Nutrition Tracker and Health Monitoring System - MedTrack

Diez, Angela May R.
Resuma, Wayne Jairoh S.
Reyes, Eli Justin L.
Salgado, Andee Ramon V.
Subalisid, Jappeth Royce

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

Many people experience health issues due to unawareness of early signs of illness or inconsistent health monitoring. This lack of awareness and consistent tracking makes it difficult to spot health issues early. These noncommunicable diseases (NCDs) – such as heart disease, cancer, and diabetes are responsible for 74% of all deaths worldwide (World Health Organization, 2025).

Without a structured system for tracking nutrition and vital signs, users may overlook small but significant changes in their health data. When combined with data security concerns that discourage consistent monitoring (Iwaya et al., 2020), this gap in monitoring directly enables the manageable health issues to progress into serious conditions.

MedTrack is designed to be a simple and user-friendly tool that allows individuals to track their daily health and nutrition data. It serves as a personal health record system, enabling users to log vital information and dietary habits to monitor progress and detect unhealthy trends over time. The program also prioritizes data security, ensuring that all user information is protected through password-secured accounts.

The Project

MedTrack allows users to create an account in which they will be able to do daily logs of their vital signs. The vital signs that will be requested from the user upon the creation of an account are as follows: Name, age, height, weight, gender, activity level, and password for the account. Once the account has been created, a CSV file will be created to hold all that account's information. This is how the program will save and access data. According to Ismail et al. (2020), efficient health data management systems should ensure secure and structured storage of user information to support both personal health tracking and broader biomedical research purposes.

Now what can the user do with their account? The user can update their logs, or delete the account. When updating the logs of the user's account, the program will first request the user to input their password; if the password the user inputs into the program matches the first password inputted by the user upon their account creation, then they will be able to add a log entry into the account. Ragupathi et al. (2022) emphasized that interactive health monitoring systems should enable continuous user engagement through regular data updates, allowing individuals to actively participate in tracking and improving their health.

What will log entries look like? Each log entry will have the date, blood pressure, resting heart rate, blood oxygen saturation, and caloric intake for the day. Once the required fields are filled out, the program will append the data into the user account's file. This type of structured daily monitoring aligns with the growing adoption of smart and connected care systems that integrate diet and vital sign tracking to provide comprehensive health insights (Coman et al., 2024).

How will the user be able to delete their account? When selecting the option to delete their account, the user will be prompted to input their password. If the password is correct, then the program will delete the account's file. This process is irreversible, so the user will be asked if they are sure if they wish to permanently delete their account and all data held within their file. As highlighted by Ismail et al. (2020), maintaining user control and privacy over personal health data is an essential component of ethical and secure health management systems.

