­­­­­

AS91906 Programming

Use complex programming techniques to develop a computer program.

Eric van Ginkel

L3 DTS­

Contents

[Project Overview 3](#_Toc172490443)

[1.1 Introduction 3](#_Toc172490444)

[1.2 Problem Summary 4](#_Toc172490445)

[1.3 Personal motivation 6](#_Toc172490446)

[1.4 Target User Group(s) 8](#_Toc172490447)

[1.5 Summary of Functions 10](#_Toc172490448)

[Research 12](#_Toc172490449)

[2.1 Similar Projects 12](#_Toc172490450)

[2.2 Online resources 14](#_Toc172490451)

[Videos: 14](#_Toc172490452)

[Documentation. 16](#_Toc172490453)

[Specifications 18](#_Toc172490454)

[3.1 Program Functions 18](#_Toc172490455)

[3.2 User Interface (UI) 21](#_Toc172490456)

[3.3 Help Documentation 24](#_Toc172490457)

[3.4 Permanent Data Storage 27](#_Toc172490458)

[3.5 Hardware and Software Requirements 31](#_Toc172490459)

[Design Documentation 33](#_Toc172490460)

[4.1 Interface Design 33](#_Toc172490461)

[4.2 Program Flow 38](#_Toc172490462)

[4.2.1 Flowcharts 41](#_Toc172490463)

[4.2.2 Pseudocode 43](#_Toc172490464)

[4.3 Class Design (UML Diagrams) 45](#_Toc172490465)

[Development Process 47](#_Toc172490466)

[5.1 Iterations and Changes 47](#_Toc172490467)

[5.2 Code Comments 93](#_Toc172490468)

[5.3 Separation of UI from Backend 95](#_Toc172490469)

[5.4 Inter-Code Communication 98](#_Toc172490470)

[5.5 Good General Techniques 100](#_Toc172490471)

[5.6 Querying and Manipulation of Data 102](#_Toc172490472)

[5.7 Defensive Programming 105](#_Toc172490473)

[Testing 107](#_Toc172490474)

[6.1 Functional Testing / Test plan 107](#_Toc172490475)

[6.2 Test Results 110](#_Toc172490476)

[Technical Documentation 113](#_Toc172490477)

[7.1 Externally Sourced Code 113](#_Toc172490478)

[7.2 Advanced Techniques 121](#_Toc172490479)

[End user considerations: 129](#_Toc172490480)

[Relevant implications: 131](#_Toc172490481)

[Evaluation: 133](#_Toc172490482)

[GitHub repository: 135](#_Toc172490483)

# Project Overview

## 1.1 Introduction

I am a digital technologies student studying at NCEA Level 3. My aim is to create a suitable complex computer program or system that meets a desired purpose. This project is part of my coursework and will be assessed based on its complexity, functionality, and alignment with the given specifications. Throughout this project I will be continuously improving, iterating and refining my project. With the hope that by the end of the project I will have a impressive project that reaches excellence level.

I have experience coding in various programming languages, including Python, HTML, CSS, and have dabbled in other languages. My project uses these skills to develop a web application using Python and Flask. The purpose of this web application is to help users track their daily protein and calorie intake, set dietary goals, and monitor their progress over time. This project not only demonstrates my technical abilities but also addresses a practical need in the health and fitness domain.

## 1.2 Problem Summary

The problem I want to face is mainly to do with fitness and health. The New Zealand health survey 2020/2021 found that around one in three adults aged 15 and above were classified as obese.

Source: <https://www.health.govt.nz/nz-health-statistics/health-statistics-and-data-sets/obesity-statistics#:~:text=Adult%20obesity%20statistics,%25)%2C%20but%20not%20for%20men>

This is a massive amount, and I want to help tackle this massive challenge. In todays fast paced world, having a healthy lifestyle can be really difficult. Especially for busy people. A good diet and proper nutrition are crucial components of a healthy lifestyle. Many people do not count their calories or track their protein intake. A high protein intake boosts metabolism, reduces appetite, and affects several weight-regulating hormones. Without a reliable method to track or monitor dietary intake, individuals easily succumb to the societal norm of being obese or unhealthy.

The primary objective of this project is to develop a web-based application that helps users effectively track their daily protein and calorie intake, set personalized dietary goals, and monitor their progress over time. This application aims to address several key issues faced by individuals trying to manage their nutrition:

1. **Lack of Awareness:** Many people are unaware of their daily protein and calorie intake, which makes it difficult for them to make informed dietary decisions. By providing a platform to log meals and track nutrient consumption, users can gain better insights into their eating habits.
2. **Goal Setting and Monitoring:** Setting realistic and achievable dietary goals is essential for maintaining motivation and achieving long-term health benefits. This application allows users to set daily protein and calorie goals and provides visual feedback on their progress, helping them stay on track.
3. **Data Visualization:** Raw data can be overwhelming and difficult to interpret. The application includes features for visualizing protein and calorie intake through charts and graphs, making it easier for users to understand their dietary patterns and make necessary adjustments.
4. **User-Friendly Interface:** Many existing nutrition tracking tools are complex and not user-friendly, deterring people from using them consistently. This project focuses on creating an intuitive and accessible interface that encourages regular use and engagement.
5. **Personalization:** Dietary needs vary widely among individuals based on factors such as age, gender, activity level, and health goals. The application allows for personalization to cater to individual dietary requirements, ensuring that the recommendations and feedback are relevant and effective.
6. **Data Security and Privacy:** With increasing concerns about data privacy, it is crucial to ensure that users' dietary information is stored securely. The application employs robust security measures to protect user data and maintain confidentiality.

This project will hopefully showcase my skills in Python and Flask for backend development, as well as HTML and CSS for frontend design. It will demonstrate my ability to create a complex computer program that solves a real-world problem and provides tangible benefits to its users.

## 1.3 Personal motivation

This project is very personal to me, I am a weightlifter and gym enthusiast. I go to the gym at least 4 times a week, and anyone who knows anything about weightlifting, body building, or resistance training knows that the hardest part about making progress is not putting in the work at the gym but rather keeping consistent and maintaining a healthy and suitable diet to help you reach your goals. Personally, I know how hard this is, I meal prep once a week when I can and still find it incredibly difficult to stay on track with my calorie or protein goals, it Is hard to count or keep track of or let alone eat right.

I have talked to many of my friends who go to the gym with me and even classmates. I am not alone in this issue. In fact, I am so invested in this topic that I have even collaborated with some friends to create my own business to also help combat this issue.

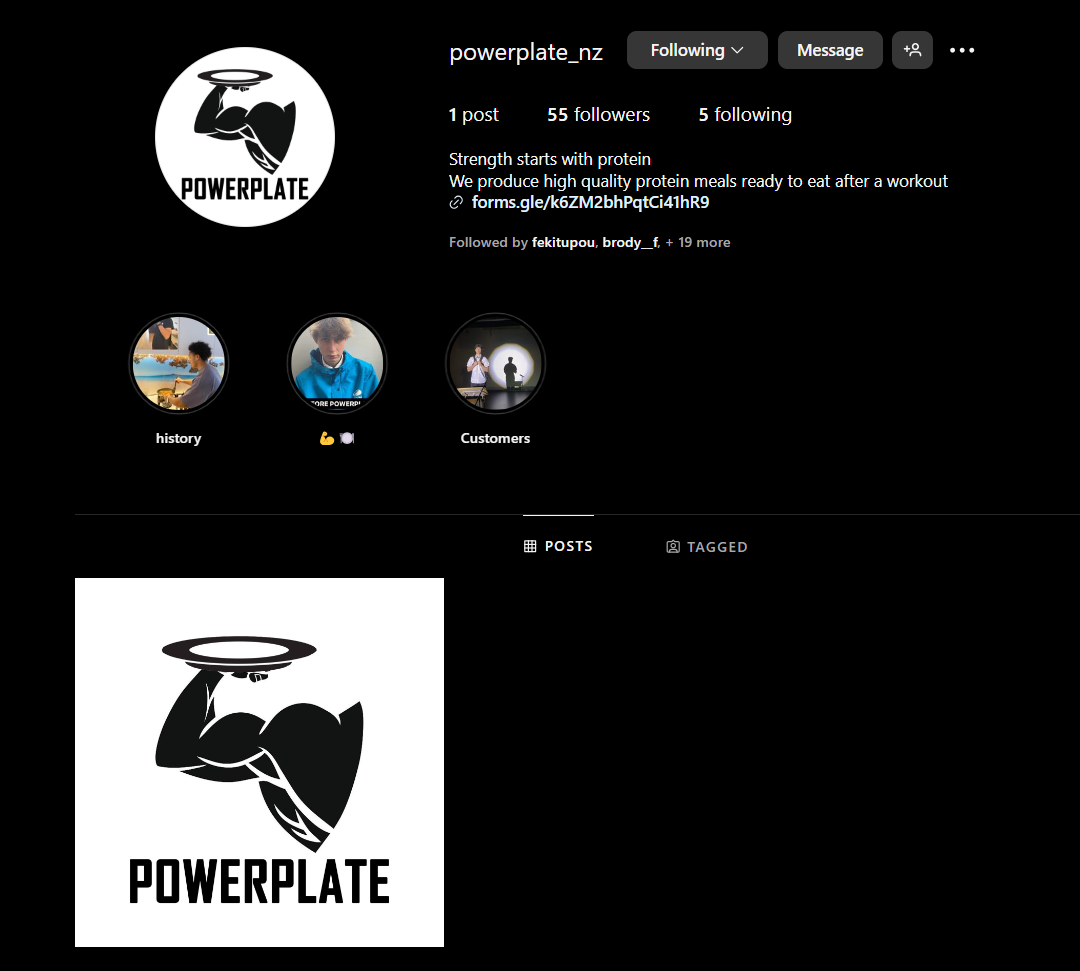
PowerPlate (MY BUSINESS):

  
PowerPlate is a forward-thinking meal preparation service based in Long Bay, dedicated to revolutionizing the way gym-goers and fitness enthusiasts approach nutrition. Recognizing the challenges of maintaining a nutritious diet tailored to athletic lifestyles, PowerPlate offers a unique solution: high-quality protein meals that are prepared in advance.

Our mission is to produce fresh, healthy, and protein-rich meals that cater to individual dietary needs, including allergies, taste preferences, and specific athletic regimens. With sustainability at our core, we use eco-friendly packaging to reduce environmental waste, ensuring that every aspect of our service aligns with our commitment to health and the environment.

PowerPlate stands out in the market by offering both custom and fixed menu options, designed to support various fitness goals, whether it's cutting, bulking, or maintaining. Our target market includes students aged 14-19, a demographic that actively engages in fitness activities but often lacks the time and resources to prepare balanced meals.

With a dedicated team passionate about fitness and nutrition, PowerPlate is set to become a trusted name in the meal prep industry, combining convenience, quality, and affordability.



My business Is very serious about meal preparation and getting people fit and healthy, I hope that this project can approach this issue as well in a different way. I want by the end of this project to be able to not only be able to use it for my self but to also share it with the world and similar minded people. I hope the standard of this project can help me reach these dreams.

## 1.4 Target User Group(s)

I understand that I am a subset of a wide range of individuals who face the same problem as me. I am a fitness enthusiast but that does not mean my project is limited to only people with the same goals and motivations as myself. My project has many use cases, and this will go over my target user demographics.

The candidates who could be interested in this project consist of individuals who are health-conscious and actively seeking to manage their nutrition for whatever personal reasons. These users can be categorized into distinct groups:

1. Fitness Enthusiasts and Athletes:
   * Motivation: Fitness enthusiasts and athletes are often focused on optimizing their performance, building muscle, and maintaining a healthy body composition. Accurate tracking of protein and calorie intake is essential for them to meet their fitness goals and improve their performance.
   * Needs: This group requires precise tracking of macronutrients, personalized dietary recommendations, and progress monitoring tools. They benefit from visual data representations to analyze their dietary patterns and make necessary adjustments to their nutrition plans.
2. Individuals Seeking Weight Management:
   * Motivation: Many people aim to lose, gain, or maintain weight through diet management. Monitoring calorie intake is crucial for these individuals to achieve their weight management goals.
   * Needs: Users in this group need an easy-to-use interface to log their meals and track their daily calorie intake. Personalized goals and feedback are important to keep them motivated and ensure they are on the right track. Visual progress indicators help them stay committed to their weight management plans.
3. Health-Conscious Individuals:
   * Motivation: Some users are generally health-conscious and seek to maintain a balanced diet to promote overall well-being and prevent health issues. They may not have specific fitness or weight goals but want to ensure they are consuming adequate nutrients.
   * Needs: This group benefits from a straightforward application that helps them log their meals and provides insights into their daily protein and calorie intake. Educational content and tips on balanced nutrition can be valuable for them.
4. Busy Professionals:
   * Motivation: Busy professionals often struggle to find the time to prepare and track healthy meals due to their hectic schedules. They need a convenient tool to manage their nutrition on the go.
   * Needs: An intuitive and time-efficient interface is crucial for this group. Features like quick meal logging, easy access to nutritional information, and goal-setting tools help them stay on top of their dietary intake despite their busy lifestyles.
5. Individuals with Specific Dietary Needs:
   * Motivation: Some users have specific dietary requirements due to medical conditions, allergies, or personal preferences (e.g., vegan, vegetarian, gluten-free). Accurate tracking of nutrient intake is essential to ensure they meet their dietary needs.
   * Needs: This group requires customizable settings to accommodate their specific dietary restrictions and preferences. The application should allow them to filter and log foods based on their dietary requirements and provide relevant nutritional information.
6. Students:
   * Motivation: Students, especially those living away from home for the first time, often face challenges in managing their diet. They need guidance and tools to maintain a balanced diet amidst their academic and social commitments.
   * Needs: A user-friendly interface with educational resources on nutrition can help students develop healthy eating habits. Meal planning and tracking features assist them in managing their nutrition effectively.

## 1.5 Summary of Functions

My Project as mentioned previously is a web based platform / tool to help users track their calories and protein, this will need many functions to be efficient and useful:

1. User Authentication:
   * Function: Secure user registration and login.
   * Description: Users can create an account by providing their email, username, and password. Existing users can log in using their credentials. This ensures that user data is private and accessible only to authenticated users.
   * Purpose: To provide a secure and personalized experience for each user.
2. Meal Logging:
   * Function: Log meals and nutritional information.
   * Description: Users can log their meals by entering details such as meal description, protein content (in grams), and calories. The logged meals are stored in the database and can be viewed later.
   * Purpose: To keep track of the user's daily protein and calorie intake.
3. Goal Setting:
   * Function: Set daily protein and calorie goals.
   * Description: Users can set personalized daily goals for protein and calorie intake. These goals are used to provide feedback on their dietary progress.
   * Purpose: To help users set and achieve their dietary targets.
4. Progress Monitoring:
   * Function: Monitor daily and historical progress.
   * Description: The application displays visual feedback on the user's progress towards their daily goals through charts and graphs. Users can view their daily intake and compare it against their goals.
   * Purpose: To provide insights into the user's dietary habits and help them stay on track.
5. Data Visualization:
   * Function: Visualize nutrient intake data.
   * Description: The application includes various charts and graphs to visualize the user's protein and calorie intake. This includes pie charts, bar graphs, and line charts that show intake over time.
   * Purpose: To make it easier for users to understand their dietary patterns and make informed decisions.
6. User Profile Management:
   * Function: Manage user profile and settings.
   * Description: Users can update their profile information, including their email, username, and password. They can also update their dietary goals as needed.
   * Purpose: To allow users to maintain and update their personal information and preferences.
7. Dashboard:
   * Function: Centralized dashboard for quick access.
   * Description: The dashboard provides an overview of the user's current day's intake, progress towards goals, and quick access to other features such as meal logging and goal setting.
   * Purpose: To provide a user-friendly interface for managing and accessing the application's features.
8. Analytics:
   * Function: Detailed dietary analytics.
   * Description: Users can access detailed analytics that show trends and patterns in their dietary intake over time. This includes historical data and insights into meal types and nutrient distribution.
   * Purpose: To help users analyze their long-term dietary habits and make necessary adjustments.
9. Help and Support:
   * Function: Provide help and support resources.
   * Description: The application includes a help section with resources and guides on how to use the app, as well as tips for maintaining a healthy diet.
   * Purpose: To assist users in effectively using the application and provide valuable dietary information.
10. Data Security:
    * Function: Ensure the security and privacy of user data.
    * Description: The application implements security measures to protect user data, including encryption of sensitive information and secure database storage.
    * Purpose: To ensure user data is protected and privacy is maintained.

These functions collectively provide a comprehensive solution for users to manage their protein and calorie intake, set and achieve dietary goals, and maintain a healthy lifestyle. The application is designed to be user-friendly, secure, and adaptable to individual needs, hopefully to a standard that not only reaches academic excellence but also to a professional use case.

# Research

## 2.1 Similar Projects

I am planning to use the Flask framework with python as the backend of this project and HTML and CSS as the front end. This is simple enough and not rocket science. There are thousands of projects online of people using this framework to create professional or personal projects. In terms of similarity, I am not the only one in the world to have had this idea of tracking protein and calories using an app / web-based application. So, there are projects out there that are probably better or similar to mine. Never less, this does not phase me as I am committed to create an application to help others and myself. This not only teaches me more programming skills but also project management.

Projects like mine:

Building a fitness tracking dashboard with python by Fernando Rodriguez

<https://rodriguezanton.com/building-a-fitness-tracking-dashboard-with-python-pt-3-flask-and-deployment/>

This project was an interesting read and gave me some insight into what needs to be added to my project and what other people think needs to be on theirs. In terms of programming, this article went heavily into detail about the programming involved. I learned a lot of useful information and tools to help create a sleek and responsive user interface, like how he used bootstrap.

Fitness tracker built with python and Flask by Nikolay Pomytkin

<https://github.com/Nikolay-Pomytkin/Gym-Me>

This GitHub repository was useful but a bit of a step ahead to where I am in my project now. It is useful to see how he set up his file layout and overall code, but understanding what is going on is not so simple. I still learned a lot from this project overview, however. Like the manifest and robot’s files that mean he can run his project as a PWA This file provides metadata about the web application, allowing the app to be installed on a user's device like a native app. The manifest. Json file includes information such as the app's name, icons, theme colours, and start URL.

Calorie Counter and Fitness Tracker by MyFitnessPal

https://www.myfitnesspal.com/

A phone with a screen showing a fitness app

Description automatically generated

On a different more professional note, instead of looking at personal projects I have researched into a professional company’s platform for fitness tracking. They are very well known and can teach me a lot about the user interface. Their app has many features such as:

* Food and Exercise Logging.
* Log Weight/Measurements, and View Progress.
* Create Your Own Foods, Meals, and Recipes.
* Ability to View Macronutrients.
* Ability to Customize Calorie Goal and Macronutrient Goals by Percentage.
* Ability to Share and View Your Diary with Others.
* Link to Partner Apps.

These features are interesting and give me a great idea into what I should use in my project. I really like the idea of being able to share and view your Diary with others, I think this is a great way to get friends to help motivate you when you have not reached your goal or also a personal motivator because if you can see that your friends have met their goal that will make you also want to meet your own goal.

