

**Daily Nutrition Tracker
Using Tkinter GUI
and Food Database API**

Final Project Report



Compiled by:

Nicholas Bryan (2802523042)
L1BC

**Algorithm and Programming
BINUS University International
Jakarta
2024**

Daily Nutrition Tracker Using Tkinter GUI and Food Database API

Nicholas Bryan
L1BC
BINUS University International

Abstract

This report represents a Python-based project developed to measure the ability to learn programming to one's own ability. The objective of this project is a program that solves a problem. To achieve this goal, it is decided that health, especially nutrition, is the focus. Tools used in this project are the Python programming language, Custom Tkinter GUI library, SQLite3 module, Datetime module, and the Edamam food database free API. The product of this project is an application that has the capability to track user information, fetch data from an external database, have an interactive and responsive display, and have error-preventing features.

Keywords: Python, Edamam API, nutrition tracker

Table of Contents

Abstract.....	i
Table of Contents.....	ii
Project Specification.....	1
A. Introduction.....	1
B. Inspiration.....	2
Solution Design.....	3
A. Edamam API Setup.....	3
B. User Nutrition.....	4
C. Sign Up System.....	4
D. Database.....	6
E. Main Page.....	8
Implementation Process.....	13
A. Essential Algorithms.....	13
B. Use Case Diagram.....	14
C. Activity Diagram.....	15
D. Class Diagram.....	18
E. Limitations.....	19
F. Modules Used.....	19
Program Functionality.....	20
A. GitHub Repository.....	20
B. Video Demo.....	23
Closing.....	24
A. Lesson Learned.....	24
B. Future Improvements.....	24
References.....	26

Project Specification

A. Introduction

The Python programming language is a high-level programming language that is popular and widely used. This project assesses the readiness to continue studying the Python programming language on an individual basis, showcasing program-designing skills, proper implementation, mastery of algorithms, programming concepts, and problem-solving skills that are emphasized upon learning new concepts.

Needing to tackle a small, real problem, the main focus of this project is on human health. A significant part of health is the ingestion of essential nutrients on a daily basis. Excessive consumption of sodium, sugar-sweetened products, and trans fats is directly linked with cardiometabolic diseases (National Institutes of Health, 2017). Therefore, paying attention to food is important to maintain well-being. In order to avoid generalization, this project must include the various aspects that affect health, such as age, gender, and medical conditions. Additionally, eating habits are significantly influenced by goals related to appearance and body, which will also be established.

The project involves a Python-based application performed locally. The functionality includes:

1. User authentication: account creation, login, and management of user data.

2. User interface: interactive interface for user inputs and feedback.
3. API integration: utilization of the Edamam API for nutritional information retrieval.
4. Data processing: analyzation and display of nutritional data in an intuitive format.

B. Inspiration

The idea of this project stems from my new way of eating in Jakarta. Nutrition is crucial to staying fit and making progress when living alone and attending the gym regularly. By signing up to a catering program, I can meet my protein and fiber targets without spending too much time preparing my own meals.

An interesting take for this project starts when I notice that caterers focus on numerous nutrients in the meal. Upon finding out the calculation process, I discovered that it is challenging. I could conclude that weighing every part of your food is time-consuming, let alone counting each of the nutrients included. Seeing that this is an interesting path for a project, I decided to make a product out of this idea.

Solution Design

A. Edamam API Setup

The data fetching used for this project is the *Nutrition Analysis API*. This API, on its own, has the ability to use Natural Language Processing to match an inputted food name and its amount. A response class named *API_edamame* is built for fetching 13 different data, which are:

1. Calories
2. Carbohydrates
3. Cholesterol
4. Fat
5. Protein
6. Sodium
7. Calcium
8. Iron
9. Potassium
10. Vitamin B-12
11. Vitamin B6A
12. Vitamin C
13. Vitamin D

There are 2 functions inside this class, *parameters()* to make a response from the API and *nutritional_data()* to return the 13 nutrient data of a food

presented inside a dictionary. To get the data from *nutritional_data()*, the user would input a food and amount, in the form of a string, as an argument. To access the correct numerical data from the JSON response, a short, repetitive *.get()* function code is called several times. Making sure that the Edamam API is working is a crucial first step to determine the workings of the project's development.

B. User Nutrition

The next step is to build a function called *nutrition()*. The purpose of this function is to find out a user's daily nutritional needs, which are the same 13 data returned from the Edamam API. This function accepts 8 arguments: sex, weight, height, age, activity level, fitness goal, carbohydrates strategy, and cardiac condition. The sex, weight, height, and age are used to count the Basal Metabolic Rate (BMR). The BMR is then going to be adjusted to count the carbohydrates intake and fat limit based on the activity level and carbohydrates strategy the user prefers. Cholesterol is based on whether the user has any cardiac problems. The rest are modified based on age and sex. The function contains a series of if-else clauses and dictionary key matching to produce the appropriate output.

