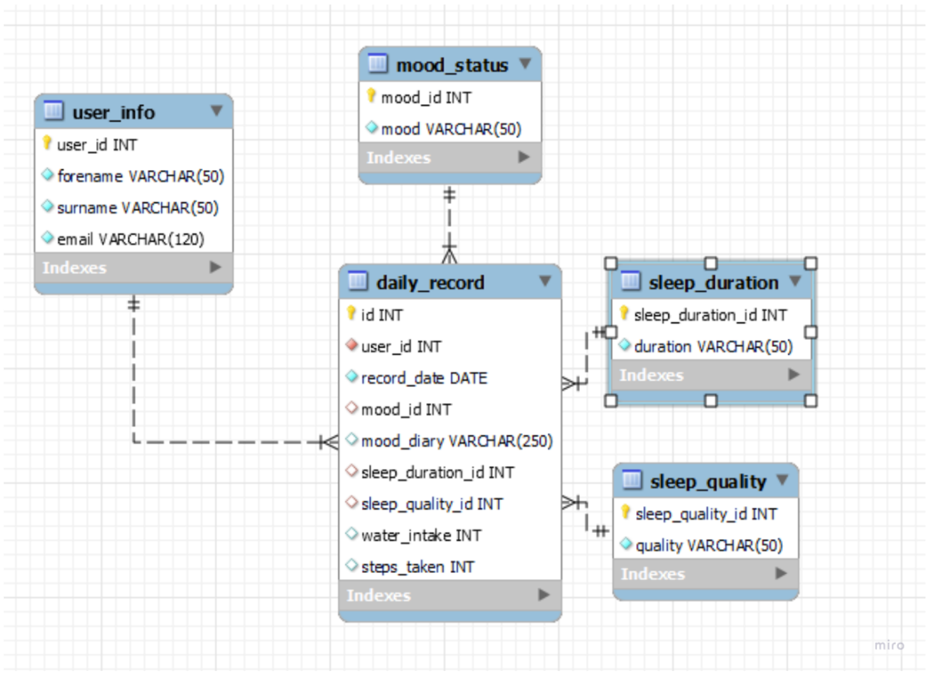
Appendix


Project ideas
Web front end to input moods
Database to store mood data
reminders: - medication -appointments?
Finch - Your New Self Care Best Friend
Game
Survival RPG
battleships
Trends in the data - visualisations, charts etc
Mood tracker
Combo mood tracker and game - Tamagotchi!
can't die, ever
in game to calm a person - directly linked to playing with tamagotchi
hours of sleep - observed sleep of tamagotchi maybe lack of sleep = sleepy pet
tamagotchi - survival based on mood and info of the user
can be helpful for things like adhd, not just mental health --> gamifying boring things
mood - activeness
amount of water drank
take meds to 'feed'?