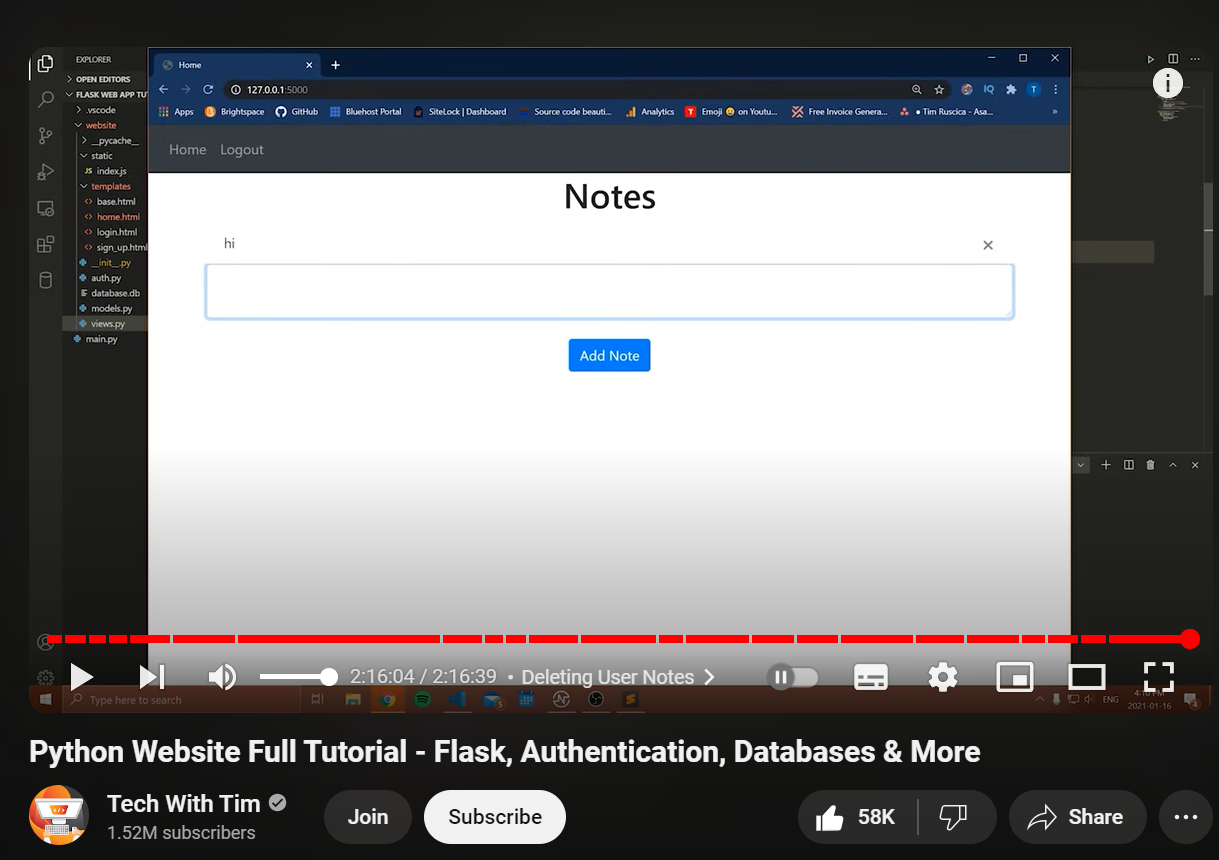
## 2.2 Online resources

There are many resources online that can be beneficial to this project, though I am tying to keep this project as unique and induvial as possible to help me refine my coding ability, I also need help to learn the Flask framework and possibly python functions and features.

### Videos:

To start with I watched this video to introduce me into the Flask framework and how I could set up my site.

<https://youtu.be/dam0GPOAvVI?si=huqb4sJROW3dSjRk>



Video description:

In this video, I'm going to be showing you how to make a website with Python, covering Flask, authentication, databases, and more. The goal of this video is to give you what you need to make a finished product that you can tweak, and turn into anything you like. We're going to also go over how you create a new user's account, how you store those in a database, how to log into those user accounts, how to log out of them, and how you associate that information with a specific user.

What I learnt:

I can not begin to express how insanely useful this video was. I learned so much in so little time. He created a project that allowed users to take notes and store them. Simple enough, but the resemblance to my own project is massive. I am also storing information and using it to display to the user.

One of the key takeaways was understanding Flask routes handling. I learned how to define routes in Flask to handle different parts of the web application, including how to set up URLs to render templates and manage user interactions. The tutorial provided clear explanations on how to structure these routes effectively.

Another critical area of learning was the database aspect, particularly using Flask-SQLAlchemy. The video introduced me to the Flask-SQLAlchemy tool and covered how to set up a database, create tables, and establish relationships between them. I learned how to perform CRUD (Create, Read, Update, Delete) operations on the database, which is essential for managing user data in my project.

Additionally, the video touched on user authentication, explaining how to create new user accounts, store user credentials securely, and implement login and logout functionality. This section was particularly relevant as it aligns closely with the user management features in my application.

However, the tutorial also highlighted areas where I need further research and practice. While the routes and views handling made sense in the context of the tutorial, I realized I need to deepen my understanding of these concepts to apply them confidently in my project. Specifically, I need to explore more complex routing scenarios and learn how to handle different types of user interactions more efficiently.

But overall, I cannot emphasize enough how good this video is, clear and simple but effective. Learned a lot of programming skills and applicable information.

Secondly, I wanted to get a deeper understanding into the Database framework I am going to use so I found this useful video online delving into the functionality and use cases of SQL Alchemy.

<https://youtu.be/aAy-B6KPld8?si=hD3OWwZHkJOCgaTO>



Video description:

In this video the presenter goes into detail about Flask SQL Alchemy. His main points where:

* The functional approach to SQL Alchemy
* The object-oriented approach to SQL Alchemy
* Relationships in the database
* Authorizations and mutability in the Database

What I learnt:

This video went in depth into the functionality of SQL Alchemy, it was very useful and taught me many things about SQL Alchemy. I learned about the functional and object-oriented approaches to interacting with SQL Alchemy, which broadened my understanding of how to effectively manage and manipulate database records. The section on defining and managing relationships in the database was particularly enlightening, as it taught me how to structure complex data models and ensure data integrity. Additionally, the video covered important aspects of authorizations and mutability, highlighting how to control access to database records and manage changes efficiently. Overall, this tutorial provided me with practical skills and a deeper comprehension of using SQL Alchemy in my Flask project, ensuring that my application's data handling will be robust and well-organized.

Although I will most likely watch more videos to cover niche and small topics these two, I believe will be enough to cover the general topic for now, and it would be very time consuming to talk about all the other small videos I will watch.

### Documentation.

<https://flask.palletsprojects.com/en/3.0.x/>



Reading documentation is really important to get a good understanding of whatever framework or dependency you are going to use. The documentation obviously could teach me everything I ever need to know about Flask, but this is too time consuming so throughout the project I will refer back to this documentation if I get stuck. To start with I learned how to install and get Flask set up. It was also really useful as in the documentation there are lots of little examples of real projects and how flask is used in them, giving me some more context and understanding.

<https://docs.sqlalchemy.org/en/20/>



the documentation for SQL Alchemy is not very well laid out and is quite messy. It is still very useful, and I will definitely refer back to it if I can but for now I did not learn anything new.

<https://www.sqlite.org/docs.html>



SQLite is a lightweight, serverless, and self-contained SQL database engine that is perfect for small to medium-sized applications like mine. In the context of my project, I will use SQLite to manage and store user data, such as protein and calorie intake, goals, and progress records. By integrating SQLite with SQLAlchemy, I can easily define and manipulate database models using Python objects, ensuring seamless data handling and efficient CRUD operations. This combination allows me to implement complex data relationships and perform queries with minimal setup, providing a robust backend for my Flask application.

The documentation is not so easily readable similarly to SQLAlchemy but I do not need to use this database engine as much as SQLAlchemy.

# Specifications

## 3.1 Program Functions

1. User Authentication:
   * Description: This function allows users to securely register, log in, and log out of the application. It ensures that only authorized users can access personalized features, protecting user data and privacy.
   * Implementation:
     + Registration: Users can create an account by providing their email, first name, and password. The password is hashed using werkzeug.security to ensure security, making it difficult for attackers to retrieve the original password even if the database is compromised.
     + Login: Users log in with their email and password. The application checks the credentials against the stored user data in the database. If the credentials match, a session is created using flask\_login, allowing the user to remain logged in across different pages.
     + Logout: Users can securely log out, ending their session. This is achieved by calling the logout\_user function from flask\_login, which clears the session and ensures that subsequent requests are not authenticated.
2. Meal Logging:
   * Description: This function enables users to log their daily meals, including details such as meal description, protein content, and calories. This is useful for tracking dietary intake and ensuring users meet their nutritional goals.
   * Implementation:
     + Form Submission: Users can add meal entries through a form on the home page. Each meal is associated with the user's ID and the current date, ensuring accurate tracking of daily intake.
     + Data Storage: Meal entries are stored in the Meal table in the database. The Meal model includes fields for meal type, protein, calories, date, and a foreign key linking the meal to the user who logged it.
3. Goal Setting:
   * Description: Users can set and update their daily protein and calorie goals. This feature helps users establish and track their dietary targets.
   * Implementation:
     + Form Submission: Users can update their dietary goals through a form on the home page. The form includes fields for daily protein and calorie goals, validated to ensure they are non-negative integers.
     + Data Storage: Goals are stored in the User table. The user's daily protein and calorie goals can be updated anytime, providing flexibility to adjust targets as needed.
4. Progress Monitoring:
   * Description: Provides visual feedback on the user's progress towards their daily dietary goals. This helps users stay motivated and track their progress over time.
   * Implementation:
     + Data Calculation: The total protein and calories consumed by the user each day are compared against their goals. This is calculated by summing the protein and calorie values of the meals logged on the current day.
     + Visualization: Uses Dash to create charts and graphs displaying the user's progress. These visualizations include progress bars and charts showing the percentage of goals achieved.
5. Data Visualization:
   * Description: Displays interactive charts and graphs showing the user's nutrient intake and goal achievement. This provides a clear and engaging way to analyze dietary patterns.
   * Implementation:
     + Dash Integration: Uses Plotly and Dash to generate visualizations. Dash apps are embedded within the main Flask app, providing seamless integration.
     + Embedding: Visualizations are embedded in the dashboard and home pages using iframes. This allows users to view their progress without navigating away from the main application.
6. User Profile Management:
   * Description: Allows users to update their profile information and view progress data. This ensures that user information is up-to-date and personalized.
   * Implementation:
     + Profile Updates: Users can change their email and name through forms on the data page. Validation checks ensure the email is unique and the name meets minimum length requirements.
     + Progress Display: Shows a list of all users and their progress on the data page. This includes a summary of each user's daily protein and calorie intake and their progress towards their goals.
7. Dashboard:
   * Description: Centralized overview displaying today’s intake, goal progress, and recent activity. The dashboard provides users with a comprehensive view of their dietary data.
   * Implementation:
     + Layout: The dashboard includes sections for meal logging, goal setting, and progress visualization. This organized layout ensures users can easily access and interact with all features.
     + Embedding: Uses an iframe to embed Dash visualizations. This allows dynamic charts and graphs to be displayed directly within the dashboard.
8. Analytics:
   * Description: Provides detailed insights into the user's dietary trends over time. This helps users understand their eating habits and make informed dietary decisions.
   * Implementation:
     + Data Analysis: Collects and analyzes user data to generate trends and patterns. This includes tracking nutrient intake and goal achievement over time.
     + Visualization: Displays insights through interactive charts and graphs created with Dash. These visualizations help users identify patterns and adjust their dietary habits accordingly.
9. Help and Support:
   * Description: Provides resources and guides for users on how to use the application. This ensures users can effectively utilize all features of the application.
   * Implementation:
     + Documentation: Includes a help section with FAQs and user guides. These resources provide step-by-step instructions and troubleshooting tips.
     + Support Form: Option to contact support for additional help. Users can submit inquiries or report issues, ensuring they receive timely assistance.
10. Data Security:
    * Description: Ensures user data is protected with encryption and secure storage practices. This is crucial for maintaining user trust and complying with data protection regulations.
    * Implementation:
      + Password Hashing: Passwords are hashed before storage using werkzeug.security. This ensures that even if the database is compromised, the original passwords cannot be easily retrieved.
      + Secure Sessions: Manages user sessions securely using flask\_login. This includes handling session creation and termination, ensuring that unauthorized access is prevented.

## 3.2 User Interface (UI)

A good user interface is what makes or breaks this project, it is key to get it right so that the end users can be happy with the results and not frustrated or confused with my project. So the user interface (UI) for my protein tracking app will be designed to be user-friendly. I need to make sure that users can navigate through the application easily and efficiently log their meals, set goals and track their progress. Below is a detailed plan for all my UI components and their purposes and functions.

Overall Design Philosophy:

The UI will be clean and minimalistic, focusing on ease of use and clarity. By leveraging Bootstrap, the application will have a modern look and feel, with consistent styling across all pages. The use of responsive design principles will ensure that the app works seamlessly on both desktop and mobile devices. I want a simple application that does not overwhelm the users but gets the job done.

Main Components:

1. Navigation Bar:
   * Description: A fixed navigation bar at the top of the page will allow users to access different sections of the application quickly.
   * Elements:
     + Home: A link to the home page where users can log their meals and view daily intake.
     + Dashboard: A link to the dashboard where users can see their progress and analytics.
     + Data: A link to a page displaying user data and allowing profile updates.
     + Login/Logout: Links for user authentication, displaying either login or logout based on the user's authentication status.
   * Technology: The navigation bar will be styled using Bootstrap classes to ensure consistency and responsiveness.
2. Home Page:
   * Description: The central hub for users to log their meals and view their daily protein and calorie intake.
   * Elements:
     + Greeting Message: A personalized greeting that welcomes the user.
     + Meal Logging Form: A form where users can enter meal descriptions, protein content, and calories.
     + Today's Meals List: A list displaying the meals logged for the current day.
     + Total Intake Display: A summary showing the total protein and calories consumed today.
     + Goal Setting Form: A form for users to set or update their daily protein and calorie goals.
     + Clear Meals Button: A button to clear the meals logged for the current day.
   * Technology: The home page will use forms created with Flask-WTF and styled with Bootstrap for a consistent look.
3. Dashboard Page:
   * Description: A page dedicated to visualizing the user's progress through interactive charts and graphs.
   * Elements:
     + Progress Charts: Charts displaying protein and calorie intake compared to daily goals.
     + Nutrient Breakdown: Graphs showing the distribution of different nutrients consumed.
   * Technology: The dashboard will embed Dash applications within iframes to display dynamic visualizations created with Plotly.
4. Data Page:
   * Description: A page for displaying and managing user profile information and overall progress.
   * Elements:
     + User Table: A table listing all users with options to update their information.
     + Progress Bars: Visual indicators showing each user's progress towards their goals.
   * Technology: The data page will use Bootstrap for styling and Flask to dynamically generate the table and progress bars.
5. Authentication Pages:
   * Description: Pages for user login and registration.
   * Elements:
     + Login Form: A form for users to enter their email and password to log in.
     + Sign-Up Form: A form for new users to create an account, including fields for email, first name, and password.
   * Technology: Both forms will be created with Flask-WTF and styled with Bootstrap to ensure a consistent user experience.
6. Styling and Responsiveness:
   * Description: Consistent styling across the application to provide a seamless user experience.
   * Elements:
     + Custom CSS: A custom CSS file to override Bootstrap styles where necessary and ensure a cohesive design.
     + Responsive Design: Use of media queries to ensure the application looks good on all screen sizes, from mobile phones to desktop monitors.
   * Technology: The custom CSS will be loaded in the base template, and Bootstrap classes will be used extensively for layout and styling.

Planned Enhancements: As the project progresses, additional enhancements will be considered, such as:

* Help and Support Section: Providing users with guides and FAQs to assist them in using the application.
* User Notifications: Implementing notifications to remind users to log their meals or update their goals.
* Theme Customization: Allowing users to switch between different themes for a personalized experience.

This plan for my user interface may change through out the process but for now is a good representation and plan of what is to come. By taking the time to break down all the components of my UI I can get a good interface that hopefully reaches the goals I have set for myself.

## 3.3 Help Documentation

The Help Documentation section will provide users with comprehensive guidance on how to use the Protein Tracking App effectively. It will cover various aspects of the application, including user authentication, meal logging, goal setting, and viewing progress. The documentation will be clear, concise, and easy to navigate, ensuring that users can quickly find the information they need.

Structure of the Help Documentation:

1. Introduction:
   * Description: An overview of the Protein Tracking App, its purpose, and its key features.
   * Content:
     + Welcome message.
     + Brief description of the app's purpose.
     + List of main features (e.g., meal logging, goal setting, progress tracking).
2. Getting Started:
   * Description: Instructions on how to create an account, log in, and navigate the main sections of the app.
   * Content:
     + Creating an Account: Step-by-step guide on how to sign up, including screenshots of the sign-up form and explanations of each field.
     + Logging In: Instructions on how to log in, with screenshots of the login form.
     + Navigating the App: Overview of the main sections (Home, Dashboard, Data) and how to access them from the navigation bar.
3. Using the Home Page:
   * Description: Detailed guide on how to log meals, view daily intake, and set goals.
   * Content:
     + Logging Meals: Explanation of the meal logging form, including how to enter meal descriptions, protein content, and calories. Include screenshots and a sample entry.
     + Viewing Today's Meals: Instructions on how to view the list of meals logged for the current day and interpret the total intake summary.
     + Setting Goals: Guide on how to use the goal setting form to set or update daily protein and calorie goals. Include screenshots and tips for setting realistic goals.
     + Clearing Meals: Instructions on how to clear the meals logged for the day using the "Clear Today's Meals" button.
4. Using the Dashboard:
   * Description: Guide on how to view and interpret the visualizations on the dashboard.
   * Content:
     + Progress Charts: Explanation of the different charts and graphs available on the dashboard, including how to read them and what information they provide (e.g., goal achievement, nutrient breakdown).
     + Interacting with Visualizations: Tips on how to interact with the charts (e.g., hovering for details, filtering data).
     + Navigating Back: Instructions on how to return to the main sections of the app from the dashboard.
5. Using the Data Page:
   * Description: Instructions on how to view and manage user profile information and overall progress.
   * Content:
     + Viewing User Data: Guide on how to view the list of users and their progress data. Include explanations of the table columns and progress bars.
     + Updating Profile Information: Instructions on how to update email and name, with screenshots of the update form.
     + Understanding Progress Bars: Explanation of how to read the progress bars and what the percentages represent.
6. Account Management:
   * Description: Guide on how to manage account settings and log out.
   * Content:
     + Changing Account Details: Instructions on how to update account information (e.g., email, password).
     + Logging Out: Guide on how to log out securely from the application.
7. Frequently Asked Questions (FAQs):
   * Description: A section addressing common questions and issues users might encounter.
   * Content:
     + General Questions: Answers to common questions about the app's purpose and functionality.
     + Technical Issues: Solutions to common technical problems (e.g., trouble logging in, form validation errors).
     + Usage Tips: Tips for making the most of the app (e.g., setting achievable goals, regularly logging meals).
8. Contact Support:
   * Description: Information on how users can get additional help if needed.
   * Content:
     + Support Form: Instructions on how to fill out the support form to contact the support team.
     + Contact Information: Email and other contact details for reaching out to support.

Implementation Plan:

1. Create Help Documentation Template:
   * Design a help documentation template using HTML and Bootstrap for consistent styling.
   * Ensure the template is responsive and easy to navigate.
2. Populate Content:
   * Write detailed content for each section outlined above.
   * Include screenshots and visual aids to enhance understanding.
3. Integrate with Application:
   * Add a "Help" link to the navigation bar that directs users to the help documentation.
   * Ensure the help documentation is accessible from all main sections of the app.
4. Review and Update:
   * Regularly review and update the help documentation based on user feedback and changes to the application.
   * Add new FAQs and support information as needed.

By providing comprehensive help documentation, the Protein Tracking App will ensure that users can effectively utilize all features, achieve their dietary goals, and have a positive experience using the application.

Although this plan is concise in reality due to time constraints and the simplicity of my project, I think I can get away with my help page being a bit shorter and less descriptive. But this is a good starting point.

## 3.4 Permanent Data Storage

Permanent data storage is a critical component of the Protein Tracking App, ensuring that all user data, including account details, meal logs, and dietary goals, are securely stored and can be retrieved and updated as needed. This section outlines the approach to managing and storing data permanently using a relational database.