C. Sign Up System

This step sees the first use of CustomTkinter GUI. A class called *UserInputSignup* is going to house all the components related to making a new account information. The first page is built using *CTK.CTk()* class

instance call, with a *.title()* function to give a header text on the window, and *.geometry()* function for the size of the window. This page is asking 4 questions:

1. Sex: a radio button input, male or female, using *CTK.CTkRadioButton()*.
2. Weight (kg): an integer/float in an entry input, using *CTK.CTkEntry()*.
3. Height (cm): an integer/float in an entry input, using *CTK.CTkEntry()*.
4. Age: an integer in an entry input, using *CTK.CTkEntry()*.

Nevertheless, all numerical inputs in the first page are going to be converted into an integer. A “Next” button is located at the bottom of the page to proceed. Then, the second page will appear, closing the page before using the *.destroy()* function. This page is similarly built as the first page. This page asks 4 more questions:

5. Activity level: a radio button input, sedentary/lightly active/moderately active/very active/extremely active.
6. Fitness goal: a radio button input, maintenance/bulking/cutting/weight-loss.
7. Carbohydrates intake strategy: a radio button input, maintenance/surplus/deficit/keto.
8. Cardiac problems: a radio button input, yes/no.

Another “Next” button is going to direct the user to a confirmation window. If they would like, they can return to the first page to change the existing data by clicking the “Back” button. Other than that, a “Next”

button is going to save all user information and an application restart is necessary. The return of this class is a dictionary containing the 8 user inputs. Clicking the next button in all these pages includes all user input validation and error prevention: the integer input denies any string inputs, and all radio buttons must be selected before proceeding. The user may not continue if they don't abide by these requirements.

D. Database

The database connection will be formed automatically when the application is run. There are 4 primary tables in the database, which are *users*, *user_information*, *user_information_fulfilled*, and *user_log*. The *users* table saves the ID, username, and password of all existing users. The *user_information* table saves the ID and 8 inputs presented in a dictionary from the Sign Up System. The *user_information_fulfilled* table is going to save the ID and 13 distinct daily nutrients each user has fulfilled. Lastly, *user_log* is a single-row table that saves the date of the active application. The *user_information_fulfilled* table is linked to the *user_log* table. Whenever the date saved in *user_log* is different from the current date, all the 13 nutrients in the *user_information_fulfilled* will be reset to 0. Then, the *user_log* updates itself to the correct current date. Each of these tables are created using the "CREATE TABLE IF NOT EXISTS" SQL line to not produce duplicate tables. Noticing the ID column is present in every table, it is referenced from the *users* table using the "FOREIGN KEY (id)

REFERENCES users (id)” line. The ID marks a single user, becoming a binding tool and navigation between tables.

There are 6 other callable functions in this class that are going to work in unison with the main application class:

1. *get_user()*: related to the login system of the app. Returns an ID matching an existing username and password. If there are no valid credentials, it returns *None*. Accesses the *user* table in the database.
2. *get_user_information()*: related to the login system of the app. Returns the user's personal condition and preferences (sex, weight, height, age, activity level, fitness goal, carbohydrates intake strategy, and cardiac condition) based on their ID. Accesses the *user_information* table in the database.
3. *get_user_information_fulfilled()*: related to the login system of the app. Returns the 13 distinct nutritional daily targets of a user based on their ID. Accesses the *user_information_fulfilled* table in the database.
4. *add_user()*: related to the signup system of the app. Checks whether the new username is unique by comparing it to the entire database. If it is unique, the username and the password is saved. This function returns the ID of the new user account created. If a duplicate is found, the function returns *None* as an indicator of failure. Accesses the *users* table in the database.
5. *add_user_information()*: related to the signup system of the app. After the *add_user()* function, the user would be asked to fill out the series

of 8 questions presented in the Sign Up System discussed before. The function arguments needed to execute *add_user_information()* is the user ID memorized in the *add_user()* function's return and the 8 new inputs regarding user's conditions and preferences. This function accesses both the *user_information* table and *user_information_fulfilled* table. The values stored in *user_information_fulfilled*, other than the ID, are 0 (default starting value).

6. *update_user_information_fulfilled()*: related to the Edamam API. Inside the application, users may enter a food name and amount. The API would fetch the nutritional values of these foods and *update_user_information_fulfilled()* function would take the existing data and add it up with the new data given by the Edamam API. Accesses the *user_information_fulfilled* table in the database.

E. Main Page

The last class is called *Application*, which holds the entire operation of the running app. The running app works by sequentially calling functions and methods. The *__init__()* function for this class is:

1. *self.db_manager = db_manager*, where *db_manager* is the initialized instance of the *Database* class. It is channelled through the argument of the *Application* class.
2. *self.gettingdata = UserInputSignup()*, which is the class instance call from the Sign Up System.