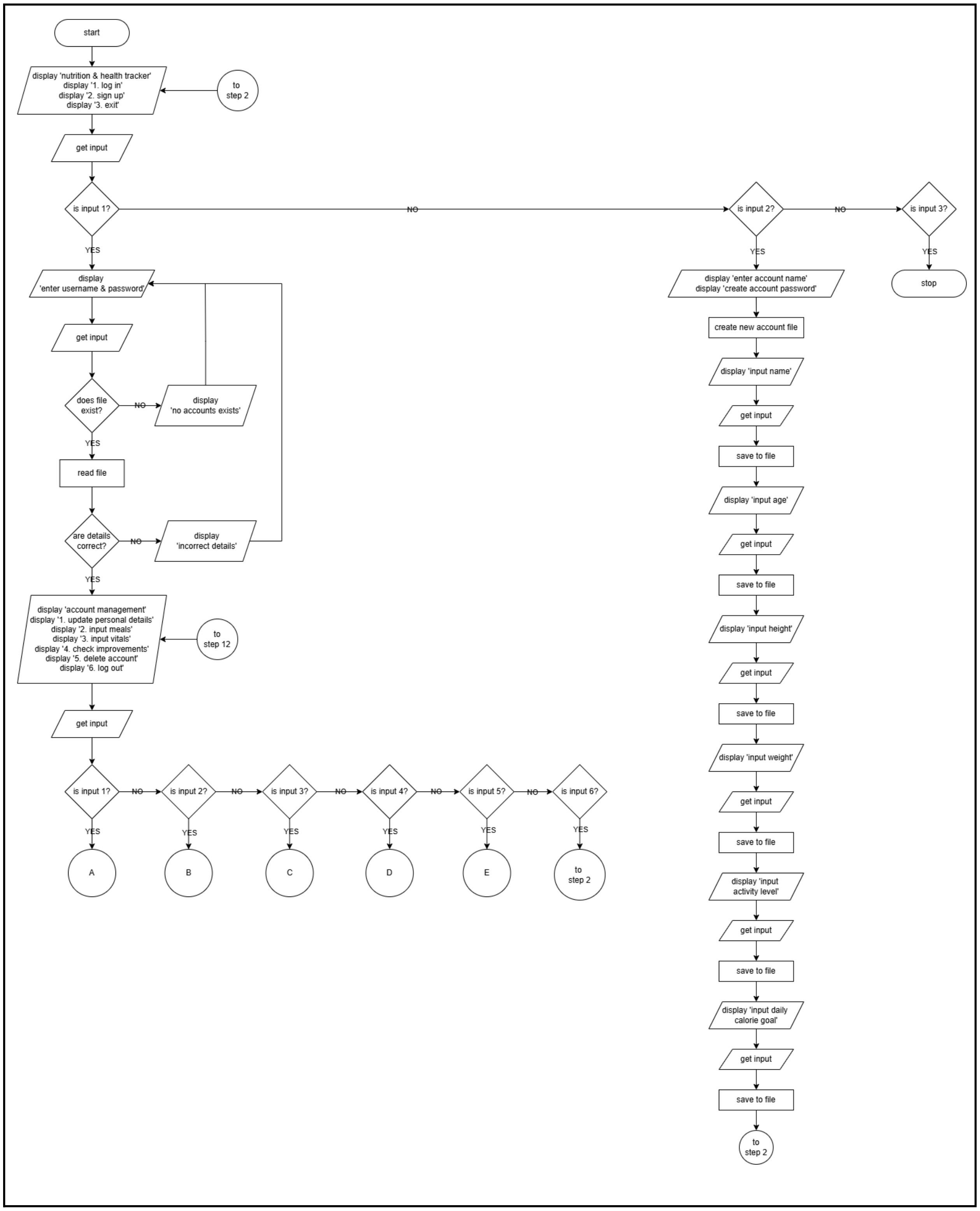
Objectives

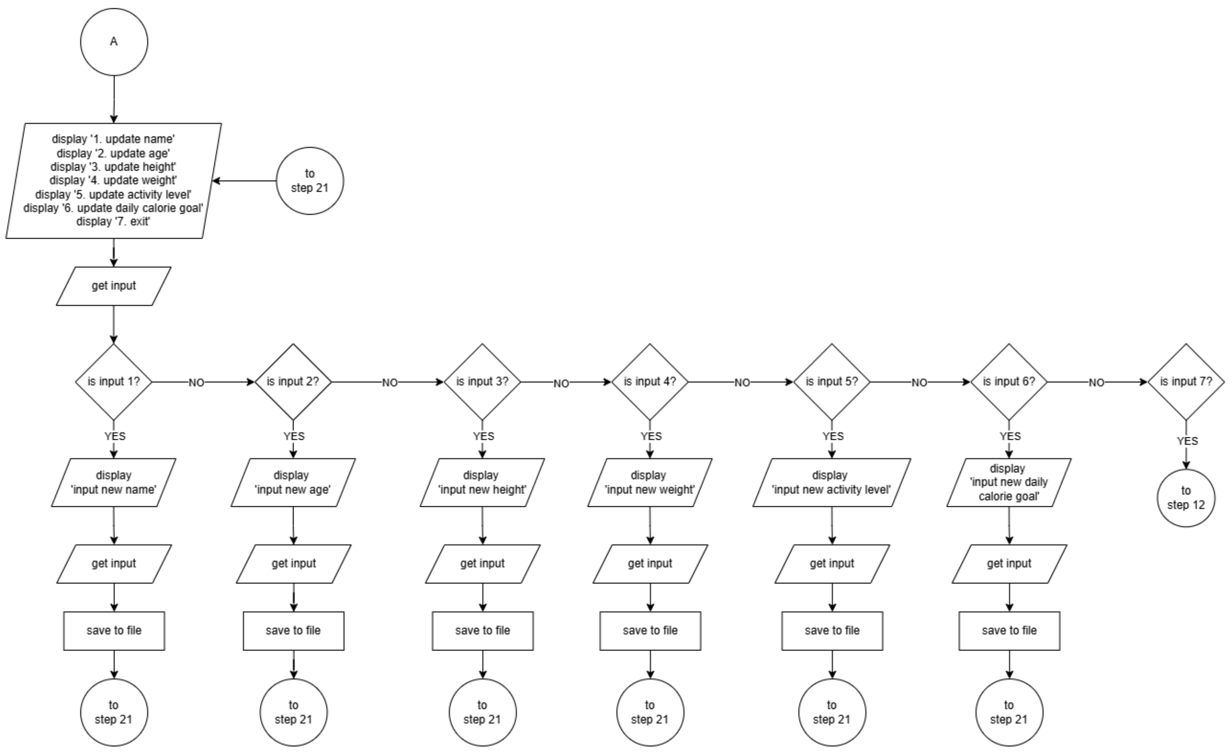
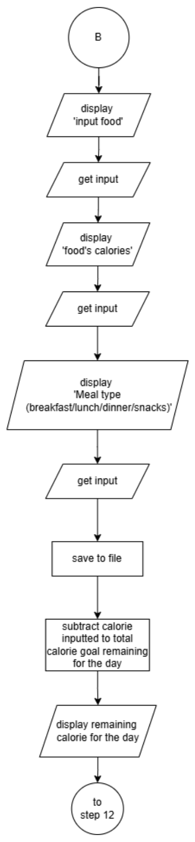
The objective of MedTrack is to track one's basic health metrics to possibly catch early signs of illness.