Database Choice:

The application will use SQLite for permanent data storage. SQLite is a lightweight, file-based database that is easy to set up and does not require a separate server, making it ideal for small to medium-sized applications like the Protein Tracking App. SQLite provides robust features for data integrity, transactions, and concurrency, ensuring reliable data management.

Database Schema: The database schema will include tables for users, meals, and any additional data needed to support the application's functionality. Each table will have a set of fields appropriate for storing the necessary data, with relationships established between tables to ensure data consistency and integrity.

Tables and Fields:

1. User Table:
   * Description: Stores user account information and dietary goals.
   * Fields:
     + id: Integer, Primary Key, Auto-incremented, unique identifier for each user.
     + email: String, Unique, stores the user's email address.
     + password: String, stores the hashed password for secure authentication.
     + first\_name: String, stores the user's first name.
     + daily\_protein\_goal: Integer, stores the user's daily protein intake goal (default: 100 grams).
     + daily\_calorie\_goal: Integer, stores the user's daily calorie intake goal (default: 1000 calories).
2. Meal Table:
   * Description: Stores information about meals logged by users.
   * Fields:
     + id: Integer, Primary Key, Auto-incremented, unique identifier for each meal.
     + date: Date, stores the date the meal was logged.
     + meal\_type: String, stores the description or type of the meal.
     + protein: Integer, stores the protein content of the meal (in grams).
     + calories: Integer, stores the calorie content of the meal.
     + user\_id: Integer, Foreign Key, references id in the User table to associate meals with users.

Database Operations:

The application will perform various operations on the database to manage data, including creating, reading, updating, and deleting records. These operations will be handled through SQLAlchemy, an Object Relational Mapper (ORM) that provides a high-level interface for interacting with the database using Python objects.

Implementation Details:

1. Creating the Database:

The database will be initialized in the create\_app function in the \_\_init\_\_.py file. If the database file does not exist, it will be created, and the necessary tables will be defined and set up.

* + python

def create\_database(app):

if not path.exists('website/' + DB\_NAME):

with app.app\_context():

db.create\_all()

print('Created Database!')

1. Models Definition:

The User and Meal models will be defined in the models.py file using SQLAlchemy's declarative base.

* + Example:

python

from . import db

from flask\_login import UserMixin

class User(db.Model, UserMixin):

id = db.Column(db.Integer, primary\_key=True)

email = db.Column(db.String(150), unique=True)

password = db.Column(db.String(150))

first\_name = db.Column(db.String(150))

daily\_protein\_goal = db.Column(db.Integer, nullable=True, default=100)

daily\_calorie\_goal = db.Column(db.Integer, nullable=True, default=1000)

meals = db.relationship('Meal', backref='user', lazy='dynamic')

class Meal(db.Model):

id = db.Column(db.Integer, primary\_key=True)

date = db.Column(db.Date, default=datetime.utcnow().date)

meal\_type = db.Column(db.String, nullable=False)

protein = db.Column(db.Integer, nullable=False)

calories = db.Column(db.Integer, nullable=False)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'))

1. CRUD Operations:

Create: New records (e.g., user accounts, meal logs) will be added to the database using SQLAlchemy's session management.

* + - Example:

Python

new\_meal = Meal(date=datetime.utcnow().date(), meal\_type=meal\_form.meal\_type.data, protein=meal\_form.protein.data, calories=meal\_form.calories.data, user\_id=current\_user.id)

db.session.add(new\_meal)

db.session.commit()

Read: Data will be queried from the database to display user information, meals, and progress.

* + - Example:

python

meals = Meal.query.filter\_by(user\_id=current\_user.id, date=datetime.utcnow().date()).all()

Update: Existing records can be modified, such as updating user goals or editing logged meals.

* + - Example:

python

user\_to\_update.first\_name = updated\_name

db.session.commit()

Delete: Records can be removed from the database, such as deleting a meal or clearing all meals for the day.

* + - Example:

python

Meal.query.filter\_by(user\_id=current\_user.id, date=today).delete()

db.session.commit()

Security Measures: To ensure the security and integrity of the data:

* Passwords will be hashed using werkzeug.security before storing them in the database.
* SQLAlchemy will be used to prevent SQL injection attacks by safely parameterizing queries.
* Regular backups of the database file will be recommended to prevent data loss.

Data loss prevention:

I am planning for the Protein tracking app to also utilize a word document to store and display user information and meal data from the database. This Word file serves as an additional backup and a convenient format for viewing and sharing data outside of the application. By exporting the data to a Word document, the administrator (Me) can easily see all the current data on the database and site, and use this data however I like, it is just extremely useful.

The Word file will be created and managed using the python-docx library, which allows for dynamic generation and modification of Word documents. The document includes sections for user login data and detailed records of meals, organized in tables for clarity. Each user's meals are grouped under their name, with columns for protein, calories, and the date of consumption, providing a structured and readable format.

The process involves fetching data from the SQLite database and populating the Word document accordingly. Functions are defined to clear the document, add titles and headings, and insert rows of data for users and their meals. This ensures that the document is always up-to-date with the latest information from the database. Additionally, the Word file serves as a non-technical backup, making the data accessible even to those who are not familiar with database systems.

By including this feature, the Protein Tracking App enhances data accessibility and ensures that important dietary information is preserved in a user-friendly format, complementing the primary database storage.

The administrator (Me) can also regularly save the word documents to my personal computer so that in the case of anything happening to the SQL Alchemy database, there is a back up of the information.

## 3.5 Hardware and Software Requirements

Although I am not creating a massively computationally intensive computer program, it is still useful to talk about what is needed for this project. To make sure that there is a smooth development process and software dependencies for developing , deploying and using the application are all met.

**Hardware Requirements:**

1. **For Development:**
   * **Processor:** Dual-core processor (Intel i5 or equivalent) or better.
   * **RAM:** Minimum 8 GB of RAM.
   * **Storage:** At least10 GB of free storage space for project files, dependencies, and database.
   * **Operating System:** Windows 10, macOS Catalina, or a recent Linux distribution.
   * **Network:** Reliable internet connection for downloading dependencies and accessing online resources.
2. **For Deployment (Server):**
   * **Processor:** Dual-core processor (Intel i5 or equivalent) or better.
   * **RAM:** Minimum 4 GB of RAM (8 GB recommended for better performance).
   * **Storage:** At least 5 GB of free storage space for the application, database, and logs.
   * **Network:** Stable internet connection with adequate bandwidth to handle user requests and data transmission.
3. **For End Users:**
   * **Device:** Any modern desktop, laptop, tablet, or smartphone.
   * **Processor:** Any recent processor.
   * **RAM:** Minimum 2 GB of RAM.
   * **Storage:** Sufficient storage to install and run a web browser.
   * **Operating System:** Any operating system capable of running a modern web browser (Windows, macOS, Linux, iOS, Android).
   * **Network:** Reliable internet connection for accessing the web application.

**Software Requirements:**

1. **For Development:**
   * **Programming Language:** Python 3.7 or higher.
   * **Web Framework:** Flask 1.1.2 or higher.
   * **Database Management:** SQLAlchemy 1.3 or higher, SQLite for local development.
   * **Front-end Libraries:** Bootstrap 4.4.1 for responsive design, jQuery 3.2.1 for interactivity.
   * **Visualization Libraries:** Plotly and Dash for creating interactive charts and graphs.
   * **Other Dependencies:**
     + Flask-WTF for form handling and validation.
     + Werkzeug for security and password hashing.
     + Flask-Login for managing user sessions.
     + python-docx for generating and managing Word documents.
     + Flask-Migrate for database migrations. (Not needed that bad)
   * **Development Environment:** Any code editor or IDE supporting Python (e.g., Visual Studio Code, PyCharm).
   * **Version Control:** Git for version control and collaboration.
2. **For Deployment (Server):**
   * **Web Server:** Gunicorn for running the Flask application.
   * **Database Management:** SQLite for small deployments or PostgreSQL for more robust solutions.
   * **Operating System Dependencies:** Required system packages for running Python applications and managing databases.
   * **Containerization (optional):** Docker for containerizing the application to ensure consistency across different environments.
3. **For End Users:**
   * **Web Browser:** Any modern web browser (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) that supports HTML5, CSS3, and JavaScript.
   * **Browser Extensions:** No specific extensions required, but ensuring the browser is up-to-date will provide the best experience.

**Additional Tools and Services:**

* **Hosting:** A cloud service provider (e.g., AWS, Heroku, DigitalOcean) for deploying the web application.
* **Monitoring and Logging:** Tools like New Relic or Loggly for monitoring application performance and logging errors.

By meeting these hardware and software requirements, the Protein Tracking App will run efficiently and provide a seamless experience for both developers and end users. Ensuring that all dependencies and system requirements are satisfied will help in achieving the desired functionality and performance of the application.

Note:

Some of these dependencies, system requirements or software may change throughout the development process. For example, I am hoping to be able to secure HTTPS for my web application so that I can host my project to others online, actually doing this may be complicated or time intensive.

# Design Documentation

## 4.1 Interface Design

Although we have already talked about the user interface design previously, it is now time to make some design decisions and set some things in place to get this project started.

Design Principles:

* Simplicity: The interface will be clean and uncluttered, focusing on essential functions and minimizing distractions.
* Consistency: Consistent use of colors, fonts, and layouts to provide a cohesive user experience.
* Responsiveness: Ensuring the app works seamlessly on desktops, tablets, and mobile devices.
* Accessibility: Designing for users with different abilities, ensuring that the interface is navigable and usable by everyone.

Main Components and Layout:

1. Navigation Bar:
   * Description: A fixed navigation bar at the top of the page provides easy access to different sections of the application.
   * Elements:
     + Brand Logo: Positioned on the left, links to the home page.(if I get a logo?)
     + Menu Items: Home, Dashboard, Data, Login/Logout, and Sign-Up links.
     + User Menu: Displays user's name when logged in, with a dropdown for profile and logout options.
   * Design:
     + Color Scheme: Dark background with white text for better contrast.
     + Responsive Behavior: Collapses into a hamburger menu on smaller screens.
2. Home Page:
   * Description: Central hub where users can log their meals, view daily intake, and set goals.
   * Layout:
     + Greeting Message: Personalized message welcoming the user, displayed prominently at the top.
     + Meal Logging Form: A form with fields for meal description, protein, and calories, and a submit button.
     + Today’s Meals List: A list of meals logged for the current day, displayed below the logging form.
     + Total Intake Display: A summary section showing total protein and calories consumed today.
     + Goal Setting Form: A form for setting daily protein and calorie goals.
     + Clear Meals Button: A button to clear today’s logged meals.
   * Design:
     + Color Scheme: Light background with dark text for readability.
     + Form Elements: Styled with Bootstrap classes for consistency and responsiveness.
     + Interactive Elements: Buttons and input fields with hover and focus states for better user interaction.
3. Dashboard Page:
   * Description: Displays visualizations of the user's progress and dietary trends.
   * Layout:
     + Header: Title and brief description of the dashboard.
     + Progress Charts: Interactive charts showing goal achievement and nutrient breakdown.
     + Nutrient Breakdown Graphs: Graphs displaying detailed nutrient intake over time.
   * Design:
     + Color Scheme: Consistent with the overall app theme, using contrasting colors for charts.
     + Embedding: Charts embedded using iframes, ensuring they are responsive and interactive.
4. Data Page:
   * Description: Displays and manages user profile information and progress data.
   * Layout:
     + User Table: A table listing all users with columns for first name, email, and goals.
     + Update Form: Form for updating user information directly within the table rows.
     + Progress Bars: Visual indicators showing each user’s progress towards their goals.
   * Design:
     + Color Scheme: Light background with dark text for readability.
     + Table Styling: Bootstrap-styled tables for a clean and organized look.
     + Progress Bars: Color-coded bars indicating percentage completion.
5. Authentication Pages:
   * Description: Pages for user login and registration.
   * Layout:
     + Login Form: Fields for email and password, with a submit button.
     + Sign-Up Form: Fields for email, first name, and password, with a submit button.
   * Design:
     + Color Scheme: Light background with contrasting input fields.
     + Form Elements: Bootstrap-styled forms for consistency and ease of use.

Colour choices:

Personally, I like the red and black colour scheme, this suits what I have envisioned.

From colours.com

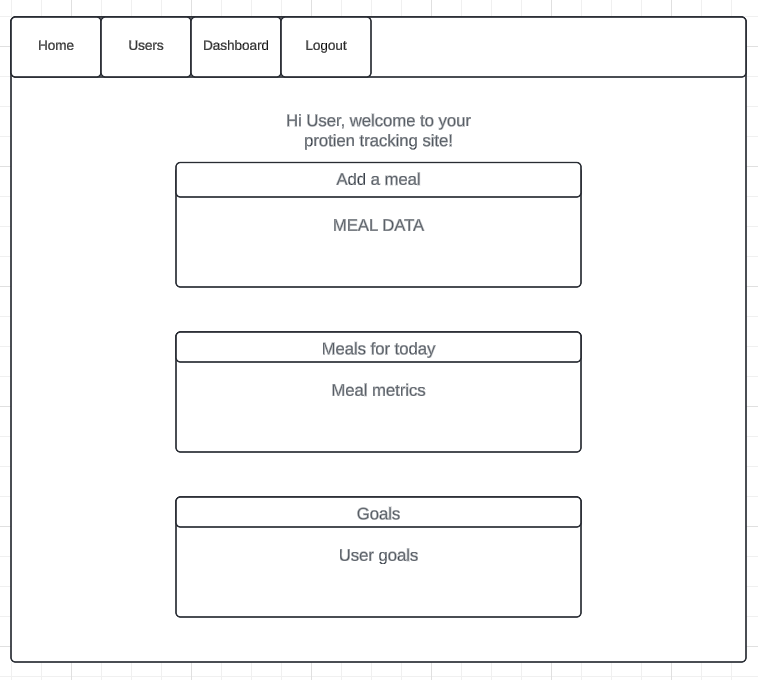


This template has a wide range of colours but I most likely will keep it simple and only use the dark black and red colours, however there may be changes in the development process. Because I have no stakeholder, I have artistic freedom and customization in my project which is very nice.

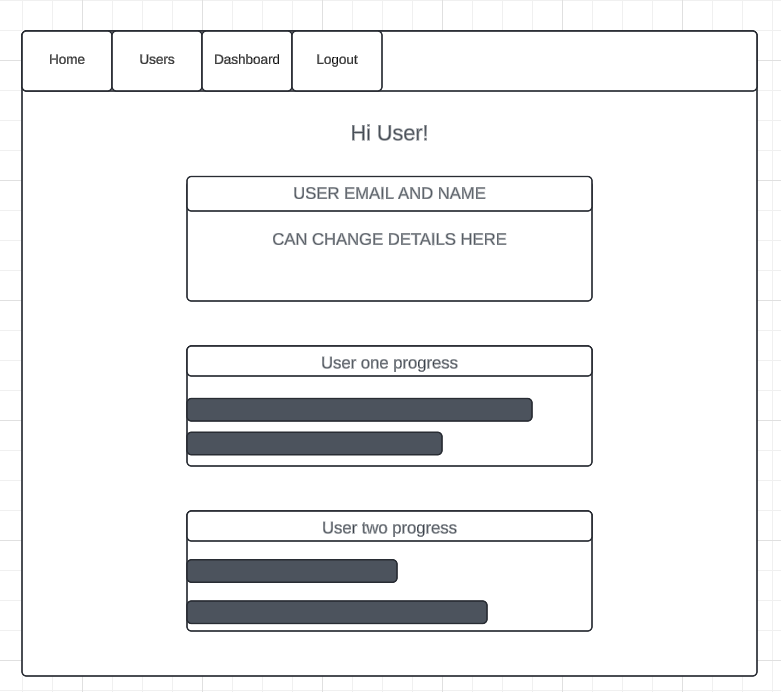
Wireframes:

To provide a visual representation of the interface design, wireframes will be created for each of the main components. These wireframes will serve as a blueprint for the development phase, ensuring that the design principles and layout are adhered to.

Home page:

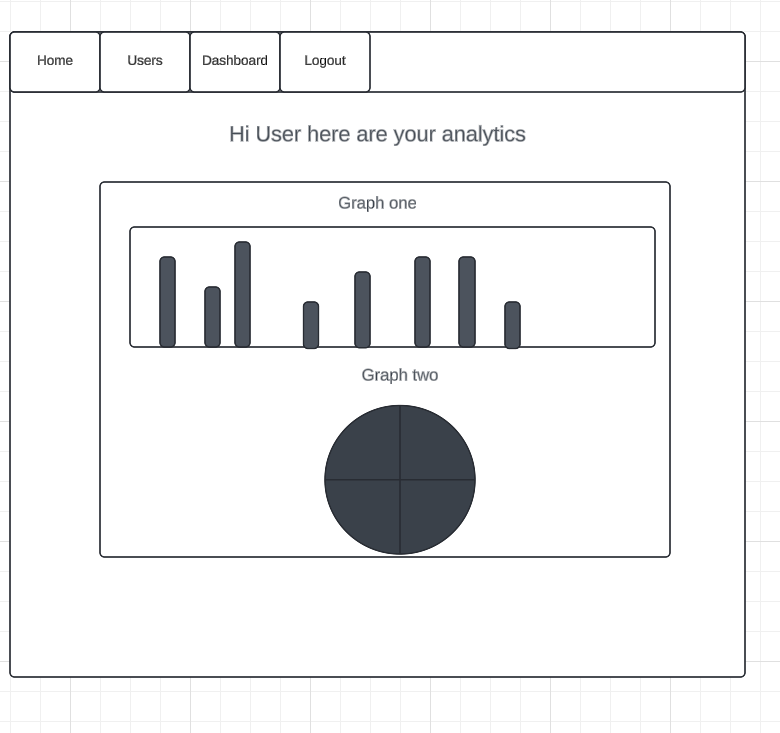
 This home page reflects what I want to be done for the user in my specifications and visually is simple enough to create and look at.

Users page:



The users page allows the user to change their personal information and also at the same time look at the other users progress hopefully motivating them to want to keep up or do better with their goals.

Analytics / dashboard page:



On this page the users can look at what their data that has been collected over their time on the site and see their overall progress and information that is interesting. Like what meals they eat the most, or what days they don’t get enough protein / calories. Things like that.

Interaction Design:

1. Hover States:
   * Menu items and buttons will change color or display a shadow effect when hovered over, providing visual feedback to the user.
2. Form Validation:
   * Real-time form validation will provide immediate feedback on user input, highlighting errors and guiding the user to correct them.
3. Responsive Navigation:
   * The navigation bar will collapse into a hamburger menu on smaller screens, ensuring that it remains accessible without taking up too much space.
4. Interactive Charts:
   * Charts and graphs on the dashboard will allow users to hover over data points to view detailed information, providing an engaging and informative experience.

## 4.2 Program Flow

The program flow of the Protein Tracking App needs to be designed to provide a simple and intuitive user experience while ensuring efficient data processing and management. This section outlines the flow of the application, from user interactions to backend processing, covering the major functionalities of the app. This is important to get right and decides how efficient and easy it is for me and the user.

1. User Authentication Flow:

* Description: Manages the process of user registration, login, and logout, ensuring secure access to the application.
* Flow:
  1. User Registration:
     + User accesses the sign-up page and fills in the registration form with their email, first name, and password.
     + Form validation checks for input correctness and uniqueness of the email.
     + On successful validation, the password is hashed, and the new user record is created in the database.
     + User is automatically logged in and redirected to the home page.
  2. User Login:
     + User accesses the login page and enters their email and password.
     + The system checks the provided credentials against the stored data.
     + On successful authentication, a session is created, and the user is redirected to the home page.
  3. User Logout:
     + User clicks the logout link.
     + The session is terminated, and the user is redirected to the login page.