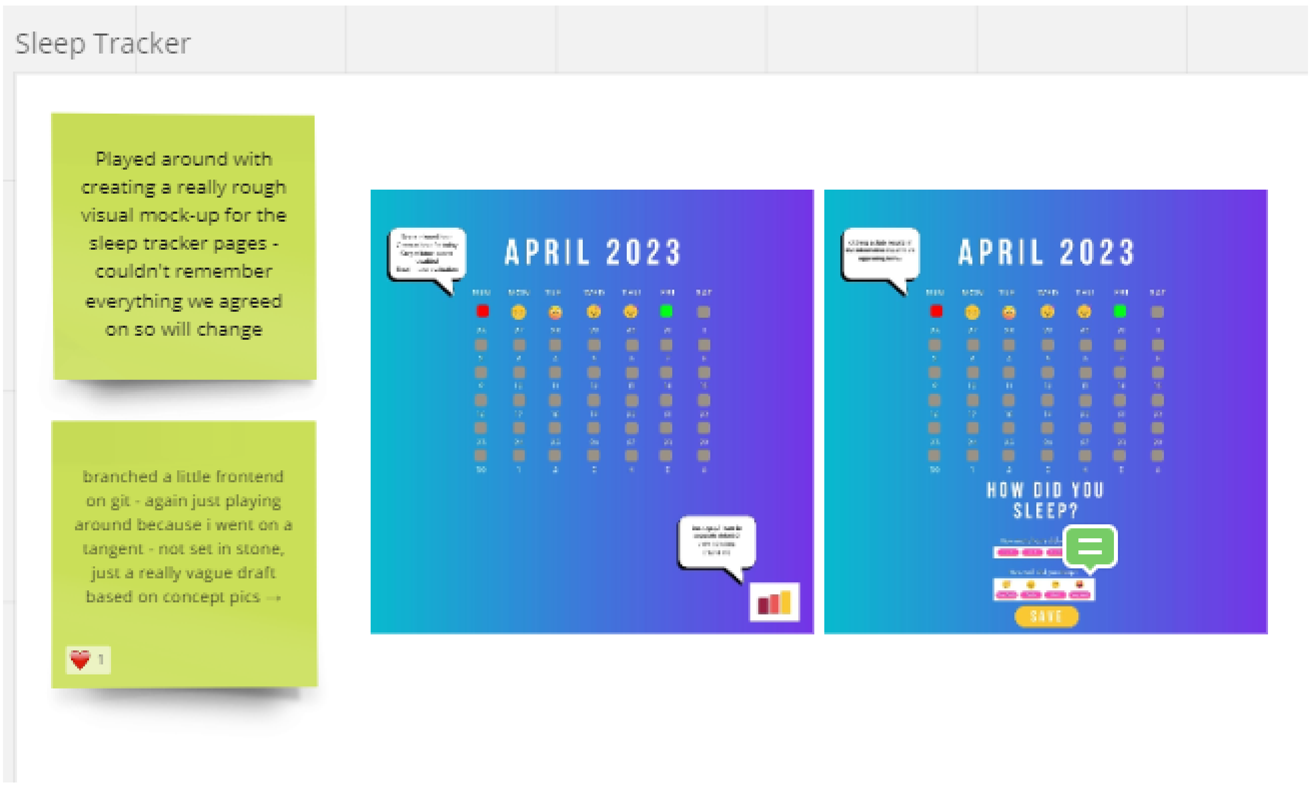
Initial ideas board


Most Basic Version - MVP
Overall calendar - drill down into the individual data types
Basic mood tracker, with web front end, python logic, db to store and retreive data
Mood: - Happy - Sad - Angry - Sleepy - Sick - Anxious
Icons to select - this also goes on calendar
Short text entry to go with it
Sleep
Emoji for good/bad sleep
Categories of hours: 1-5 5-8 8-10 10+
Water
Bottle that we fill throughout the day
1st iteration - keep it simple - just number in ml
Activity
Amount of steps
1st iteration - keep it simple - just steps number
Improved Version
Basic mood tracker plus tamagotchi, maybe not connected to the mood tracking data
Water
Bottle that we fill throughout the day, target amount of water
Activity
Target amount of steps
Fully Featured Version
Basic mood tracker plus tamagotchi, that reacts to the tracking data entered etc, reports to visualise data etc etc