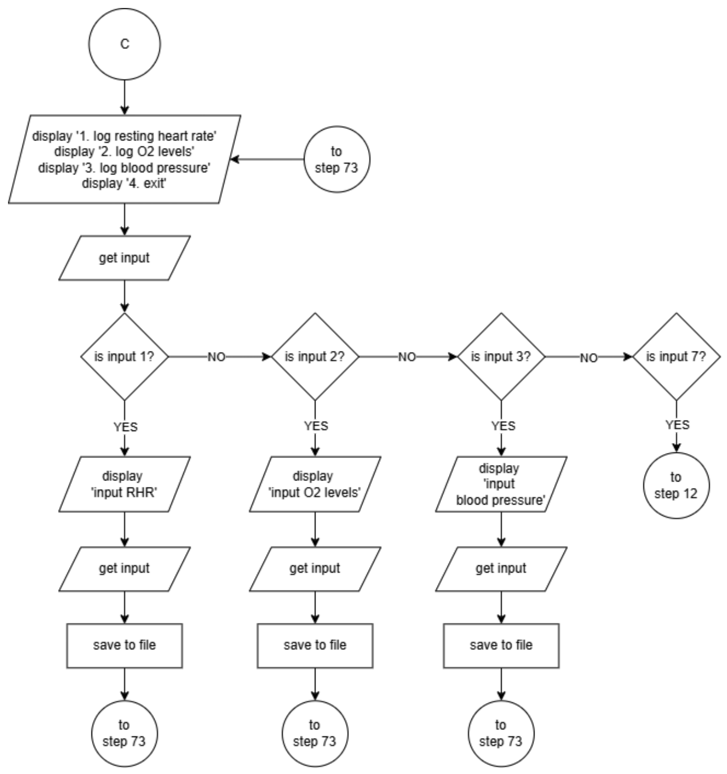
1. Implement an Admin system where users are able to create their personal accounts for tracking their daily calorie intake and vitals.
2. Allow users to input their vitals information and daily caloric intake.
3. Display the user's weekly average vitals, the change from the previous week's average, and their weekly average caloric intake.