2. Meal Logging Flow:

* Description: Allows users to log their meals, including details such as meal description, protein content, and calories.
* Flow:
  1. Meal Entry:
     + User accesses the home page and fills in the meal logging form.
     + Form validation ensures that all required fields (meal description, protein, calories) are filled in correctly.
     + On form submission, the meal data is saved to the Meal table in the database with the current date and user ID.
     + The home page is refreshed to display the newly logged meal in the list of today's meals.
  2. View Today's Meals:
     + The home page retrieves and displays all meals logged by the user for the current date.
     + The total protein and calorie intake for the day are calculated and displayed.

3. Goal Setting Flow:

* Description: Enables users to set and update their daily protein and calorie goals.
* Flow:
  1. Set/Update Goals:
     + User accesses the home page and fills in the goal setting form with their daily protein and calorie goals.
     + Form validation ensures that the goals are non-negative integers.
     + On form submission, the user's goals are updated in the User table in the database.
     + The home page is refreshed to display the updated goals.

4. Progress Monitoring Flow:

* Description: Provides users with visual feedback on their progress towards their daily dietary goals.
* Flow:
  1. Data Retrieval:
     + The dashboard retrieves the user's logged meals and goals from the database.
     + The total protein and calories consumed are compared against the user's goals.
  2. Visualization:
     + Dash callbacks generate interactive charts and graphs displaying the user's progress.
     + The charts include progress bars and graphs showing goal achievement and nutrient breakdown.

5. Data Visualization Flow:

* Description: Displays interactive charts and graphs showing the user's nutrient intake and goal achievement.
* Flow:
  1. Data Preparation:
     + The system retrieves and processes the user's meal data to prepare it for visualization.
  2. Chart Generation:
     + Dash and Plotly are used to create interactive charts and graphs.
     + The charts are embedded in the dashboard and home pages using iframes.

6. User Profile Management Flow:

* Description: Allows users to update their profile information and view progress data.
* Flow:
  1. View Profile:
     + User accesses the data page to view their profile information and progress.
  2. Update Profile:
     + User edits their email or first name directly in the profile update form.
     + On form submission, the updated information is saved to the User table in the database.
     + The data page is refreshed to display the updated profile information.

7. Data Export Flow:

* Description: Allows users to export their dietary data to a Word document.
* Flow:
  1. Data Retrieval:
     + The system retrieves the user's meal data from the database.
  2. Document Generation:
     + The docx\_handeling.py script generates a Word document using the python-docx library.
     + The document includes sections for user information and logged meals, organized in tables.
  3. Download Document:
     + User clicks an export button to download the generated Word document.

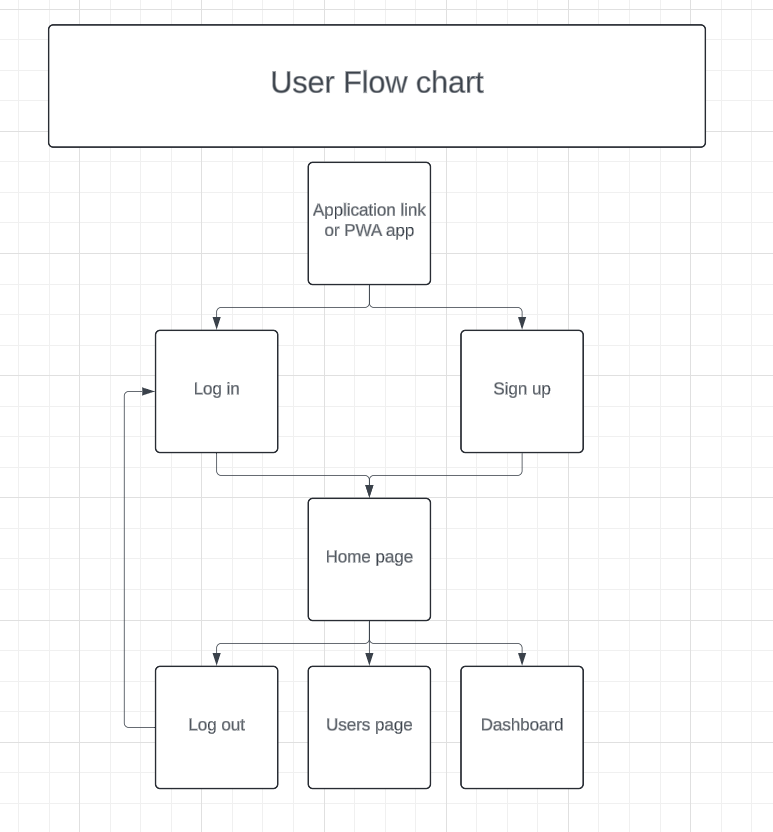
8. Error Handling Flow:

* Description: Ensures that the application gracefully handles errors and provides feedback to users.
* Flow:
  1. Form Validation Errors:
     + Input validation errors are displayed to the user with appropriate messages.
  2. Database Errors:
     + Any database-related errors (e.g., connection issues, query failures) are logged, and the user is notified with a friendly error message.
  3. General Errors:
     + Unhandled exceptions are caught, logged, and the user is shown a generic error message.

### 4.2.1 Flowcharts

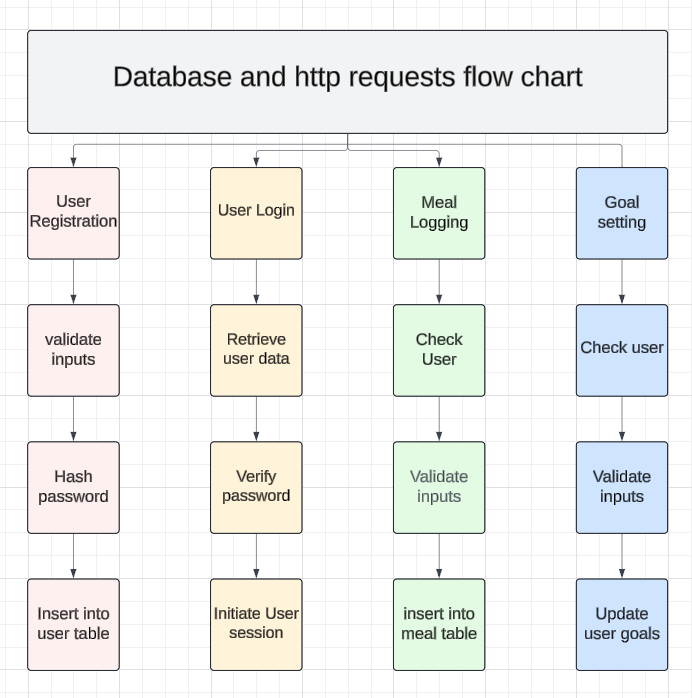
Made with: [www.lucidchart.com](http://www.lucidchart.com)

User flow chart:



This User flowchart shows visually the layout of the website I am aiming to achieve. It starts at the application link or app, as a PWA it can be made into a icon on the users phone that they can click easily. From there, they can either log in or sign up, but if they were previously logged in the site will remember their user session and just forward them to the home page. From the home page users can either go to the users page to see other users metrics and change their own information if needed or they can go to the Dashboard to see a better view of their analytics and graphs. The user can also sign out, this is there as more of a user experience heuristic that is mandatory but useful to have .

Database interaction flow chart:



This flow chart shows us visually the process involved in the database and http requests that we want to set up, handling the user login, user registration, meal logging and goal setting. It is useful to get a good grasp of what is going on beforehand to get a structured approach to the project.

### 4.2.2 Pseudocode

Pseudo code is a useful tool to get a good understanding of what we actually want to accomplish and give us a head start into thinking about how we are going to do it. The following are some examples of important aspects of my project that I am planning for. Obviously, I am not writing the whole project here in pseudo code as that would be a waste of time.

**User registration:**

If user enters email, first name, and password:

Check if email already exists

If email exists:

Show "Email already exists" message

Else:

Hash the password

Save user info (email, first name, hashed password) in database

Log the user in

Show "Registration successful" message

**User login:**

If user enters email and password:

Find user by email in database

If user not found:

Show "Email does not exist" message

Else:

Check if password is correct

If password is incorrect:

Show "Incorrect password" message

Else:

Log the user in

Show "Login successful" message

**Meal logging:**

If user enters meal description, protein, and calories:

Save meal info (user ID, meal description, protein, calories, current date) in database

Show "Meal logged successfully" message

**Goal setting:**

If user enters daily protein goal and daily calorie goal:

Save goals in database for the user

Show "Goals updated successfully" message

## 4.3 Class Design (UML Diagrams)

Unified modelling language (UML) diagrams are a useful tool to help visualize and break down the structure of a system. For my project, a UML diagram can be used to represent the static structure or database layout and functionality, showing the systems classes, their attributes, methods and relationships between objects.

**USER CLASS**

The User class represents the users of my project. Each user has a unique ID, email, first name, hashed password, and daily goals for protein and calories. Users can log their meals and update their profile information.

**Attributes:**

* id: Integer (Primary Key)
* email: String
* first\_name: String
* password: String (Hashed)
* daily\_protein\_goal: Integer
* daily\_calorie\_goal: Integer

**Methods:**

* \_\_init\_\_(): Initializes a new user.
* set\_goals(protein\_goal, calorie\_goal): Sets daily goals for protein and calories.
* log\_meal(meal): Logs a meal for the user.
* update\_profile(new\_email, new\_first\_name): Updates the user’s profile information.

**MEAL CLASS**

The Meal class represents a meal logged by a user. Each meal has a unique ID, the user who logged it, the date it was logged, a description of the meal, and the nutritional content in terms of protein and calories.

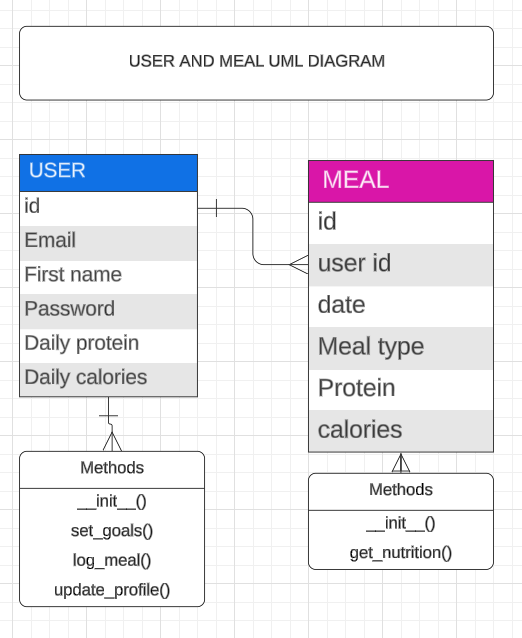
**Attributes:**

* id: Integer (Primary Key)
* user\_id: Integer (Foreign Key to User)
* date: Date
* meal\_type: String
* protein: Integer
* calories: Integer

**Methods:**

* \_\_init\_\_(): Initializes a new meal.
* get\_nutritional\_info(): Returns the nutritional information of the meal.

**DIAGRAM**

****

# Development Process

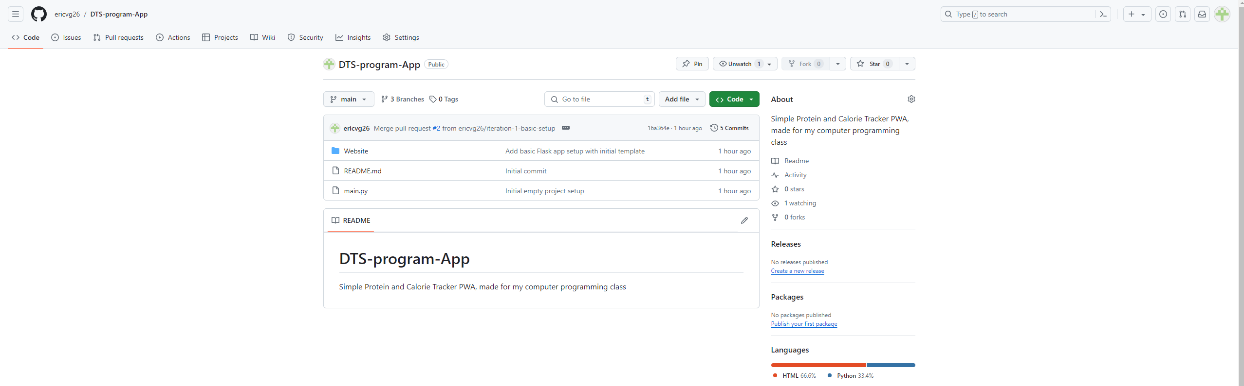
## 5.1 Iterations and Changes

**Iteration one:**

This is one of the most important steps of our development process, getting everything set up correctly and orderly. Because I reviewed projects similar to mine and have seen their code, I know what the correct file orientation is and how to maintain a simple easy flowing program. The first iteration involves setting up the foundational structure of the Protein Tracking App. This includes initializing the project, setting up the basic file structure, and implementing the initial user authentication system.

The end goal of this first iteration is to get a working Flask site with a home page, sign in page, sign up page, with everything working as intended.

Set up my GitHub repository:



File structure set up:

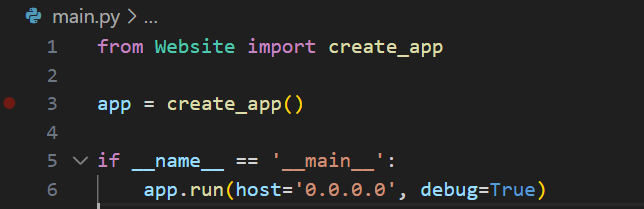
A screenshot of a computer program

Description automatically generated

As you can see, from the structure we mentioned earlier in the planning section, I have created my file layout that I can now iterate and code in, to create my project.

Code implementation:

Main.py:



Explanation:

**Purpose:** This is the entry point of the Flask application.

* **Functionality:**
  + create\_app() is called to initialize the Flask application.
  + The application runs on host='0.0.0.0' allowing access from any IP address, with debug=True enabling debug mode for development purposes.

Init.py:

A screen shot of a computer program

Description automatically generated

**Explanation:**

* **Purpose:** Initializes the Flask application and sets up the database and user session management.
* **Functionality:**
  + create\_app(): Configures the Flask app, sets secret keys and database URIs, initializes the database, and registers blueprints for views and authentication.
  + create\_database(app): Checks if the database file exists, and if not, creates it.
  + login\_manager: Manages user login sessions and sets the default login view.

Auth.py:

A screen shot of a computer program

Description automatically generated

**Explanation:**

* **Purpose:** Manages user authentication including login, logout, and sign-up functionalities.
* **Functionality:**
  + login(): Validates user credentials and logs in the user if correct.
  + logout(): Logs out the current user.
  + sign\_up(): Registers a new user after validating the input data.

Models.py:

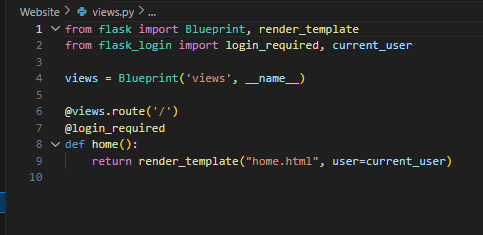
A screen shot of a computer code

Description automatically generated

**Explanation:**

* **Purpose:** Defines the database models for the application.
* **Functionality:**
  + User model: Stores user information, including login credentials and dietary goals.

Views.py:



**Explanation:**

* **Purpose:** Handles the routes and logic for the main functionalities of the application.
* **Functionality:**
  + home(): Manages the home page where users can log meals and set goals.

Base.html:

A screen shot of a computer program

Description automatically generated`

**Explanation:**

* **Purpose:** Serves as the base template for the web application, providing a consistent layout and styling for all pages.
* **Functionality:**
  + **Head Section:**
    - Includes meta tags for character set and viewport settings to ensure responsiveness on mobile devices.
    - Links to external CSS files from Bootstrap and Font Awesome for styling and icons.
    - Provides a block for child templates to set the page title dynamically.
    - Links to a custom CSS file (style.css) and a manifest file (manifest.json) for Progressive Web App (PWA) configuration.
  + **Body Section:**
    - Contains a navigation bar with links to different pages (Home, Users, Dashboard, Login, Sign Up, Logout), which are dynamically displayed based on user authentication status.
    - Includes a section for displaying flash messages (alerts) to inform users of success or error messages.
    - Defines a block for the main content ({% block content %}), which will be overridden by child templates.
    - Loads JavaScript files for jQuery, Popper.js, and Bootstrap to provide interactive components and behaviors.

All the html files are pretty self-explanatory, they just display information online.

Home.html:

A computer screen with text

Description automatically generated

Login.html:

A screen shot of a computer program

Description automatically generated

Sign\_up.html:

A screen shot of a computer program

Description automatically generated

Visual output:

A screenshot of a login form

Description automatically generated

­­

A screenshot of a login screen

Description automatically generated

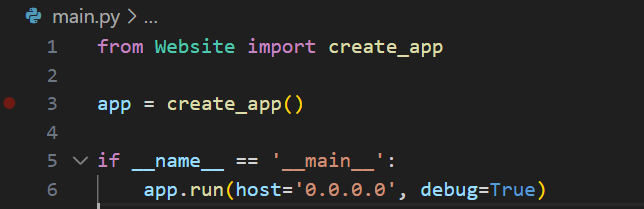
I have got the bare bones structure of my project, but it does not do it’s intended purpose at all yet. A lot of work is needed to be done and in the next iteration you will see a lot of progress, I am planning to only display three iterations of my project as this best suits what is going on.

**Iteration two (THE MOST IMPORTANT):**

This is the most important iteration, here is where the most work gets done and I will really get into the nitty gritty explaining all the code functionality and how everything works, in this iteration you can expect to see an almost finished project that should do it’s intended purpose.

Main.py:

This has no changes as nothing new was needed to be added, this remains the same.



\_\_init\_\_.py:



**Explanation:**

1. **Imports:**
   * Flask: The core Flask class used to create a Flask application.
   * SQLAlchemy: An ORM (Object-Relational Mapping) tool.
   * path: From the os module, used to check if the database file exists.
   * LoginManager: Manages user sessions and handles user authentication in Flask applications.
   * Migrate: Handles database migrations using Alembic in a Flask app.

(Please note that migrate has no functionality tools in the program but is rather a debugging tool used for the data base)

1. **Global Variables:**
   * db: An instance of SQLAlchemy.
   * DB\_NAME: The name of the database file.
2. **create\_app Function:**
   * **Initialization:**
     + Creates an instance of the Flask application.
     + Configures the app with a secret key (SECRET\_KEY) and the database URI (SQLALCHEMY\_DATABASE\_URI), pointing to the SQLite database.
   * **Database Initialization:**
     + Initializes the database with the Flask app context.
   * **Blueprint Registration:**
     + Registers blueprints for the main views (views) and authentication (auth), which organize the routes and views of the application.
   * **Database Models:**
     + Imports the database models (User and Meal) to ensure they are registered with the app.
   * **Database Creation:**
     + Calls create\_database to create the database if it does not already exist.
   * **Login Manager:**
     + Initializes the LoginManager, setting the login view to auth.login and configuring the app to load users by their ID.
   * **Database Migrations:**
     + Initializes Migrate to handle database migrations.

(again this is only for debugging and not for actual functionality)

* + **Dash App Integration:**
    - Imports and initializes the Dash app within the Flask app context.

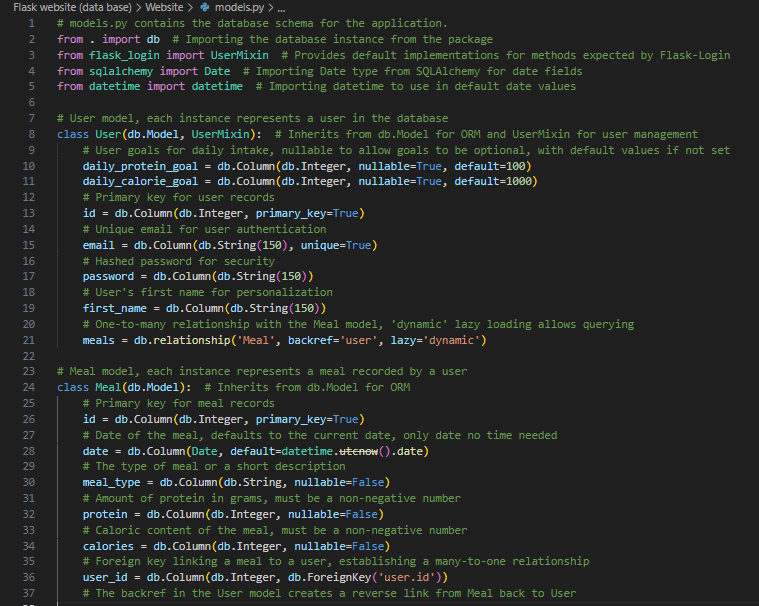
(this is what I said earlier will help with showing data )

* + **Return:**
    - Returns the configured Flask application instance.