Initial ideas for different iterations



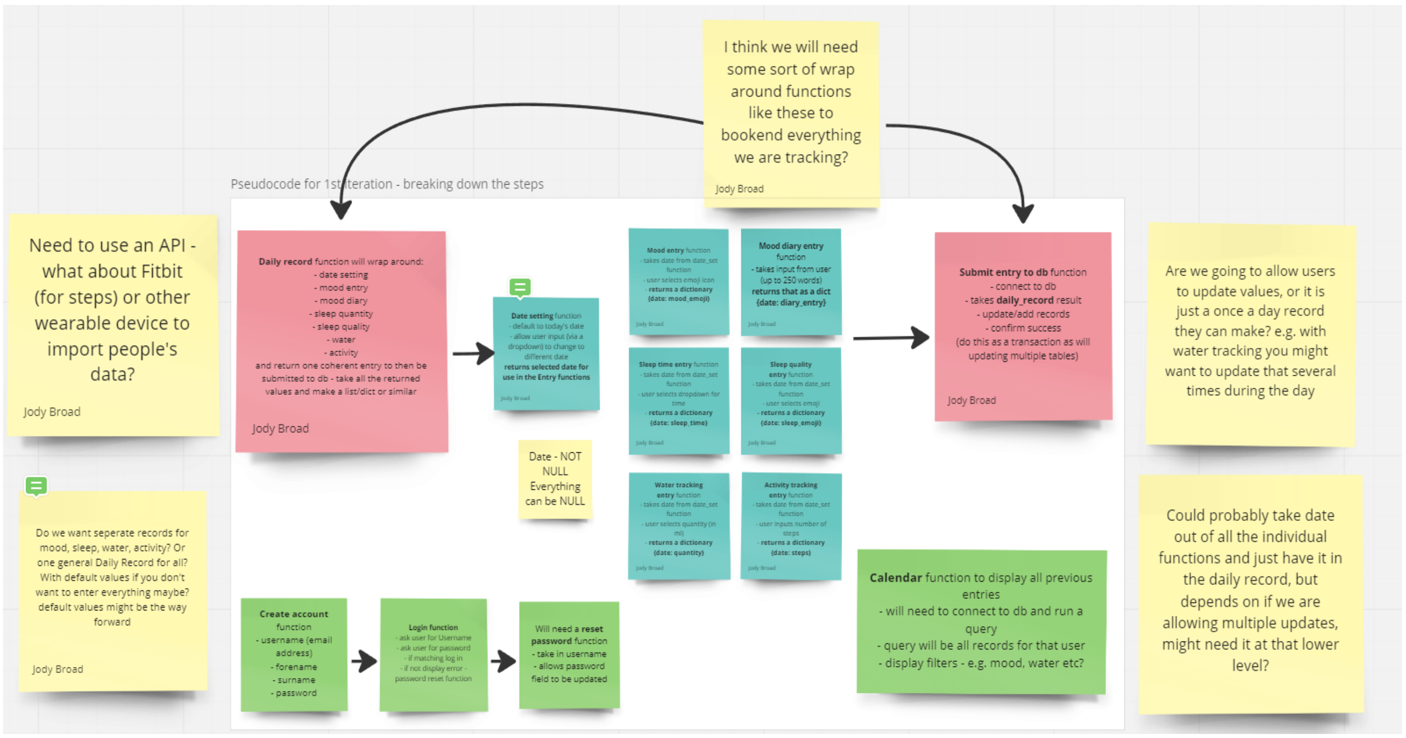
Initial ERD for our SQL database


Sleep Tracker

Played around with creating a really rough visual mock-up for the sleep tracker pages - couldn't remember everything we agreed on so will change

branched a little frontend on git - again just playing around because i went on a tangent - not set in stone, just a really vague draft based on concept pics →

Early mock up for a sleep tracking page showing the calendar idea


Pseudocode for 1st iteration - breaking down the steps

I think we will need some sort of wrap around functions like these to bookend everything we are tracking?
Jody Broad

Need to use an API - what about Fitbit (for steps) or other wearable device to import people's data?
Jody Broad

Do we want seperate records for mood, sleep, water, activity? Or one general Daily Record for all? With default values if you don't want to enter everything maybe? default values might be the way forward
Jody Broad

Daily record function will wrap around:
- date setting
- mood entry
- mood diary
- sleep quantity
- sleep quality
- water
- activity
and return one coherent entry to then be submitted to db - take all the returned values and make a list/dict or similar
Jody Broad

Date setting function
- default to today's date
- allow user input (via a dropdown) to change to different date
returns selected date for use in the Entry functions
Jody Broad

Date - NOT NULL
Everything can be NULL

Mood entry function
- takes date from date_set function
- user selects emoji icon
- **returns a dictionary {date: mood_emoji}**
Jody Broad

Mood diary entry function
- takes input from user (up to 250 words)
returns that as a dict {date: diary_entry}
Jody Broad

Sleep time entry function
- takes date from date_set function
- user selects dropdown for time
- **returns a dictionary {date: sleep_time}**
Jody Broad

Sleep quality entry function
- takes date from date_set function
- user selects emoji
- **returns a dictionary {date: sleep_emoji}**
Jody Broad

Water tracking entry function
- takes date from date_set function
- user selects quantity (in ml)
- **returns a dictionary {date: quantity}**
Jody Broad