3. *self.fooddatafromAPI = API_edamame()*, which is the class instance call from the API module.
4. The model of the CustomTkinter window using *CTK.set_appearance_mode()* and *CTK.set_default_color_theme()*.
5. Calling the first function, which is *self.create_entry_window()*.

create_entry_window() function is the first window in the application. It displays two buttons, asking whether the user would like to log in or sign up. The “Login” button has an internal command to execute *self.create_login_page*, while the “Sign Up” button has an internal command to execute *self.create_signup_page*. Assuming that the user doesn’t have an account, the intuitive next step is to click the “Sign Up” button.

create_signup_page() function closes the *create_entry_window()* window and creates a new one displaying username entry inputs, password entry inputs, and a “Submit” button. The “Submit” button has an internal command to execute *self.signup*.

signup() function is a buffer between the application and the database used to check the validity of user inputs. *self.signup()* does not close the *create_signup_page()* window immediately. The username and password entries are retrieved, and then the lengths of the inputs are checked. If one of the inputs is empty, a label would appear to guide the user to enter a valid credential. If not, then *self.db_manager.add_user(username, password)* would be called. If there

exists a valid ID, subsequently the *create_signup_window()* would close, and *self.gettingdata.create_first_window()* would be called, which is the first function of the *UserInputSignup()* class. Finishing the chain of questions, *self.db_manage.add_user_information()* will be called. To funnel the inputs received from *UserInputSignup()*, the dictionary values are called as such: *self.gettingdata.user_data[name_of_key]*. However, if there is no existing valid ID, another label would be shown, informing the user that the username they are using has been taken. This function ends the signup sequence, and the app is restarted.

create_login_page() function is called when the “Login” button in *create_entry_window()* is clicked. The window will display username entry inputs, password entry inputs, and a “Submit” button. The “Submit” button has an internal command to execute *self.login*.

login() function, similar to *signup()* function, is a buffer and does not close the window immediately. The username and password entries are retrieved, and then *self.db_manager.get_user(username, password)* is called. If there exists a matching username-password combo, the ID of it is going to be used to fetch data from the databases using *self.db_manager.get_user_information()* and *self.db_manager.get_user_information_fulfilled()*. The return values of these functions are saved in a variable and passed through as arguments into *self.show_main_page()*. Oppositely, if a username-password combo doesn't exist, the user would be informed of the error.

show_main_page() function displays 4 elements: a table, an entry input, and 2 buttons. The table is divided into 4 columns, which are “Information,” “Fulfilled,” “Daily Needs,” and “Measurements.” There are a total of 13 different rows, representing the name, the amount fulfilled, the daily amount, and the unit of measurement used for every single nutrient. The daily amount is retrieved by processing the data through the steps described in User Nutrition. To add a food into the database, the entry input and “Find my food!” button is used. “Find my food!” button has an internal command to execute *self.fetch_data_from_api*. The second button is an “Exit” button to close the application.

fetch_data_from_api() function is going to retrieve the food and amount description from the entry. This input allows multiple inputs at a time by splitting the string. Each item is checked for any empty strings, and then the *self.fooddatafromAPI.nutrition_data(item)* will be called. The 13 nutrient informations are totaled and sent to the *self.add_nutrition_data_to_database()* function. If there is an item that is unregistered, or the data is lost, the value of that item is converted to the default (0.0 or 1.0, depending on where the error happens).

add_nutrition_data_to_database() function fetches the current nutrient-fulfilled amount and adds it together with the new inputs from *fetch_data_from_api()*. The new total is then used to be the updated value of the *user_information_fulfilled* database by calling

self.db_manager.update_user_information_fulfilled(). Finishing this part, the user may return to the main page once more to examine the new *user_information_fulfilled* display.

Implementation Processes

A. Essential Algorithms

There are 4 crucial algorithms utilized in the nutrition tracker app's source code, which are user authentication algorithm, SQL query execution algorithm, API request handling, and dictionary verification algorithm.

1. **User Authentication:** Users using the app are given the ability to sign up and login, allowing accurate data management. Based on success or failure, the GUI helps in directing users accordingly.
2. **SQL Query:** The database has the ability to insert, select, and update values in tables. Insertions are applied in the adding of new users and their answers during signup. Selections are applied in retrieving existing data during login based on valid IDs. Lastly, updates are applied in the modification of values in the *user_information_fulfilled* database.
3. **API Request Handling:** The Edamam API setup has the ability to send requests and handle multiple items by creating separate API calls. Retrieval of nutritional data for each food item is specific to reduce load.
4. **Dictionary Verification:** To ensure proper and effective usage of variables, most of the function's return values are in the form of dictionaries. Dictionaries are useful to verify input and output integrity.