1. **create\_database Function:**
   * **Check and Create:**
     + Checks if the database file exists in the specified path. If not, it creates the database within the Flask app context and prints a confirmation message.

This \_\_init\_\_.py file is central to setting up the Flask application, configuring the database, managing user sessions, and integrating the Dash app for data visualization. It ensures that the necessary components are initialized and connected properly, allowing the application to function as intended.

Models.py:



**Explanation:**

1. **Imports:**
   * db: The SQLAlchemy database instance imported from the package.
   * UserMixin: Provides default implementations for Flask-Login methods to manage user sessions.
   * Date: A SQLAlchemy type used to store date fields in the database.
   * datetime: A standard Python module used to get the current date.
2. **User Model:**
   * **daily\_protein\_goal:** An integer field to store the user's daily protein goal. It can be null and has a default value of 100 grams.
   * **daily\_calorie\_goal:** An integer field to store the user's daily calorie goal. It can be null and has a default value of 1000 calories.
   * **id:** The primary key for the User table, uniquely identifying each user.
   * **email:** A string field to store the user's email. It is unique and used for authentication.
   * **password:** A string field to store the user's hashed password for security.
   * **first\_name:** A string field to store the user's first name.
   * **meals:** A relationship field that sets up a one-to-many relationship with the Meal model. The backref attribute adds a reverse reference from the Meal back to the User, and lazy='dynamic' allows for dynamic querying.
3. **Meal Model:**
   * **id:** The primary key for the Meal table, uniquely identifying each meal entry.
   * **date:** A date field to store the date of the meal. It defaults to the current date using datetime.utcnow().date().
   * **meal\_type:** A string field to store the type or description of the meal. It cannot be null.
   * **protein:** An integer field to store the amount of protein in the meal. It cannot be null and must be non-negative.
   * **calories:** An integer field to store the caloric content of the meal. It cannot be null and must be non-negative.
   * **user\_id:** An integer field that sets up a foreign key relationship linking the meal to a user, establishing a many-to-one relationship. The user\_id references the id field in the User table.
   * **backref:** The back reference in the User model creates a reverse link from Meal back to User, allowing easy access to the user's meals.

This models.py file defines the layout for my projects database, including the User and Meal models, which are essential for managing user data and meal entries. The relationships between the models allows efficient querying and data manipulation within the application.

Views.py:



**Code Explanation:**

1. **Imports:**
   * Blueprint, request, redirect, url\_for, flash, render\_template: Flask components used to manage routing and rendering templates.
   * login\_required, current\_user: Flask-Login functions to handle user sessions and access control.
   * User, Meal: The models representing users and meals in the database.
   * db: The SQLAlchemy database instance.
   * MealForm, GoalForm: The forms for logging meals and setting goals.
   * datetime: Standard Python module to handle date and time.
2. **Blueprint Setup:**
   * views = Blueprint('views', \_\_name\_\_): Creates a Blueprint for the main site routes.
3. **Home Route:**
   * **Route:** @views.route('/', methods=['GET', 'POST'])

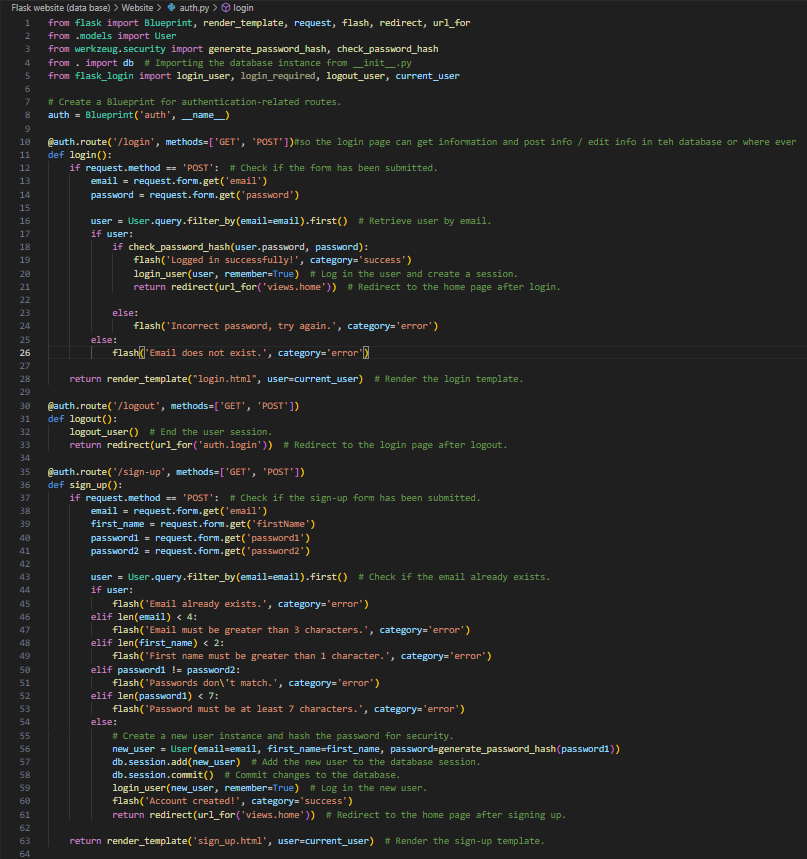
(this means that it can get requests and post requests)^

* + **Function:** home()
  + **Purpose:** Displays the home page where users can log meals and set goals.
  + **Key Operations:**
    - Fetches all users for admin purposes.
    - Initializes meal and goal forms.
    - Processes form submissions for adding meals and setting goals.
    - Calculates the total protein and calories consumed by the user today.
    - Pre-fills the goal form with the user's current goals.
    - Renders the home.html template with the necessary data.

1. **Data Route:**
   * **Route:** @views.route('/data', methods=['GET', "POST"])
   * **Function:** data()
   * **Purpose:** Displays a page showing the progress of all users.
   * **Key Operations:**
     + Fetches all users and calculates their daily progress.
     + Renders the data.html template with users' progress data.
2. **Update User Route:**
   * **Route:** @views.route('/update\_user/<int:user\_id>', methods=['POST'])
   * **Function:** update\_user(user\_id)
   * **Purpose:** Allows users to update their profile information.
   * **Key Operations:**
     + Validates that only the admin or the user themselves can update the info.
     + Updates the user's name and email in the database.
3. **Clear Daily Meals Route:**
   * **Route:** @views.route('/clear\_daily\_meals', methods=['POST'])
   * **Function:** clear\_daily\_meals()
   * **Purpose:** Allows users to delete their meals for the current day.
   * **Key Operations:**
     + Deletes all meal records for the current user for today from the database.
4. **Delete User Route:**
   * **Route:** @views.route('/delete\_user/<int:user\_id>', methods=['POST'])
   * **Function:** delete\_user(user\_id)
   * **Purpose:** Allows admins to delete user accounts.
   * **Key Operations:**
     + Deletes the specified user from the database.
5. **Dashboard Route:**
   * **Route:** @views.route('/dashboard')
   * **Function:** dashboard()
   * **Purpose:** Renders the dashboard page.
   * **Key Operations:**
     + Renders the dashboard.html template.
6. **Dash Routes:**
   * **Route:** @views.route('/dash') and @views.route('/dash\_app')
   * **Functions:** dash() and dash\_app()
   * **Purpose:** Redirects to or renders the Dash app embedded in the application.
   * **Key Operations:**
     + Redirects to the Dash app route or renders the dash\_app.html template.

This views.py file defines the routes and views for the main functionality of the application, including user authentication, meal logging, goal setting, and data visualization. Each route is associated with a specific function that handles the logic for rendering templates and interacting with the database.

Auth.py:



**Iteration Two: auth.py**

**Code Explanation:**

1. **Imports:**
   * Blueprint, render\_template, request, flash, redirect, url\_for: Flask components used to manage routing and rendering templates.
   * User: The model representing users in the database.
   * generate\_password\_hash, check\_password\_hash: Functions from Werkzeug to hash and verify passwords securely.
   * db: The SQLAlchemy database instance.
   * login\_user, login\_required, logout\_user, current\_user: Flask-Login functions to handle user sessions and access control.
2. **Blueprint Setup:**
   * auth = Blueprint('auth', \_\_name\_\_): Creates a Blueprint for authentication-related routes.
3. **Login Route:**
   * **Route:** @auth.route('/login', methods=['GET', 'POST'])
   * **Function:** login()
   * **Purpose:** Handles user login.
   * **Key Operations:**
     + Checks if the login form is submitted (POST request).
     + Retrieves the user from the database using the provided email.
     + Verifies the password using check\_password\_hash.
     + Logs in the user and redirects to the home page if the credentials are correct.
     + Displays error messages if the email does not exist or the password is incorrect.
4. **Logout Route:**
   * **Route:** @auth.route('/logout', methods=['GET', 'POST'])
   * **Function:** logout()
   * **Purpose:** Handles user logout.
   * **Key Operations:**
     + Ends the user session using logout\_user.
     + Redirects to the login page after logging out.
5. **Sign-Up Route:**
   * **Route:** @auth.route('/sign-up', methods=['GET', 'POST'])
   * **Function:** sign\_up()
   * **Purpose:** Handles user registration.
   * **Key Operations:**
     + Checks if the sign-up form is submitted (POST request).
     + Retrieves the user from the database to check if the email already exists.
     + Validates the form inputs (email length, first name length, password match, password length).
     + Creates a new user instance and hashes the password using generate\_password\_hash.
     + Adds the new user to the database session and commits the changes.
     + Logs in the new user and redirects to the home page if the registration is successful.
     + Displays error messages if the email already exists or if any validation fails.

This auth.py file defines the routes and views for user authentication, including login, logout, and sign-up functionalities. Each route is associated with a specific function that handles the logic for rendering templates, validating user inputs, interacting with the database, and managing user sessions.

Forms.py:



**Code Explanation:**

1. **Imports:**
   * FlaskForm: An extension of Flask that adds support for quickly building web forms with WTForms.
   * StringField, IntegerField, SubmitField: Field types from WTForms to create form classes.
   * DataRequired, NumberRange: Validators from WTForms to apply validation rules to the fields.
2. **MealForm Class:**
   * **Definition:** class MealForm(FlaskForm)
   * **Purpose:** This form is used for submitting meals with validation rules.
   * **Fields:**
     + meal\_type: A StringField for meal description with a DataRequired validator to ensure the field cannot be empty.
     + protein: An IntegerField for protein content with a DataRequired validator to ensure the field cannot be empty.
     + calories: An IntegerField for calorie content with a DataRequired validator to ensure the field cannot be empty.
     + submit: A SubmitField to add the meal when the form is submitted.
3. **GoalForm Class:**
   * **Definition:** class GoalForm(FlaskForm)
   * **Purpose:** This form allows users to set their daily nutritional goals.
   * **Fields:**
     + daily\_protein: An IntegerField for the daily protein goal with DataRequired and NumberRange(min=0) validators to ensure the field is not empty and the value is non-negative.
     + daily\_calories: An IntegerField for the daily calorie goal with DataRequired and NumberRange(min=0) validators to ensure the field is not empty and the value is non-negative.
     + submit: A SubmitField to update the goals when the form is submitted.

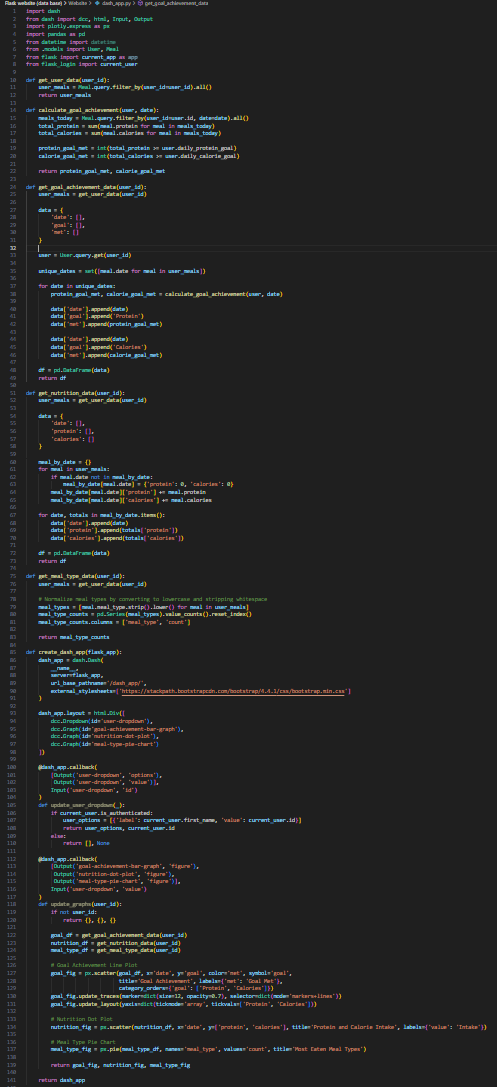
**Summary**

This forms.py file defines two form classes using FlaskForm and WTForms:

* MealForm allows users to input details about their meals, including meal description, protein content, and calories.
* GoalForm allows users to set their daily nutritional goals for protein and calorie intake.

Both forms use validators to ensure that the submitted data is valid and meets the required criteria before being processed.

Dash\_app.py:



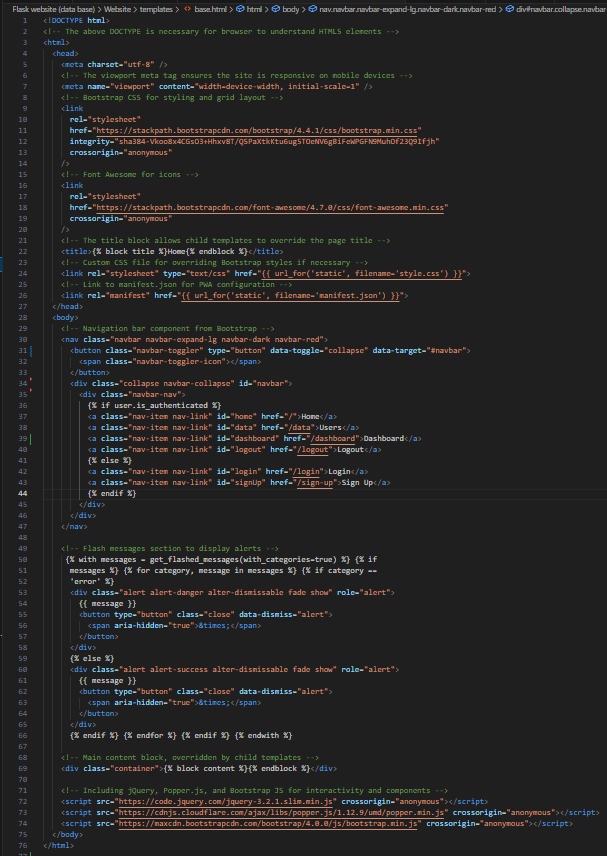
**Code Explanation:**

1. **Imports:**
   * dash, dcc, html, Input, Output: Components from the Dash framework for creating interactive web applications.
   * plotly.express as px: Plotly Express for easy plotting.
   * pandas as pd: Pandas for data manipulation.
   * datetime: Standard Python module for handling date and time.
   * User, Meal: Models from the application.
   * current\_app as app: Flask application instance.
   * current\_user: Current logged-in user from Flask-Login.
2. **Helper Functions:**
   * get\_user\_data(user\_id): Retrieves all meals logged by the user with the specified ID.
   * calculate\_goal\_achievement(user, date): Calculates whether the user's daily protein and calorie goals were met on a specific date.
   * get\_goal\_achievement\_data(user\_id): Collects data on goal achievement over time for the specified user. This data includes dates and whether protein and calorie goals were met.
   * get\_nutrition\_data(user\_id): Aggregates the user's protein and calorie intake over time.
   * get\_meal\_type\_data(user\_id): Counts the different types of meals logged by the user.
3. **Creating the Dash App:**
   * create\_dash\_app(flask\_app): Function to create the Dash app and link it with the Flask app.
   * dash.Dash(...): Initializes the Dash app with the Flask server and external stylesheets for Bootstrap.
   * **Layout:**
     + html.Div(...): Main container for the Dash app layout.
     + dcc.Dropdown(id='user-dropdown'): Dropdown for selecting users.
     + dcc.Graph(id='goal-achievement-bar-graph'): Graph for goal achievement.
     + dcc.Graph(id='nutrition-dot-plot'): Graph for nutrition data.
     + dcc.Graph(id='meal-type-pie-chart'): Pie chart for meal types.
4. **Callbacks:**
   * **Update User Dropdown:**
     + @dash\_app.callback(...): Decorator to define a callback function.
     + update\_user\_dropdown(\_): Populates the user dropdown with the current user's name and ID if authenticated.
   * **Update Graphs:**
     + @dash\_app.callback(...): Decorator to define a callback function.
     + update\_graphs(user\_id): Updates the graphs based on the selected user ID.
       - **Goal Achievement Line Plot:** Uses plotly.express.scatter to create a scatter plot for goal achievement.
       - **Nutrition Dot Plot:** Uses plotly.express.scatter to create a dot plot for protein and calorie intake.
       - **Meal Type Pie Chart:** Uses plotly.express.pie to create a pie chart showing the distribution of meal types.

**Summary**

This dash\_app.py file integrates Dash with Flask to create a user dashboard for visualizing meal data. It defines helper functions to fetch and process user data, sets up the Dash app layout with interactive graphs, and implements callbacks to update the graphs based on user interactions. This allows users to view their progress towards dietary goals, track their nutrition intake, and analyse meal types through an intuitive web interface.

Base.html:



The base.html file serves as a template for all other HTML files in the application. It includes common elements like the header, navigation bar, and footer, which can be reused across different pages.

1. **DOCTYPE and HTML Structure:**
   * <!DOCTYPE html>: Specifies that the document is HTML5.
   * <html>: Root element of the HTML document.
2. **Head Section:**
   * <meta charset="utf-8" />: Specifies the character encoding for the HTML document.
   * <meta name="viewport" content="width=device-width, initial-scale=1" />: Ensures the site is responsive on mobile devices.
   * <link rel="stylesheet" href="...">: Includes Bootstrap and Font Awesome for styling and icons.
   * <title>{% block title %}Home{% endblock %}</title>: Title block that child templates can override.
   * <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css') }}">: Link to the custom CSS file.
   * <link rel="manifest" href="{{ url\_for('static', filename='manifest.json') }}">: Link to the PWA configuration file.
3. **Body Section:**
   * **Navigation Bar:**
     + <nav class="navbar navbar-expand-lg navbar-dark navbar-red">: Bootstrap navigation bar with custom styling.
     + Contains links for Home, Users, Dashboard, Login, Sign Up, and Logout, displayed based on user authentication status ({% if user.is\_authenticated %}).
   * **Flash Messages:**
     + {% with messages = get\_flashed\_messages(with\_categories=true) %}: Displays flash messages for user feedback.
     + Categorizes messages as either 'error' or 'success' and displays them with appropriate styling.
   * **Main Content:**
     + <div class="container">{% block content %}{% endblock %}</div>: Main content block to be overridden by child templates.
4. **JavaScript Includes:**
   * <script src="..."></script>: Includes jQuery, Popper.js, and Bootstrap JS for interactivity and components.