Activity tracking entry function
- takes date from date_set function
- user inputs number of steps
- **returns a dictionary {date: steps}**
Jody Broad

Submit entry to db function
- connect to db
- takes **daily_record** result
- update/add records
- confirm success
(do this as a transaction as will updating multiple tables)
Jody Broad

Create account function
- username (email address)
- forename
- surname
- password

Login function
- ask user for Username
- ask user for password
- if matching log in
- if not display error - password reset function

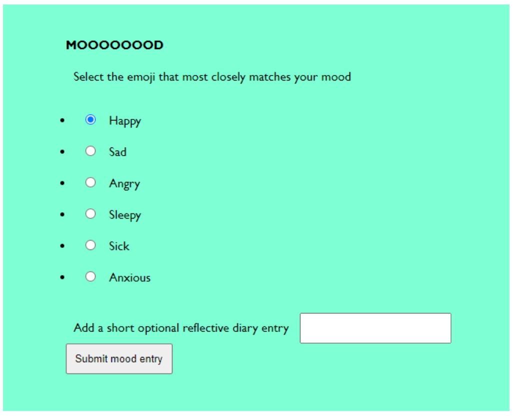
Will need a **reset password** function
- take in username
- allows password field to be updated

Calendar function to display all previous entries
- will need to connect to db and run a query
- query will be all records for that user
- display filters - e.g. mood, water etc?

Are we going to allow users to update values, or it is just a once a day record they can make? e.g. with water tracking you might want to update that several times during the day

Could probably take date out of all the individual functions and just have it in the daily record, but depends on if we are allowing multiple updates, might need it at that lower level?

Initial pseudocode for how the logic of the program works



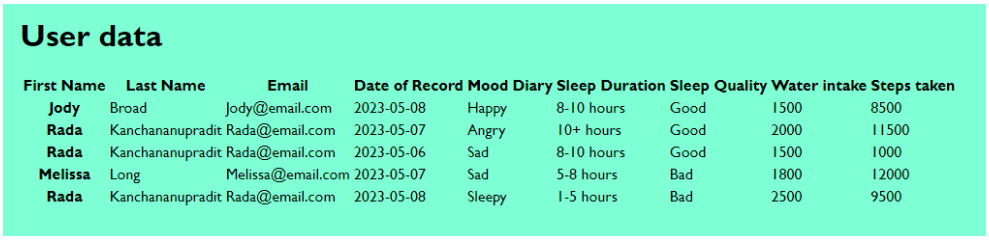
MOOOOOOOD

Select the emoji that most closely matches your mood

- [✓ Happy]
- Sad
- Angry
- Sleepy
- Sick
- Anxious

Add a short optional reflective diary entry

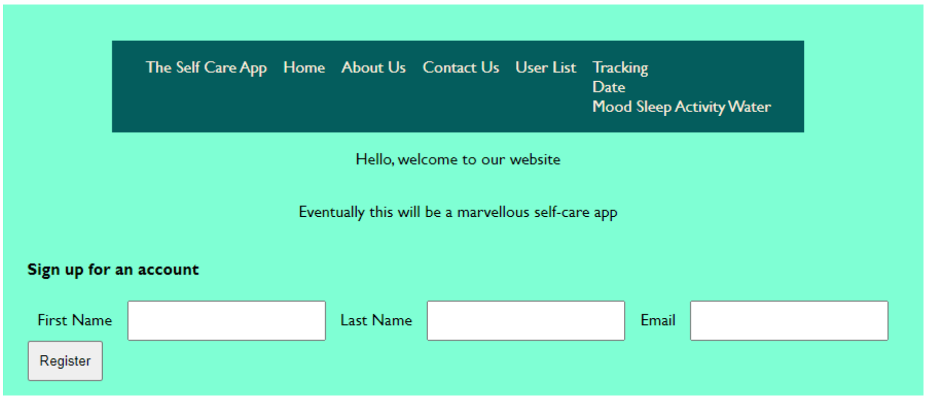
Early iteration of the mood tracking form built by Jody