Inspecting the source code further, the time complexity for multiple API calls is $O(n^2)$ or quadratic time complexity. In *fetch_data_from_api()*, there are nested for-loops used to iterate through multiple food inputs and multiple access to a dictionary.

B. Use Case Diagram

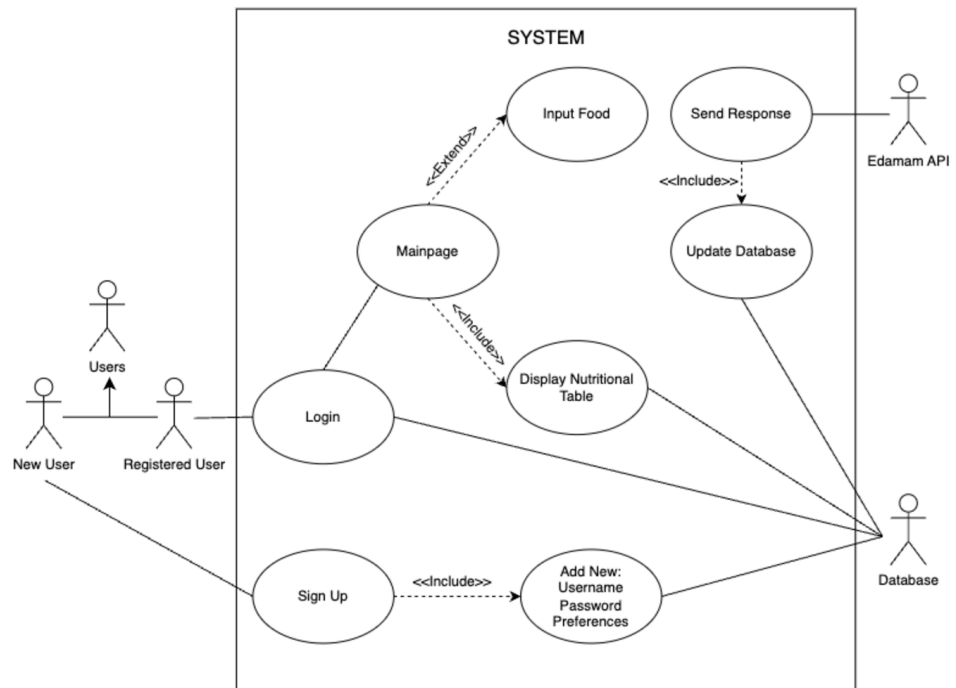


Figure 1. Use Case Diagram

C. Activity Diagram

1. Entry Into Application

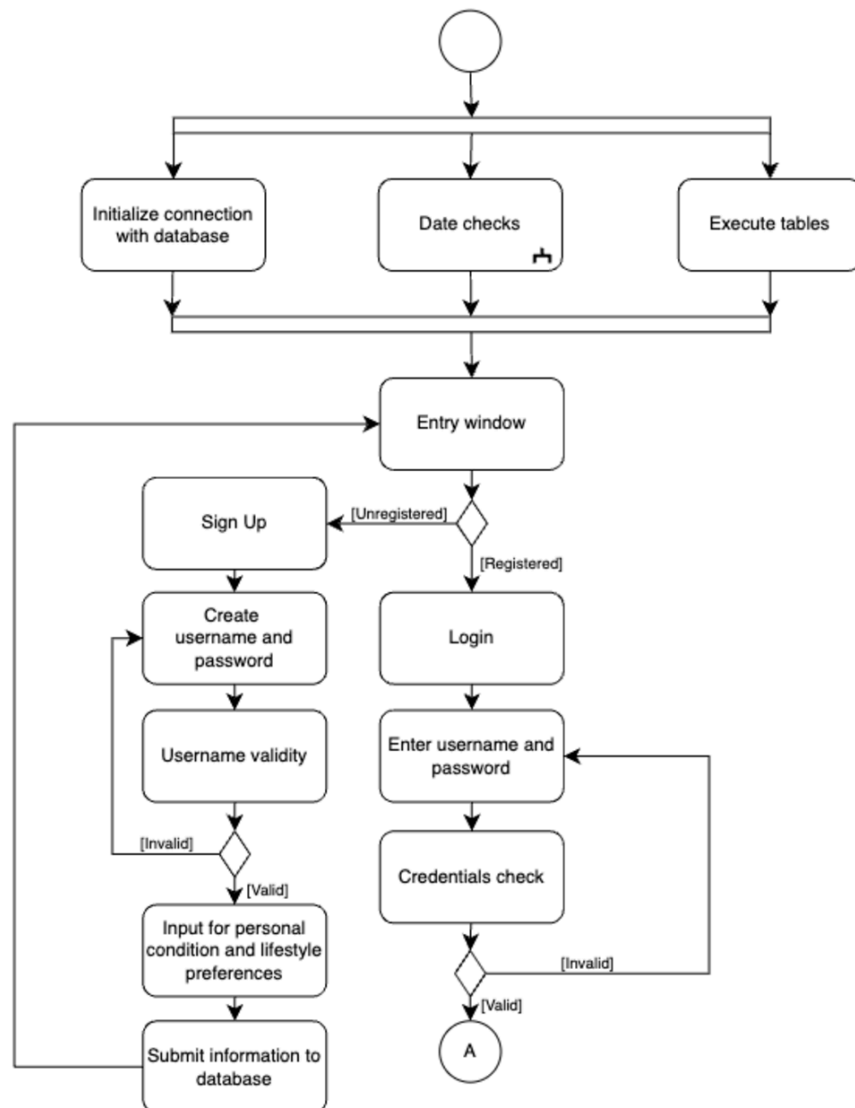


Figure 2. Activity Diagram Signup Login

2. Inside Application



Figure 3. Activity Diagram Inside Application

Due to the nature of the program, the terminating symbol in the activity diagram explains the exiting of the program. The user may let the program run without it quitting.

3. Date Check

To minimize the complexity of the activity diagram, a sub-activity is referenced for the date checking system of the application.

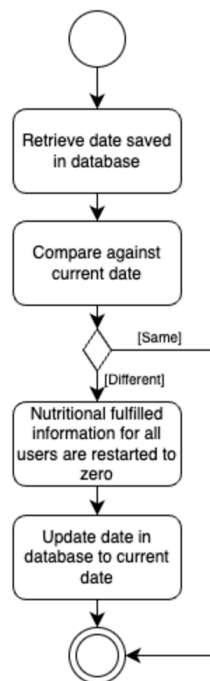


Figure 4. Activity Diagram for Date Checks

D. Class Diagram

This class diagram displays the arrangements and relationships between classes. Any functions or variables corresponding to the *CustomTkinter* module will not be shown, for it doesn't contribute to the broad understanding of the classes.