**Summary**

The base.html template sets up the basic structure and styling for the web application. It includes a responsive navigation bar, placeholders for flash messages, and a content block that can be extended by other templates. This modular approach ensures consistency across the site and makes it easier to manage common elements.

Dashboard.html:

A screen shot of a computer code

Description automatically generated

1. **Template Inheritance:**
   * {% extends "base.html" %}: Inherits the structure and styling from base.html.
2. **Title Block:**
   * {% block title %}Dashboard{% endblock %}: Sets the page title to "Dashboard".
3. **Content Block:**
   * {% block content %}...{% endblock %}: Contains the main content for the dashboard page.
   * Displays a personalized greeting using {{ user.first\_name }}.
   * Embeds an iframe to display the Dash application (dash\_app) within the page.

**Summary**

The dashboard.html file extends the base.html template to create a user-specific dashboard. It includes a personalized greeting and embeds an iframe to show analytical data from the Dash app, all within a responsive layout.

Data.html (I understand this was poor naming but this is the users page):



1. **Template Inheritance:**
   * {% extends "base.html" %}: Inherits the base structure from base.html.
2. **Title Block:**
   * {% block title %}Data{% endblock %}: Sets the page title to "Data".
3. **Custom CSS:**
   * Defines responsive design styles to ensure cards and tables fit on small screens and are scrollable if necessary.
4. **Main Content Block:**
   * **Welcome Card:** Displays a personalized welcome message to the logged-in user.
   * **User Data Table:**
     + A table that lists all users, allowing the logged-in user to update their information.
     + Includes a form for each user, allowing updates to their name and email. Only the logged-in user can see and edit their email.
   * **Progress Cards:**
     + Displays progress bars for each user's protein and calorie goals.
     + Progress bars visually represent the percentage of the user's goals achieved for the day.

**Summary**

The data.html file extends the base.html template and provides a user interface for viewing and updating user data. It includes a responsive table for user information, forms for updating user details, and progress cards to display daily protein and calorie goal achievements. The page is designed to be user-friendly and informative, allowing users to easily manage their data and track not only their own progress but also their peers.

Home.html:

A screen shot of a computer program

Description automatically generated

1. **Template Inheritance:**
   * {% extends "base.html" %}: Inherits the base structure from base.html.
2. **Title Block:**
   * {% block title %}Home{% endblock %}: Sets the page title to "Home".
3. **Content Block:**
   * **Greeting Message:**
     + Personalized greeting message displaying the user's first name.
   * **Meal Entry Form:**
     + A form for adding new meals, including meal type, protein, and calories. Uses Flask-WTF for form handling and validation.
   * **Today's Meals Section:**
     + Displays a list of meals added for today with their protein and calorie content.
     + Shows the total protein and calories consumed today.
     + Includes charts to visually represent protein and calorie intake using Chart.js.
   * **Clear Meals Button:**
     + Button to clear all meals recorded for today.
   * **Goal Setting Section:**
     + Displays the user's current daily protein and calorie goals.
     + A form to update these goals, using Flask-WTF for form handling and validation.
4. **Chart.js Integration:**
   * **Protein Chart:**
     + A pie chart showing the proportion of protein consumed versus remaining protein to meet the daily goal.
   * **Calorie Chart:**
     + A bar chart showing the calories consumed versus remaining calories to meet the daily goal.

**Summary**

The home.html file is the main page where users interact with the protein tracking application. It includes forms for logging meals and setting goals, displays today's meal entries and nutritional intake, and provides visual feedback through charts. This page is designed to be user-friendly and informative, allowing users to easily track and manage their protein and calorie intake.

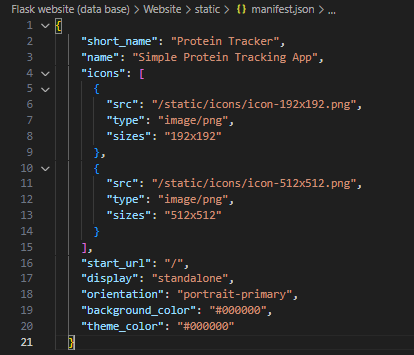
Manifest.json:

**What is a Progressive Web App (PWA)?**

A Progressive Web App (PWA) is a web application that provides an app-like experience on mobile devices and desktops. PWAs are designed to work offline, load quickly, and look and feel like native apps. They can be added to the home screen of a device, giving users a seamless and engaging experience.

**Role of manifest.json in a PWA**

The manifest.json file is essential for making a web app a PWA. It gives the browser information about the app, such as its name, icons, and how it should behave when installed on a device.



1. **short\_name and name:**
   * These specify the app's name for different contexts. The short name is used where space is limited, and the full name is used elsewhere.
2. **icons:**
   * These are the icons used for the app on the home screen and app launcher, in different sizes.
3. **start\_url:**
   * This is the URL that the app opens to when launched from the home screen.
4. **display:**
   * This setting makes the app appear like a standalone application without browser UI elements.
5. **orientation:**
   * This locks the app's orientation to portrait mode.
6. **background\_color and theme\_color:**
   * These define the background color when the app is launched and the theme color for the app's UI.

**Summary**

The manifest.json file is a key component of a PWA, providing the necessary information to ensure the app can be installed and function like a native app on users' devices.

**Note:**

**The login and sign-up HTML pages had minimal changes that did not display any complex programming skills so will not be mentioned here. This also includes the style.css file.**

**Visual output:**

Login page:

A black and white striped background

Description automatically generated

Sign up page:

A screenshot of a video game

Description automatically generated

Home page:

A screenshot of a computer

Description automatically generated

Users page:

A screenshot of a computer

Description automatically generated

Dashboard:

A screenshot of a computer

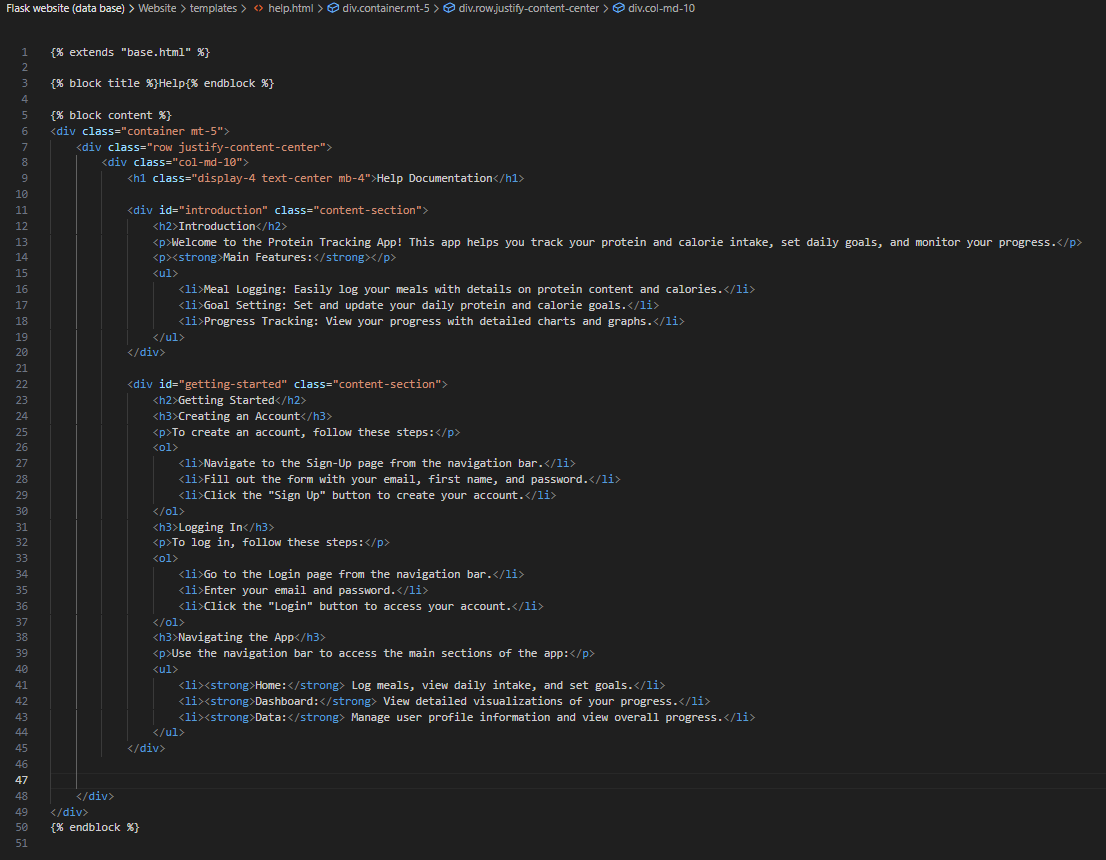
Description automatically generated

**Iteration Three:**

In this iteration I will be adding the final touches to my project like adding a word text file to store my data base in a different manor, also I want to include a help section in my project for obvious reasons. That is what you can expect from this iteration.

Help section:

Help.html:



This is intuitive and just displays a help section on the website for users if needed.

Docx\_handeling.py:

A screen shot of a computer program

Description automatically generated

The docx\_handling.py file is designed to manage user data stored in an SQLite database and export it into a Word document. This script ensures that user and meal information is organized and easily accessible in a document format. Below is a detailed explanation of the key functions and their purposes:

1. **Clearing the Document:**
   * clear\_document(doc\_path): This function opens the specified Word document and removes all existing content, including paragraphs and tables. This ensures the document starts fresh each time the script runs.
2. **Adding a Title:**
   * add\_title(doc\_path, title\_text): Adds a title to the Word document. The title text is styled as a 'Title' to ensure it stands out.
3. **Fetching Data from the Database:**
   * fetch\_users\_from\_db(db\_path): Connects to the SQLite database and retrieves user information, including user ID, name, email, and daily goals for protein and calories.
   * fetch\_meals\_from\_db(db\_path): Connects to the SQLite database and retrieves meal information, including user ID, protein content, calorie content, and the date of the meal.
4. **Adding User Login Information:**
   * add\_user\_logins(doc\_path): Adds a heading "User Login Data" and a table with columns for user ID, name, email, protein goal, and calorie goal. This table is then populated with user data fetched from the database.
5. **Adding User Meal Sections:**
   * add\_user\_meals(doc\_path, user\_name, user\_id): Adds a heading for each user's meals and creates a table to log meal data, including protein, calories, and date. If a user already exists in the document, an error is raised to prevent duplication.
6. **Adding Rows to Tables:**
   * add\_row\_to\_table(doc\_path, table\_number, row\_content): Adds a new row to the specified table in the document. It ensures the new row's content matches the table's column count and checks for duplicate entries before adding.
7. **Initial Setup and Data Insertion:**
   * The document is first cleared and titled using clear\_document and add\_title.
   * User and meal data are fetched from the database using fetch\_users\_from\_db and fetch\_meals\_from\_db.
   * The script checks if user login information needs to be added by calling check\_start\_up\_procedures.
   * Finally, for each user and their respective meals, add\_user\_meals and add\_row\_to\_table are called to populate the document with the fetched data.

This script efficiently organizes and exports database information into a structured Word document, providing a clear and accessible format for reviewing user and meal data.

VISUAL OUTPUT:

A close-up of a document

Description automatically generated

A screenshot of a computer

Description automatically generated

## 5.2 Code Comments

Code commenting is important so that people who did not create the code (like Mr Khan) can understand what is going on. Although the coding languages I am using are simple enough to read, all the dependencies and random functions might get overwhelming to read, this is why code commenting is useful and important.

Examples of me using code commenting:

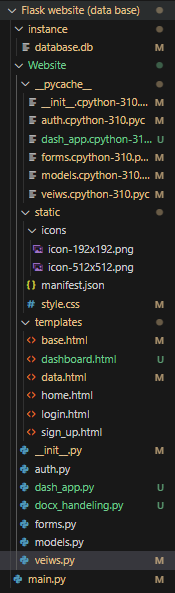


This is a short snippet of my code in the views.py file of my project. As you can see there is a lot of code commenting so that its easy for me to look back on to understand what is happening and even someone with no coding experience to get the idea of what is going on.

## 5.3 Separation of UI from Backend

Separating the user interface (UI) from the backend logic is a fundamental practice in web development. This separation enhances maintainability, scalability, and readability of the code. It allows developers to work on the front-end and back-end independently, making the development process more efficient and organized.

Project structure:



In this project, we have clearly separated the UI components (HTML, CSS) from the backend logic (Python scripts).

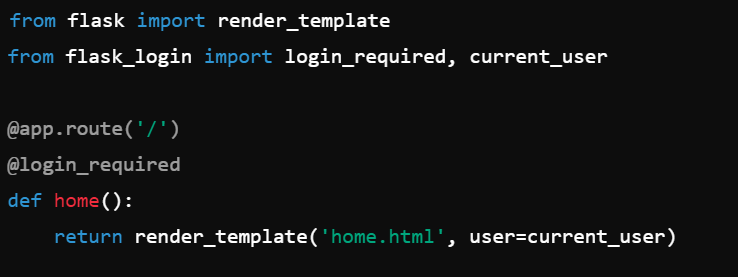
Separation techniques:

**Flask Templates**

Flask uses templates to separate HTML content from Python logic. The render\_template function is used to render HTML files and pass data from the backend to the UI.

Here is an example using the home page:

The view function (in python) handling the back end / route of the page:



The template being displayed

A computer screen with white text

Description automatically generated

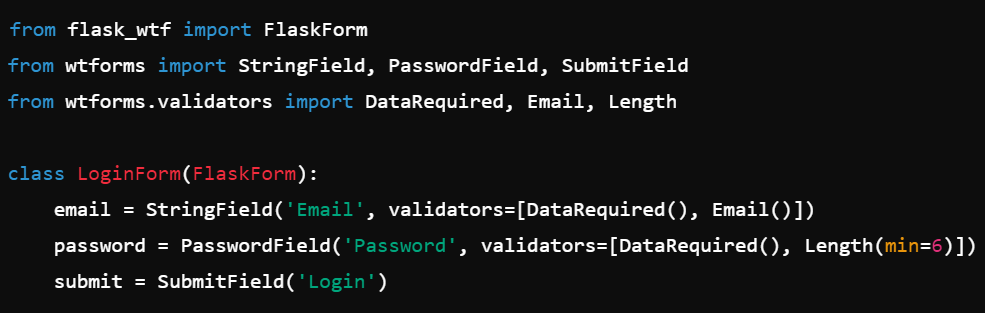
This is the absolute bare bones of how this works, we use the route to handle the post and get requests and also tell the page what to display or who like the user. This is the back end. Then we use the html file to display the front end and user interface with the information given from the back end.

**Flask Form Handling:**

Forms are handled using Flask-WTF, which separates form validation and processing from the HTML structure. Forms are defined in forms.py and rendered in templates with validation logic in the backend.

Here is an example of a form and it being rendered in a template:

Form class (python):

****

Rendering the form in a template (html):

**A computer screen shot of white text

Description automatically generated**

As you can see, a form is created in the back end using python then in the front end it can be displayed. This is really useful for handling all the logic and computation, as this should be done in a language like python and would be nearly impossible or a waste of time to do in html that is only used for visual display operations.

These were the key things I thought were important for separation from the front and back end of my program. But I am sure I missed other areas where this is done too.

## 5.4 Inter-Code Communication

Inter-code communication refers to the methods and protocols used for different parts of an application to communicate with each other. In a web application, this typically involves the interaction between the frontend (UI) and the backend (server logic), as well as between different backend components.

**Overview**

In our project, inter-code communication is primarily handled through the following mechanisms:

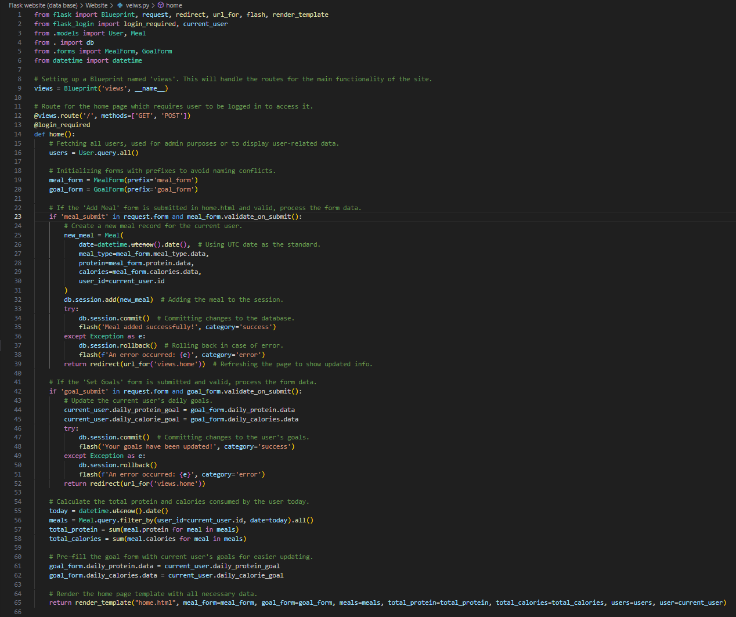
* HTTP Requests and Responses
* Flask Blueprints
* Database Operations
* Form Handling
* API Integration (if applicable)

Each of these components communicates with the others to ensure the application functions correctly and efficiently.

**HTTP Requests and Responses**

The primary mode of communication between the frontend and backend is through HTTP requests and responses. When a user interacts with the frontend, such as submitting a form, the browser sends an HTTP request to the server. The server processes this request and sends back an HTTP response.

Here is an example of HTTP request using views.py:



In this example, when the user submits the meal form, an HTTP POST request is sent to the server. The server processes the form data, updates the database, and sends an HTTP response back to the browser to update the page.

**Flask Blueprints**

Flask Blueprints allow us to organize our application into modules, each with its own routes and view functions. This modular approach facilitates communication between different parts of the application.

For example, in auth.py:



Here, the auth Blueprint manages authentication-related routes, ensuring that these routes can communicate with the rest of the application seamlessly. Like how the login function is used in the login page, to check if they completed the login details, check if the login details are correct and then finally create a user instance of them using flask\_login.

**Database Operations**

Database operations are a critical aspect of inter-code communication. They involve reading from and writing to the database to store and retrieve user data, meal entries, and goals.

There are multiple instances where the models.py file that creates the database and its tables and everything is used in other files. Almost every file in my project uses it to manipulate, store or view data in the database.

## 5.5 Good General Techniques

Other than commenting as we talked about before, there are a few other good programming techniques that I tried to maintain throughout the development process.

Version control:

Using GitHub helped me maintain versions of my code in case anything went wrong, and I needed to go back to a previous working version.

Security practices:

Password Hashing: Passwords are hashed before being stored in the database to enhance security.

A black screen with white text

Description automatically generated

Login Required Decorator: Protecting routes with the @login\_required decorator ensures that only authenticated users can access certain parts of the application.

A screen shot of a computer

Description automatically generated

User input validation:

Form Validation: Ensuring that user inputs are validated before processing them helps prevent errors and security vulnerabilities.

A screen shot of a computer

Description automatically generated

Error handling:

Try-Except Blocks: Proper error handling ensures that the application can handle unexpected situations gracefully without crashing.

A computer screen with white text

Description automatically generated

Adding a help section:

Adding a help section even though my project is really simple and easy to navigate is a good general practice to help users:

A screenshot of a computer

Description automatically generated

­­

## 5.6 Querying and Manipulation of Data

Querying and manipulating data is a crucial part of any web application, enabling the retrieval, updating, and deletion of information stored in a database. In this project, SQLAlchemy is used as the Object-Relational Mapping (ORM) tool to interact with the SQLite database. Below, I explain the key aspects of querying and manipulating data in the application.

**SQLAlchemy ORM**

SQLAlchemy provides a high-level abstraction over database operations, allowing us to work with database records as if they were regular Python objects. This approach simplifies complex database interactions and enhances code readability.

**Example: Querying Data**

In the application, querying data is essential for retrieving user and meal information. Below are examples of how we perform different types of queries.