User data

First Name	Last Name	Email	Date of Record	Mood Diary	Sleep Duration	Sleep Quality	Water intake	Steps taken
Jody	Broad	Jody@email.com	2023-05-08	Happy	8-10 hours	Good	1500	8500
Rada	Kanchananupradit	Rada@email.com	2023-05-07	Angry	10+ hours	Good	2000	11500
Rada	Kanchananupradit	Rada@email.com	2023-05-06	Sad	8-10 hours	Good	1500	1000
Melissa	Long	Melissa@email.com	2023-05-07	Sad	5-8 hours	Bad	1800	12000
Rada	Kanchananupradit	Rada@email.com	2023-05-08	Sleepy	1-5 hours	Bad	2500	9500

Early iteration of the display all user data web page built by Jody - this is pulling and joining multiple tables from the database



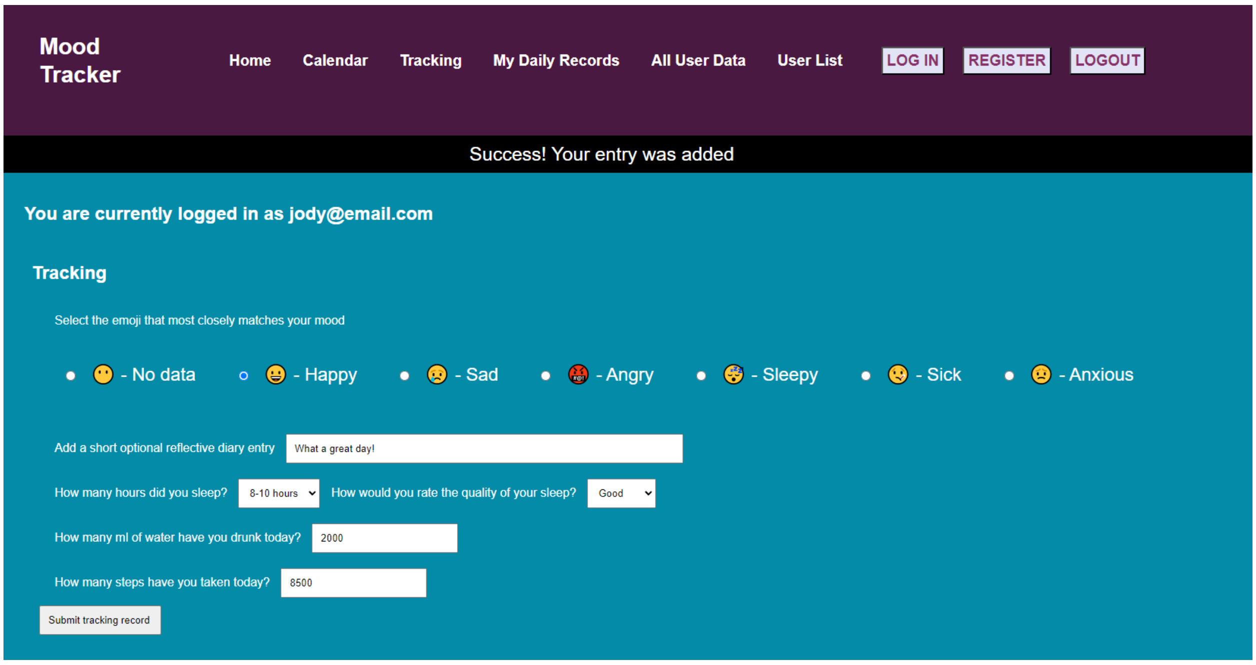
Hello, welcome to our website

Eventually this will be a marvellous self-care app

Sign up for an account

First Name Last Name Email

Early iteration of navbar built by Jody - we later decided to combine all tracking pages into one



Success! Your entry was added

You are currently logged in as jody@email.com

Tracking

Select the emoji that most closely matches your mood

- 😶 - No data
- 😀 - Happy
- 😞 - Sad
- 😡 - Angry
- 😴 - Sleepy
- 😟 - Sick
- 😟 - Anxious

Add a short optional reflective diary entry: What a great day!

How many hours did you sleep? 8-10 hours How would you rate the quality of your sleep? Good

How many ml of water have you drunk today? 2000

How many steps have you taken today? 8500

Tracking form used to enter data. The functionality was built by Jody. The formatting of this page was done by Rada.