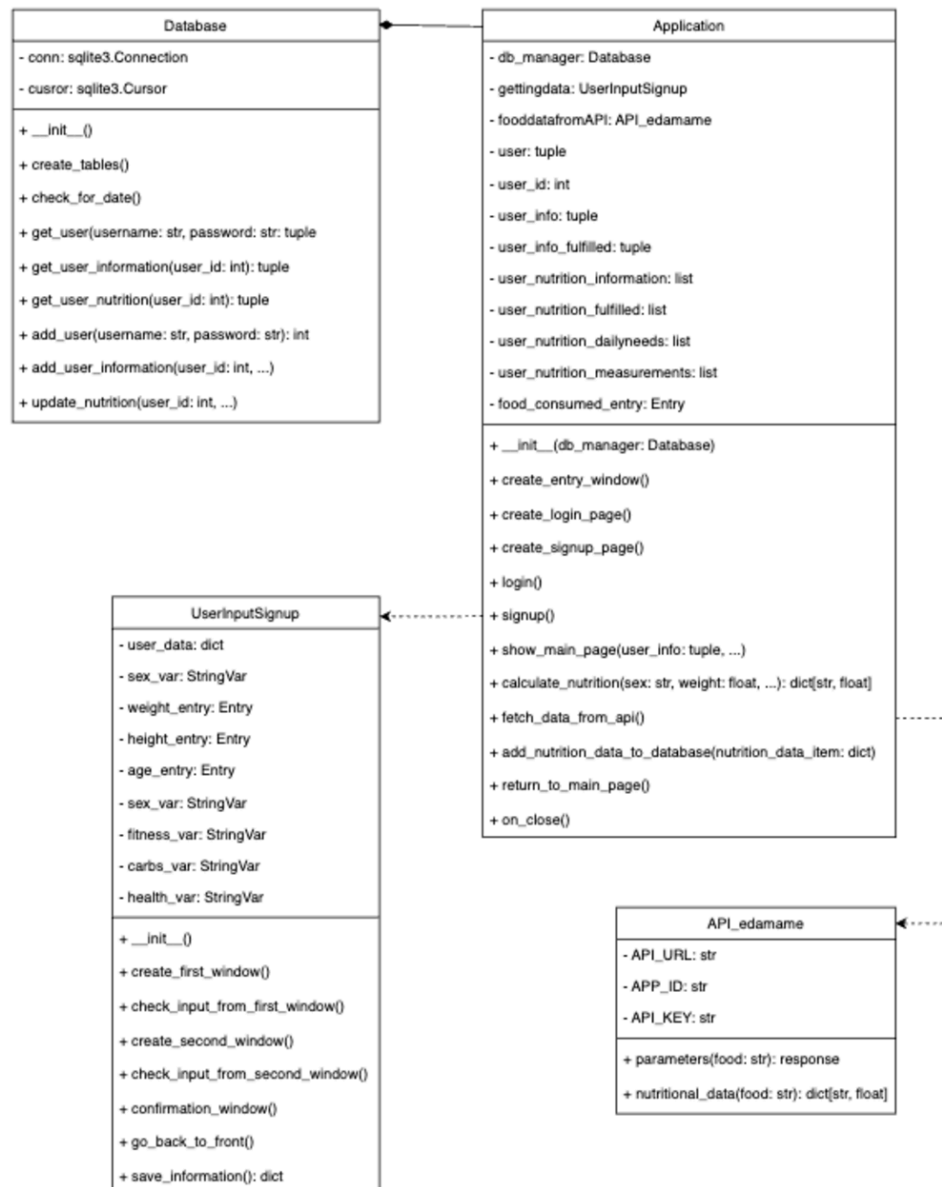


Figure 5. Class Diagram

E. Limitations

Unfortunately, the SQLite3 database used in this project works locally on a single computer, which implies all the operations cannot be accessed through the internet. This limits the usability of the application for multi-user, online usage.

F. Modules Used

1. Requests: a Python library that enables making HTTP requests, useful for interacting with web APIs or doing web scraping. The library offers an interface which simplifies the process of various HTTP methods (GET, POST, PUT, DELETE) such as retrieving data from a server or creating customized responses in response to user actions.
2. SQLite3: a C library that provides a lightweight disk-based database that doesn't require a separate server process. It uses a nonstandard variant of the SQL query language.