Fetching all users:



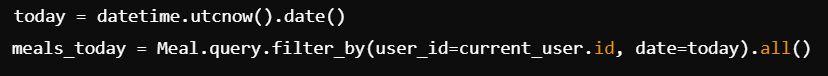
This query retrieves all records from the User table.

Fetching meals for the current user:



This query retrieves all meal records associated with the currently logged-in user.

Fetching meals for a specific date:



**Example: Manipulating Data**

Data manipulation includes adding, updating, and deleting records in the database. Below are examples of how these operations are performed.

Adding a new meal:

A computer screen with white text

Description automatically generated

This code snippet creates a new meal record and adds it to the database.

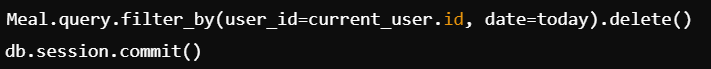
Updating user goals:

A black background with white text

Description automatically generated

This code updates the daily protein and calorie goals for the current user.

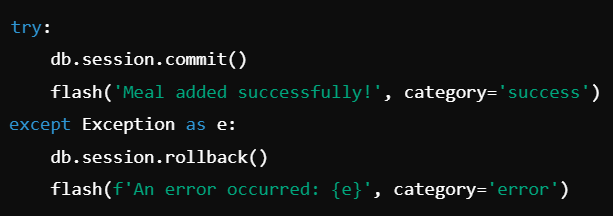
Deleting a meal:



This code deletes all meal records for the current user on the current date.

**Ensuring Data Integrity**

To ensure data integrity and avoid inconsistencies, we use transactions and handle exceptions appropriately. For instance, when adding a new meal, if an error occurs during the commit operation, we rollback the transaction to maintain database integrity.



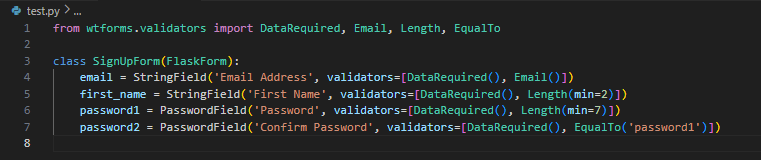
## 5.7 Defensive Programming

Defensive programming is a practice of writing code that anticipates and handles potential errors and unexpected inputs gracefully. This approach improves the robustness and reliability of the application by ensuring that it can cope with various edge cases and erroneous situations without crashing or producing incorrect results.

**Input Validation**

Validating user input is a crucial aspect of defensive programming. In our application, we use Flask-WTF to validate form inputs and ensure that they meet the required criteria before processing them.

Example: Sign-Up Form Validation:

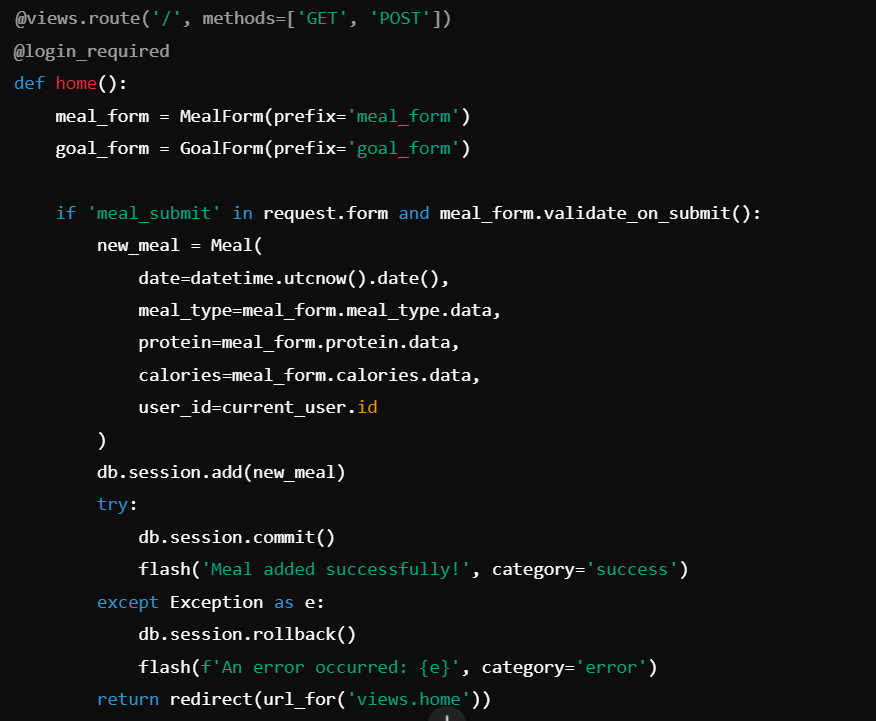


This form ensures that the email is valid, the first name is at least 2 characters long, and the passwords are at least 7 characters long and match each other.

**Error Handling**

Proper error handling prevents the application from crashing and provides meaningful feedback to the user. We use try-except blocks to catch exceptions and handle them appropriately.

Example: Adding a New Meal:



If an error occurs while adding a new meal, the transaction is rolled back, and an error message is displayed to the user.

**Handling Invalid User Actions**

We handle cases where users attempt invalid actions, such as trying to update or delete records that they do not have permission to modify.

Example: Updating User Data:

A screenshot of a computer program

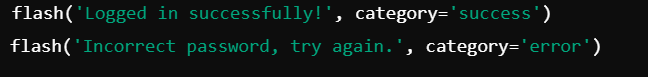
Description automatically generated

This route checks if the current user has permission to update the specified user data. If not, an error message is displayed, and the user is redirected.

**Providing Meaningful Feedback**

We use flash messages to provide users with feedback on their actions, indicating whether an operation was successful or if an error occurred.

Example: Flash Messages



These messages inform the user of the outcome of their actions, helping them understand what happened and how to proceed.

# Testing

## 6.1 Functional Testing / Test plan

Functional testing is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of functional tests is to check whether the system behaves as expected and performs its functions correctly. In this project, we will perform functional testing to ensure that all features work as intended.

**Testing Strategy**

1. **Unit Testing**: Testing individual components or functions to ensure they work as expected in isolation.
2. **Integration Testing**: Testing the interaction between different components to ensure they work together as expected.
3. **End-to-End Testing**: Testing the complete workflow from the user's perspective to ensure the entire system works as intended.

**Test Cases**

**1. User Authentication**

* **Test Case: User Registration**
  + **Description**: Verify that a new user can register successfully with valid inputs.
  + **Steps**:
    1. Navigate to the sign-up page.
    2. Enter valid email, first name, and password.
    3. Click the "Sign Up" button.
  + **Expected Result**: The user should be registered successfully, and a success message should be displayed.
* **Test Case: User Login**
  + **Description**: Verify that a registered user can log in successfully with valid credentials.
  + **Steps**:
    1. Navigate to the login page.
    2. Enter registered email and password.
    3. Click the "Login" button.
  + **Expected Result**: The user should be logged in successfully, and a success message should be displayed.
* **Test Case: User Logout**
  + **Description**: Verify that a logged-in user can log out successfully.
  + **Steps**:
    1. Click the "Logout" button.
  + **Expected Result**: The user should be logged out successfully, and the login page should be displayed.

**2. Meal Logging**

* **Test Case: Add a New Meal**
  + **Description**: Verify that a user can add a new meal entry with valid inputs.
  + **Steps**:
    1. Navigate to the home page.
    2. Enter valid meal description, protein content, and calorie content.
    3. Click the "Add Meal" button.
  + **Expected Result**: The meal should be added successfully, and a success message should be displayed.
* **Test Case: Clear Daily Meals**
  + **Description**: Verify that a user can clear all meals recorded for the day.
  + **Steps**:
    1. Navigate to the home page.
    2. Click the "Clear Today's Meals" button.
  + **Expected Result**: All meals for the day should be cleared, and a success message should be displayed.

**3. Goal Setting**

* **Test Case: Update Daily Goals**
  + **Description**: Verify that a user can update their daily protein and calorie goals with valid inputs.
  + **Steps**:
    1. Navigate to the home page.
    2. Enter valid protein and calorie goals.
    3. Click the "Update Goals" button.
  + **Expected Result**: The goals should be updated successfully, and a success message should be displayed.

**4. Data Visualization**

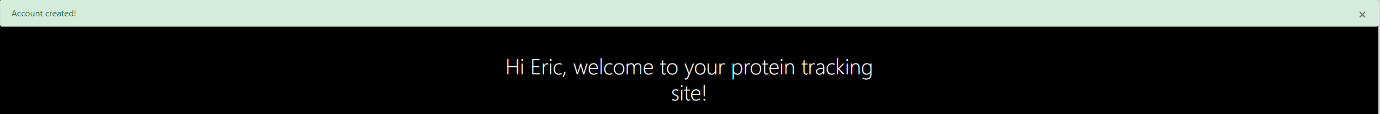
* **Test Case: View Dashboard Analytics**
  + **Description**: Verify that a user can view their analytics on the dashboard.
  + **Steps**:
    1. Navigate to the dashboard page.
  + **Expected Result**: The dashboard should display the user's analytics, including charts for protein and calorie intake.
* **Test Case: View User Progress**
  + **Description**: Verify that a user can view the progress of all users on the data page.
  + **Steps**:
    1. Navigate to the data page.
  + **Expected Result**: The data page should display the progress bars for protein and calorie goals for all users.

### 6.2 Test Results

**User Authentication:**

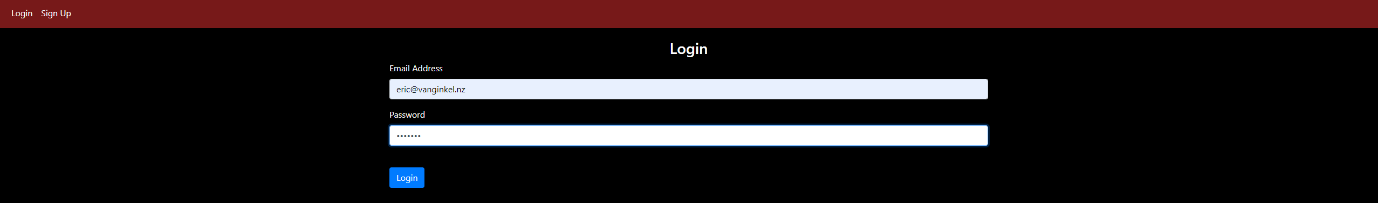
Test Case: User Registration:

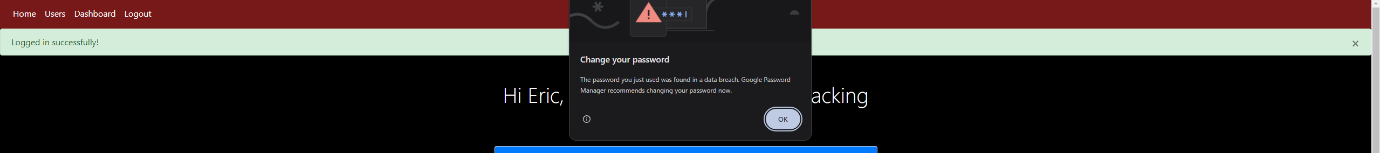
****

****

It works as intended and signs a user up and adds their information to the database and are an actual user.

Test Case: User Login:





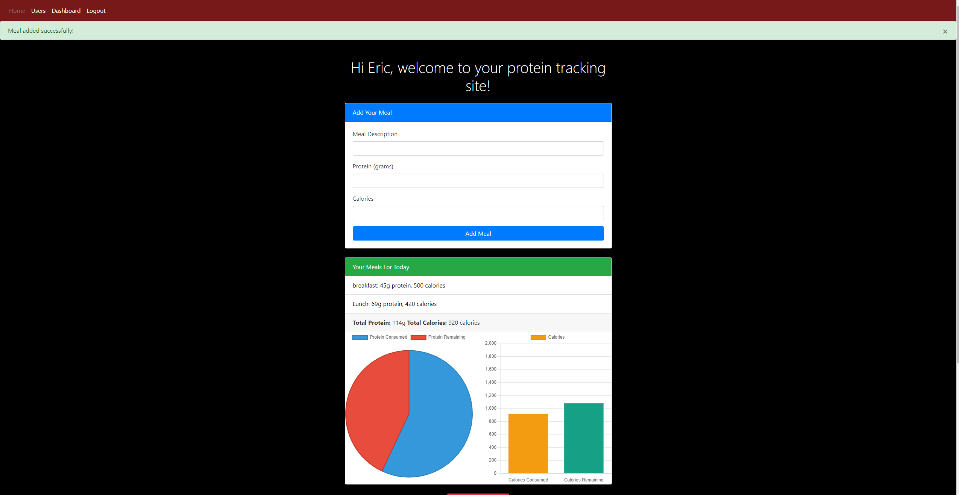
The log in function works as expected, google even recommends a safer password interesting side note.

Test Case: User Logout:

Yes, the log out function works and redirects the user to the log in page and they have no access to anything they had before signed in.

**Meal logging:**

Add a New Meal

****

It works as intended, I can add a meal and the meal is shown with statistics comparing to my daily intake goals. Nice!

Clear Daily Meals



Clearing Daily Meals works and all my meals are deleted for that day only.

**Goal Setting**

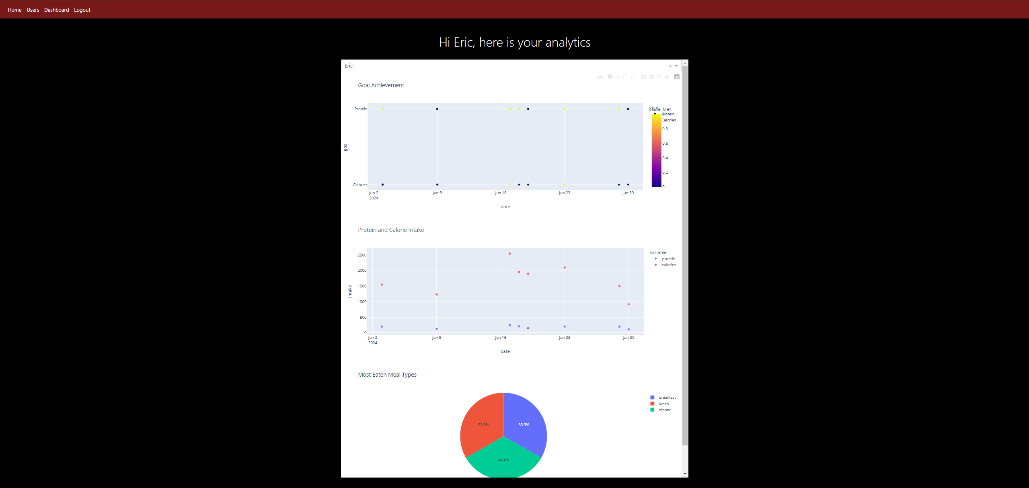
Update Daily Goals



It works and I can change the daily intake goal values and it changes all the graphs with the new data as well.

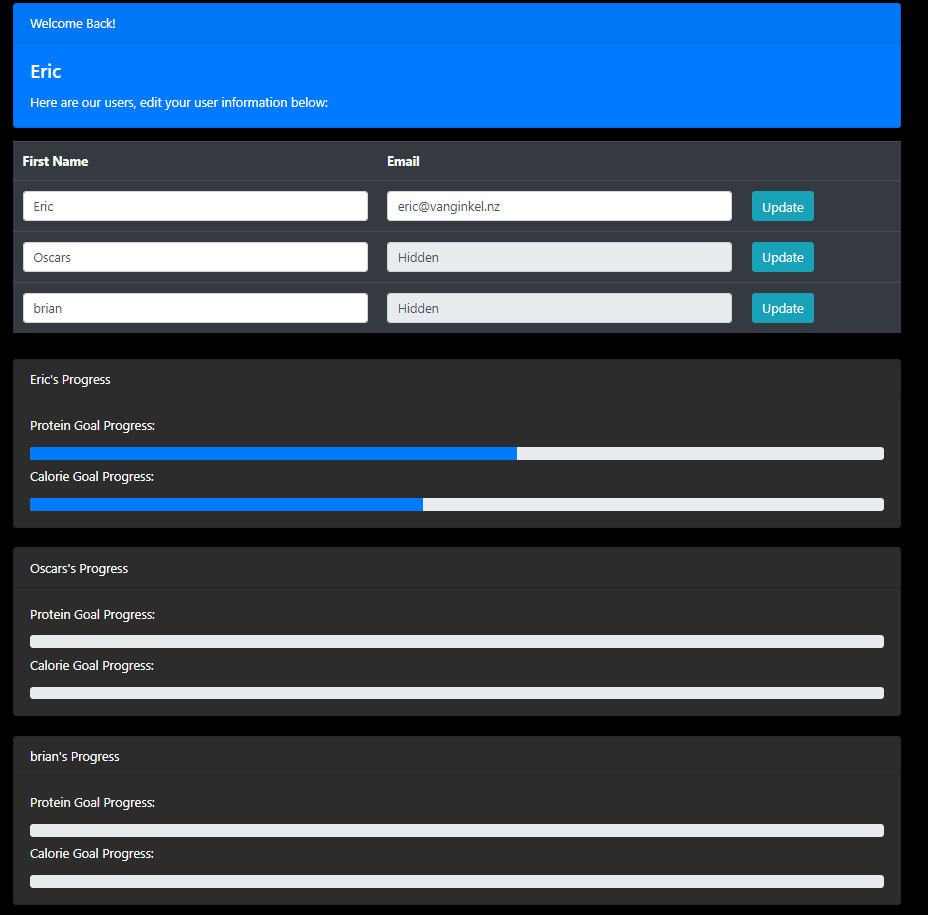
**Data visualization**

**View Dashboard Analytics**

****

Works as intended and I can see all my data that has been accumulated over time with stats

**View User/s Progress**

****

No issues here, works how I want it to. I can see not only the user I am logged in as graph but also the other people’s graphs. And I can only change and see my own login information not anyone else’s. As you can see bellow what happens if I try to change another users login information:



# Technical Documentation

## 7.1 Externally Sourced Code

In this project, several external libraries and frameworks have been utilized to enhance functionality and streamline development. Utilizing well-established libraries not only speeds up the development process but also ensures that the application adheres to industry standards and best practices.

**1. Flask**

****

**Description**: Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it a popular choice for developers to build web applications quickly.

**Usage**:

* **Routing**: Flask's routing mechanism is used to define various endpoints for the application.
* **Templates**: Flask uses Jinja2 as its template engine to render HTML pages dynamically.

**Example**:

A computer screen with white text

Description automatically generated

**2. Flask-WTF**



**Description**: Flask-WTF integrates the WTForms library with Flask, providing form validation and rendering.

**Usage**:

* **Form Handling**: Used to create and validate forms for user input, such as registration and login forms.

**Example**:



**3. SQLAlchemy**



**Description**: SQLAlchemy is an SQL toolkit and Object-Relational Mapping (ORM) library for Python. It provides a full suite of well-known enterprise-level persistence patterns.

**Usage**:

* **Database Management**: Used to define and manipulate the database schema and perform CRUD operations.

**Example**:

A screen shot of a computer program

Description automatically generated

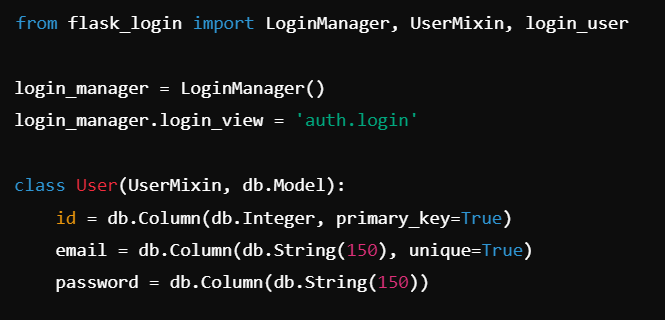
**4. Flask-Login**

**Description**: Flask-Login provides user session management for Flask. It handles the common tasks of logging in, logging out, and remembering a user’s session over extended periods.

**Usage**:

* **User Authentication**: Manages user sessions and access control.