3. CustomTkinter: a Python desktop UI-library based on Tkinter, providing modern aesthetics and customizable widgets. CustomTkinter has a consistent look across all desktop platforms.
4. Tkinter: a Python library used to construct basic GUI applications. Tkinter allows the addition of widgets, which work similarly to HTML elements.
5. Datetime: a module supplying classes to work with date, time, and time intervals.

Program Functionality

A. GitHub Repository

<https://github.com/NichBry25/Nutrition-Tracker>

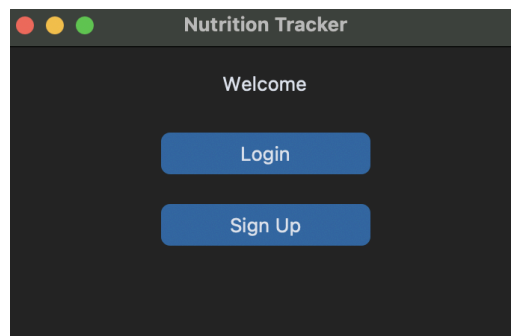


Figure 6. Login or Sign Up Page

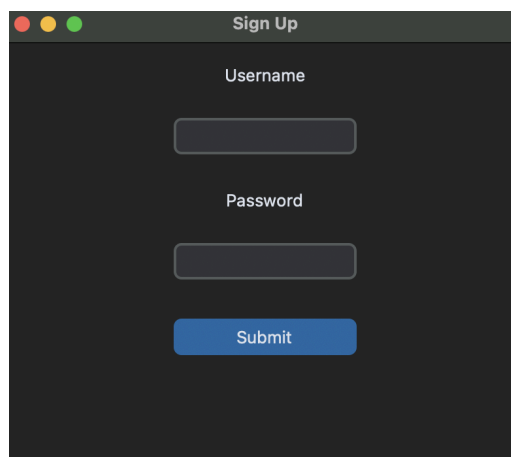
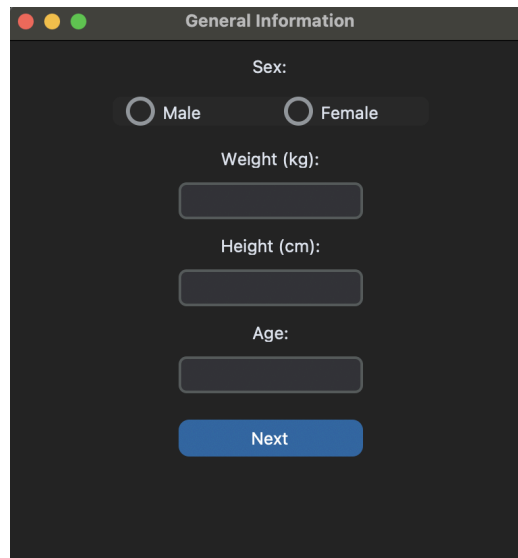
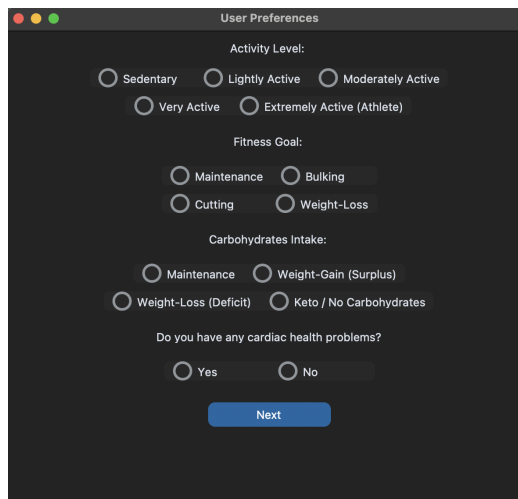


Figure 7. Sign Up Page



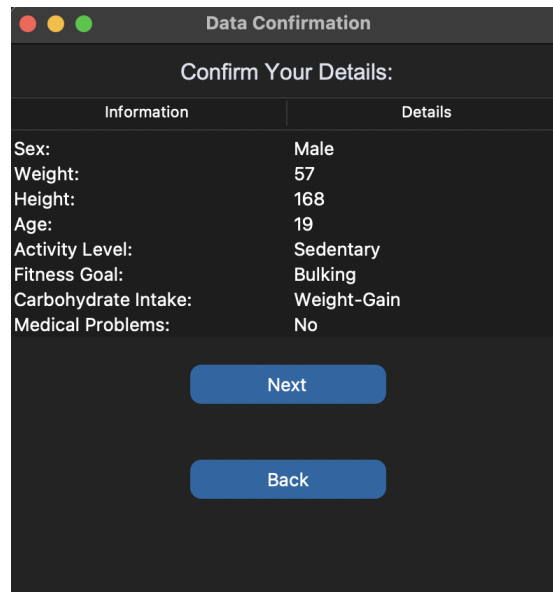
A dark-themed window titled "General Information" with standard macOS window controls (red, yellow, green buttons) in the top-left corner. The form contains the following elements: a "Sex:" label followed by two radio buttons labeled "Male" and "Female"; a "Weight (kg):" label followed by a rectangular text input field; a "Height (cm):" label followed by a rectangular text input field; an "Age:" label followed by a rectangular text input field; and a blue "Next" button at the bottom.

Figure 8. General Information Page



A dark-themed window titled "User Preferences" with standard macOS window controls (red, yellow, green buttons) in the top-left corner. The form contains the following elements: an "Activity Level:" label followed by five radio buttons labeled "Sedentary", "Lightly Active", "Moderately Active", "Very Active", and "Extremely Active (Athlete)"; a "Fitness Goal:" label followed by four radio buttons labeled "Maintenance", "Bulking", "Cutting", and "Weight-Loss"; a "Carbohydrates Intake:" label followed by four radio buttons labeled "Maintenance", "Weight-Gain (Surplus)", "Weight-Loss (Deficit)", and "Keto / No Carbohydrates"; a "Do you have any cardiac health problems?" label followed by two radio buttons labeled "Yes" and "No"; and a blue "Next" button at the bottom.

Figure 9. User Preferences Page

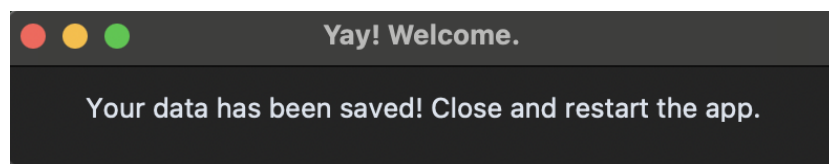


A macOS-style window titled "Data Confirmation" with a dark gray background. It features a table with two columns: "Information" and "Details". The table contains the following data:

Information	Details
Sex:	Male
Weight:	57
Height:	168
Age:	19
Activity Level:	Sedentary
Fitness Goal:	Bulking
Carbohydrate Intake:	Weight-Gain
Medical Problems:	No

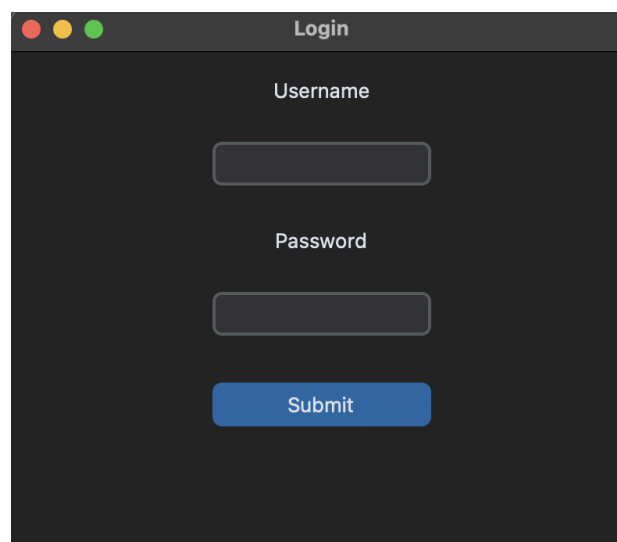
Below the table are two blue buttons: "Next" and "Back".