**Example**:



**5. Chart.js**

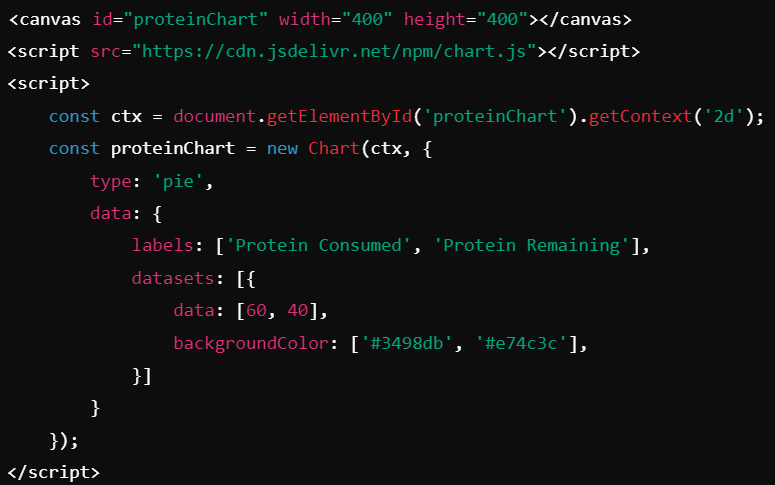


**Description**: Chart.js is a JavaScript library that allows developers to create beautiful and responsive charts easily.

**Usage**:

* **Data Visualization**: Used to create dynamic charts displaying the user's protein and calorie intake.

**Example**:



**6. Bootstrap**

A purple logo with text

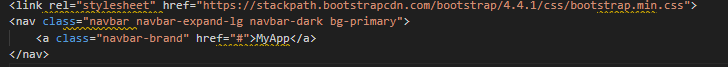
Description automatically generated

**Description**: Bootstrap is a popular CSS framework for developing responsive and mobile-first websites.

**Usage**:

* **Styling**: Used for the layout and styling of the web pages, ensuring they are responsive and visually appealing.

**Example**:



**7. Dash and Plotly**



**Description**: Dash is a productive Python framework for building web applications. Built on top of Flask, Plotly.js, and React.js, Dash is ideal for building data visualization apps with highly custom user interfaces. Plotly is a graphing library that makes interactive, publication-quality graphs online.

**Usage**:

* **Data Visualization**: Dash is used to create interactive web-based data visualizations, and Plotly is used to generate the charts and graphs.

**Example**:



**8. Pandas**

A blue and black text

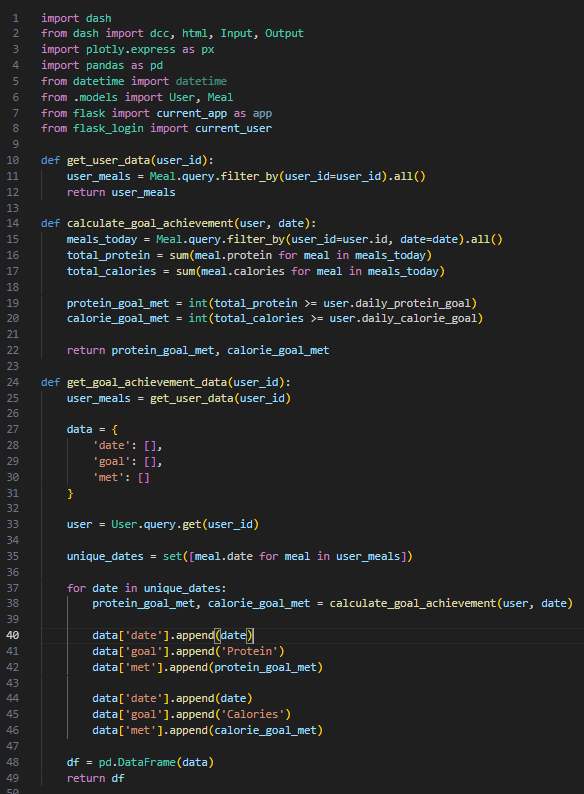
Description automatically generated

**Description**: Pandas is a powerful, open-source data manipulation and analysis library for Python. It provides data structures and functions needed to manipulate structured data seamlessly.

**Usage**:

* **Data Handling**: Pandas is used for data manipulation and analysis, making it easy to load, clean, and analyze data efficiently.

**Example**:



## 7.2 Advanced Techniques

In this project, several advanced programming techniques have been employed to ensure that the application is robust, scalable, and efficient. These techniques demonstrate a deep understanding of software development principles and showcase the ability to solve complex problems effectively.

**1. Object-Relational Mapping (ORM) with SQLAlchemy**

**Description**: Object-Relational Mapping (ORM) allows developers to interact with the database using Python objects instead of raw SQL queries. SQLAlchemy, a powerful ORM library, is used in this project to manage database interactions.

**Implementation**:

* **Model Definition**: SQLAlchemy models are defined for User and Meal, representing the corresponding database tables.
* **Relationships**: A one-to-many relationship is established between User and Meal to link meals to specific users.
* **CRUD Operations**: SQLAlchemy provides an easy-to-use interface for creating, reading, updating, and deleting records in the database.

**2. User Authentication with Flask-Login**

**Description**: User authentication is a critical aspect of web applications. Flask-Login is used to handle user session management, ensuring that users can securely log in, log out, and access protected routes.

**Implementation**:

* **Login Management**: Flask-Login provides decorators and functions to manage user login sessions.
* **Password Hashing**: Passwords are securely hashed using werkzeug.security to protect user credentials.
* **Session Handling**: The login\_user and logout\_user functions manage user sessions efficiently.

**Example**:

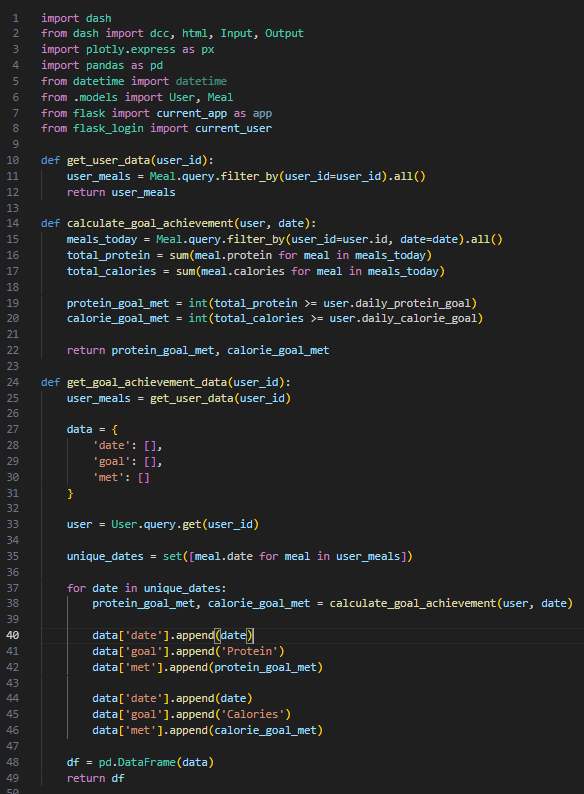


**3. Data Visualization with Dash and Plotly**

**Description**: Dash and Plotly are used to create interactive data visualizations, providing users with insights into their dietary habits. These libraries enable the creation of dynamic charts and graphs embedded directly into the web application.

**Implementation**:

* **Dash Integration**: Dash is integrated with Flask to serve interactive web applications.
* **Plotly Graphs**: Plotly is used to generate high-quality, interactive graphs that visualize user data.
* **Callback Functions**: Dash callbacks are used to update graphs based on user input and data changes.

**Example:  
**

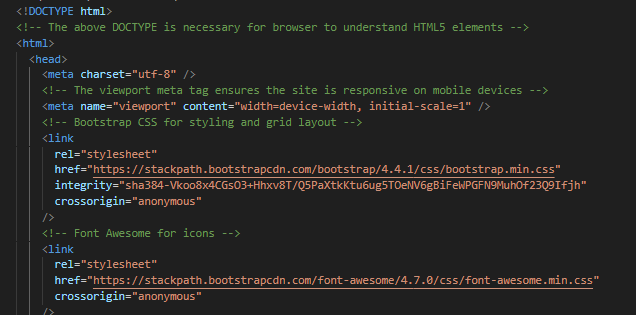
**4. Responsive Design with Bootstrap**

**Description**: Bootstrap is a popular front-end framework that helps create responsive and mobile-first web pages. It ensures that the application is accessible and visually appealing on all devices.

**Implementation**:

* **Grid System**: Bootstrap's grid system is used to create a flexible and responsive layout.
* **Predefined Classes**: Utilizes Bootstrap's predefined classes for consistent styling and responsive behavior.
* **Customization**: Custom CSS is added to override default Bootstrap styles to match the application's design requirements.

**Example**:



**5. Progressive Web App (PWA) Capabilities**

**Description**: Implementing Progressive Web App (PWA) capabilities allows the application to provide a native app-like experience. PWAs are reliable, fast, and engaging, offering features such as offline access and push notifications.

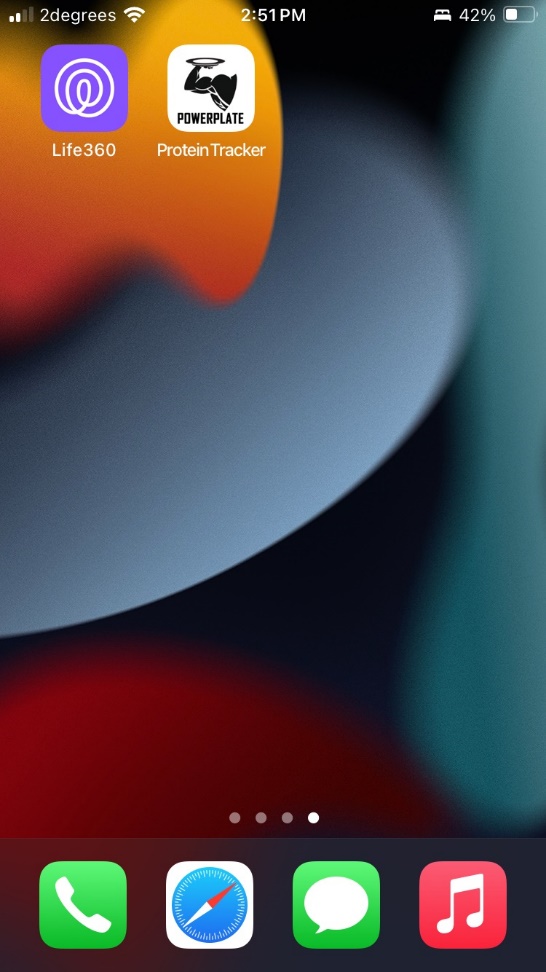
**Implementation**:

* **Service Workers**: Service workers are used to cache assets and handle offline access.
* **Manifest File**: A web app manifest is included to define the app's metadata, enabling it to be installed on user devices.
* **Responsive Design**: Ensures the application works seamlessly across different devices and screen sizes.

**Example**:

A screen shot of a computer program

Description automatically generated

A screenshot of a login page

Description automatically generated

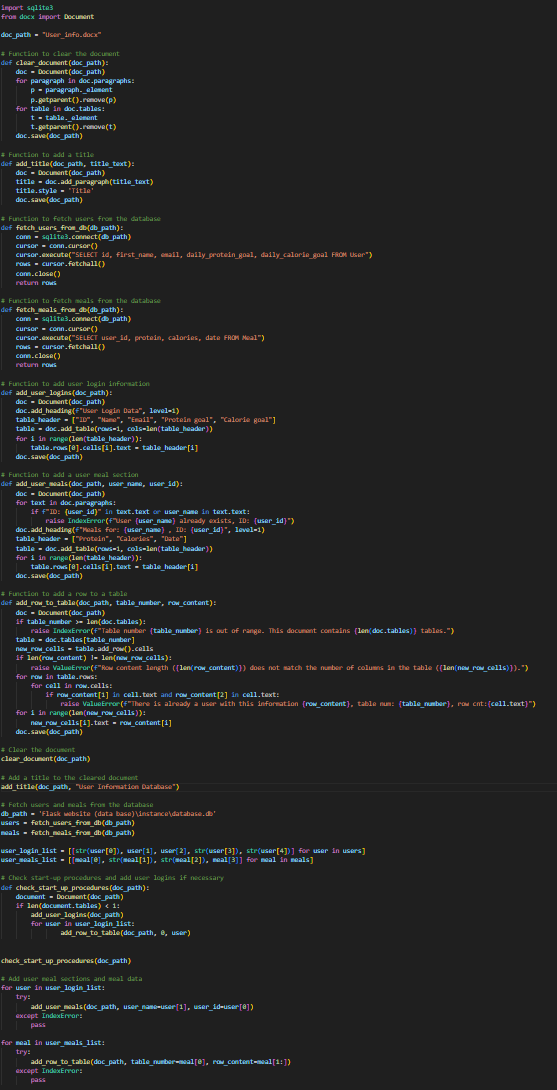
**6. DOCX Processing and Data Manipulation**

**Description**: In addition to managing data within a database, this project includes functionality for exporting and manipulating data in a DOCX file. This allows users to generate reports or summaries of their data in a widely-used document format. Or as a separate storage option for the administrator (me)

**Implementation**:

* **Reading and Writing DOCX Files**: The python-docx library is used to handle DOCX files, allowing the project to create, modify, and save Word documents.
* **Database Interaction**: Data is fetched from the SQLite database using SQLAlchemy, and this data is then written to a DOCX file.
* **Automation**: The process is automated to regularly update the DOCX file with the latest data from the database, ensuring that reports are always up-to-date.

**Example**:



# End user considerations:

**Usability**

**Ease of Use:**

I think that my app is designed to be easy to use and user friendly. with user-friendly navigation and intuitive interfaces. Clear labels, simple forms, and straightforward navigation ensure that users can easily understand how to use the app without requiring extensive technical knowledge. (help page if they don’t know)

**Responsive Design:**

My application works across various devices, including desktops, tablets, and smartphones. This flexibility is crucial for users who need to log their meals and check their progress on the go.

**Functionality**

**Accurate Meal Logging:**

Users can log their meals with precise details about the meal type, protein content, and calories. This functionality helps users keep track of their daily intake accurately, aiding them in meeting their dietary goals.

**Goal Setting and Progress Tracking:**

The app allows users to set daily protein and calorie goals and tracks their progress towards these goals. Visual feedback through charts and graphs helps users understand their progress and make informed dietary choices.

**Data Management:**

User data is securely stored and easily retrievable, allowing users to view and manage their past entries. This includes the ability to update profile information and reset daily meal logs.

**Security**

**Secure Authentication:**

The app has secure authentication mechanisms, making sure that the user data is protected. Passwords are hashed, and user sessions are managed securely to prevent unauthorized access.

**Privacy:**

User data is only available to themselves, other than the competitive nature of the app where we allow users to see other peoples goal progress daily.

**Support and Documentation**

**Help Documentation:**

Comprehensive help documentation is available within the app, guiding users on how to use different features effectively. This includes step-by-step instructions

# Relevant implications:

## 

**Social –**

# 

Social implications of my app are an important aspect to cover. My app needs to cater to my intended audience and others. In the context of disabilities, I made sure my website included fonts that were large enough to read for those with poor eyesight and easy enough to navigate for young kids’ or people with other disabilities. However, my colour choices could have been better for people with coloured blindness. Other than this there is no racist, sexist or homophobic literature or pictures on my site. There is nothing to offend others and is strictly professional. Privacy and data collection is also important to note. My App collects a lot of data from the user such as dietary and email and passwords. I make sure it is stored safe and nothing is shared. Another social implication is politics. My site does not talk anywhere about representing any beliefs or parties so this can be ignored.

**Cultural –**

Cultural implications include languages, customs, cultures and beliefs. In context of my site, for languages, I could have added multiple languages as a feature to make it more accessible. Making sure that translations are correct and appropriate in the context and language. Visually, it is important not to add any pictures, emblems, symbols or colours that could possibly carry negative meanings in certain cultures. So, researching what goes into my site is important and making sure it is not anything negative to anyone. In context of the location of the business I am making a website for, it is in New Zealand. So, taking into consideration the Māori language and culture into my site could make a huge difference for the people of New Zealand. Adding Te reo Māori is important especially in New Zealand as not only the government but also the people of New Zealand have been advocating for a greater use of the language. This could benefit the sight in the eye of the public and benefit potential Māori users.

**Legal –**

I do not use any information on the site that has potential for copy right infringements. Also, no money is collected from my site as it is open source and free at the moment. In terms of legality, I have little to worry about other than data collection and privacy laws. I will keep my users data safe and not share it with anyone.

**Usability and Accessibility –**

In order to create a functional site for everyone to use it must be designed in a way that does not overwhelm the user and is simple to navigate. Using fonts that are large enough for everyone to read and are clear enough to read is important. Also, consideration to colours should be made for the colourblind people and to not give people migraines and irritation to the eyes with bright colours. Though I will admit I took little consideration into this. Also using clear descriptive titles for all links and buttons for accessibility to the user. Also keeping a consistent layout throughout the pages will make sure that the user can navigate the site efficiently and quickly to not waste their time. Images and text must not be placed in awkward positions and should be clear. It is also important to make sure that all links and buttons work as intended and all text is spelled correctly with no errors

**Aesthetics –**

Aesthetics refers to how the website looks visually. Things to consider when making a website aesthetically pleasing are things like colour scheme, imagery, consistency, and unique design. In the context of my site I think at the moment this is my biggest issue. I am not a very artistic person and my html and css skills need some working on. So, in the future I would love to make my application look nicer. For now it works and that is good enough.

**Sustainability and Future Proofing –**

Making sure my website is framed in a way so that improvements and sustainability is possible is crucial. As eventually any website can become outdated. To do this, people reading the code for the website would need comments to explain what’s going on and how it works. Making sure compatibility with the website and browsers stay consistent is also important, so keeping updated on that is good. Overall, it is important that the website is framed in a way that future developers can make improvements easily and quickly to assure the future of the site is good.

# Evaluation:

I am very pleased with the outcome of my project. I have successfully created an application that not only I can use, but also my friends and people interested in health and fitness. Although there are plenty of applications out there that have all my features and more, my project is free! So, once I secure a HTTPS with an emphasis on the S and host it using some server agency I can share my project to people around the world. In terms of meeting the specifications:

User authentication wise works brilliantly, I can have multiple different users all storing and using the website at the same time no problem, with all their unique data. Signing up a new user is straightforward and easy to do. So I am happy with this aspect.

In terms of the meal logging aspect:

It could be better; I will not lie. In the future I would want to be able to have way more specific meal description abilities. Like being able to search for a specific food like eggs, beef, chicken etc and then setting an amount like 200g or something and have the code search up online using web scraping or a massive database to find the calories and protein per gram of that food, this way the user can streamline and personalize their macro tracking even better. But for now this simplistic approach is good if users have a good idea of how much protein was in the meal they just ate or can search it up.

A screenshot of a computer

Description automatically generatedIn terms of Goal setting, which is the whole premise of the site, It is again simplistic but affective. The only thing that I can think of improving would be to maybe add a feature that takes into account the BMI index and whether the user is cutting or bulking. Than give them a recommendation based on their specific information what calories and protein they need to be eating daily. But if the user knows themselves this is not needed.

A white rectangular object with red and green dots

Description automatically generatedIn terms of progress monitoring and Data visualization. I am quite proud with what I have accomplished. The user can see through out time how well they have been doing with their goals and how much protein / calories that they have been eating.

A massive shoutout to dash and plotly for making this process much easier than it would have been to create my own graphing software.

In the end, My project though it has some rough edges still works. Nothing is perfect and there are so many optimizations and future improvements I have in mind. But I hope you have enjoyed reading my report and understanding what it takes to develop a flask site. Thank you 😊

# GitHub repository:

<https://github.com/ericvg26/DTS-program-App/blob/main/README.md>