Figure 10. Data Confirmation Page



A macOS-style window titled "Yay! Welcome." with a dark gray background. It contains a single line of text: "Your data has been saved! Close and restart the app."

Figure 11. Sign Up Successful Popup



A macOS-style window titled "Login" with a dark gray background. It contains two text input fields: "Username" and "Password". Below the "Password" field is a blue button labeled "Submit".

Figure 12. Login Page

Information	Fulfilled	Daily Needs	Measurements
Calories Intake:	0	2589.65625	kcal
Carbohydrates Intake:	0	1424.3109375000001	g
Cholesterol Limit:	0	300	mg
Fat Limit:	0	647.4140625	g
Protein Intake:	0	104.5	g
Sodium Limit:	0	2000	mg
Calcium Intake:	0	1300	mg
Iron Intake:	0	11	mg
Potassium Intake:	0	4700	mg
Vit B12 Intake:	0	2.4	microgram
Vit B6A Intake:	0	1.3	mg
Vit C Intake:	0	75	mg
Vit D Intake:	0	15	microgram

Find and add my food!

Exit

Figure 13. Main Page

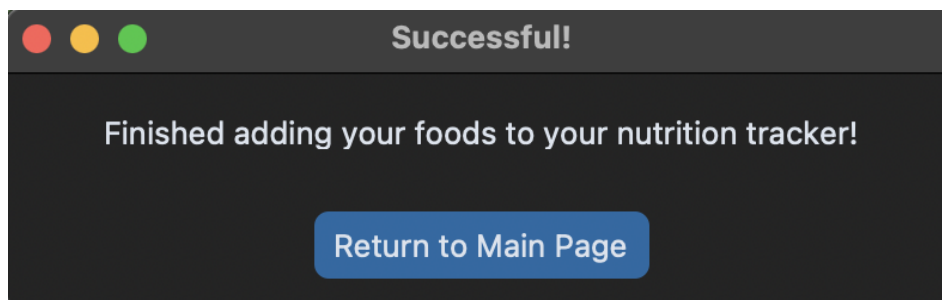


Figure 14. Food Added Popup

B. Video Demo

<https://youtu.be/PWHwsy95dOg?si=yANJsVg1jTa2XR1q>

Closing

A. Lessons Learned

From this project, I learned that Python is an easily understood programming language. However, Python has its own setbacks, such as a slower program execution. This teaches me to be smarter on how to implement certain algorithms to work in a timely manner. Nonetheless, Python is great because of the community. The community works together to build and create modules and libraries that enable an efficient environment.

In terms of preparation to continue programming alone, this project teaches me to be resilient. Any logical and substantial ideas could be brought to life as long as there exists the willingness to learn more. This project encourages me to do more outside what I am instructed, pushing the limits of my ability to create more projects that are useful.

B. Future Improvements

After finishing this project, I realized there are a couple features that I could have added. The simplest addition that comes to mind is to give the users the ability to update their information basis. Users using the application are expected to be more healthy, which entails a change in weight, personal preference of fitness, and such.

A possible improvement for usability is not using the SQLite3 database and instead opting for a server-based database. However, doing

this in the time given for the project, as a beginner in Python, would be possibly too complicated and time-consuming.

References

- National Institutes of Health (NIH). (n.d.). *How dietary factors influence disease risk*. Retrieved December 16, 2024, from <https://www.nih.gov/news-events/nih-research-matters/how-dietary-factors-influence-disease-risk>
- Python Software Foundation. (n.d.). *sqlite3 — DB-API 2.0 interface for SQLite databases*. Retrieved December 16, 2024, from <https://docs.python.org/3/library/sqlite3.html>
- Schimansky, T. (n.d.). *CustomTkinter documentation*. Retrieved December 16, 2024, from <https://customtkinter.tomschimansky.com/>
- Simplilearn. (n.d.). *Python graphical user interface (GUI) tutorial*. Retrieved December 16, 2024, from <https://www.simplilearn.com/tutorials/python-tutorial/python-graphical-user-interface-gui#:~:text=Tkinter%20is%20a%20Python%20library,let's%20get%20started%20with%20Tkinter>
- Simplilearn. (n.d.). *Python requests: A detailed tutorial*. Retrieved December 16, 2024, from <https://www.simplilearn.com/tutorials/python-tutorial/python-requests#:~:text=Python%20requests%20is%20a%20library,both%20GET%20and%20POST%20methods>