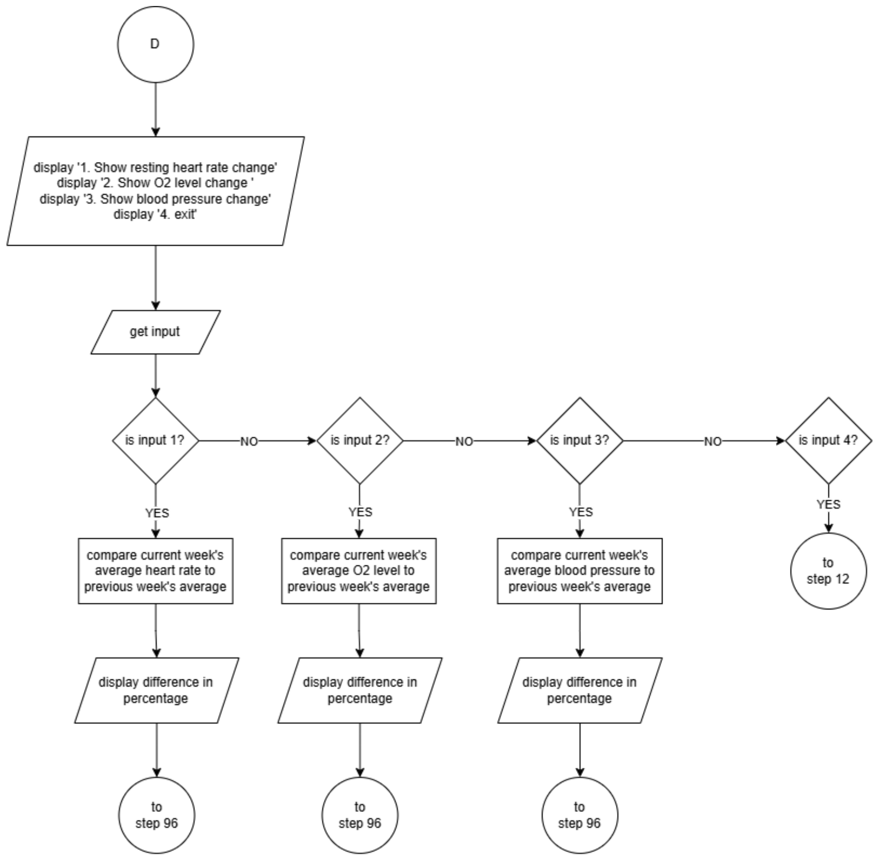
Flowchart of the System

This is the program's flowchart. This is a graphical representation of the program's code. You may use this as a visual aid to understanding how the program functions.


start
display 'nutrition & health tracker' display '1. log in' display '2. sign up' display '3. exit'
to step 2
get input
is input 1?
NO
is input 2?
NO
is input 3?
YES
stop
YES
display 'enter username & password'
get input
does file exist?
NO
display 'no accounts exists'
YES
read file
are details correct?
NO
display 'incorrect details'
YES
display 'account management' display '1. update personal details' display '2. input meals' display '3. input vitals' display '4. check improvements' display '5. delete account' display '6. log out'
to step 12
get input
is input 1?
is input 2?
is input 3?
is input 4?
is input 5?
is input 6?
A
B
C
D
E
to step 2
display 'enter account name' display 'create account password'
create new account file
display 'input name'
get input
save to file
display 'input age'
display 'input height'
display 'input weight'
display 'input activity level'
display 'input daily calorie goal'
to step 2


A
display '1. update name'
display '2. update age'
display '3. update height'
display '4. update weight'
display '5. update activity level'
display '6. update daily calorie goal'
display '7. exit'
to step 21
get input
is input 1?
NO
is input 2?
NO
is input 3?
NO
is input 4?
NO
is input 5?
NO
is input 6?
NO
is input 7?
YES
display 'input new name'
get input
save to file
to step 21
YES
display 'input new age'
get input
save to file
to step 21
YES
display 'input new height'
get input
save to file
to step 21
YES
display 'input new weight'
get input
save to file
to step 21
YES
display 'input new activity level'
get input
save to file
to step 21
YES
display 'input new daily calorie goal'
get input
save to file
to step 21
YES
to step 12

B
display 'input food'
get input
display 'food's calories'
get input
display 'Meal type (breakfast/lunch/dinner/snacks)'
get input
save to file
subtract calorie inputted to total calorie goal remaining for the day
display remaining calorie for the day
to step 12


C
display '1. log resting heart rate'
display '2. log O2 levels'
display '3. log blood pressure'
display '4. exit'
to step 73
get input
is input 1?
NO
is input 2?
NO
is input 3?
NO
is input 7?
YES
YES
YES
YES
display 'input RHR'
display 'input O2 levels'
display 'input blood pressure'
to step 12
get input
get input
get input
save to file
save to file
save to file
to step 73
to step 73
to step 73


D
display '1. Show resting heart rate change'
display '2. Show O2 level change '
display '3. Show blood pressure change'
display '4. exit'
get input
is input 1?
NO
is input 2?
NO
is input 3?
NO
is input 4?
YES
YES
YES
YES
compare current week's average heart rate to previous week's average
compare current week's average O2 level to previous week's average
compare current week's average blood pressure to previous week's average
to step 12
display difference in percentage
display difference in percentage
display difference in percentage
to step 96
to step 96
to step 96

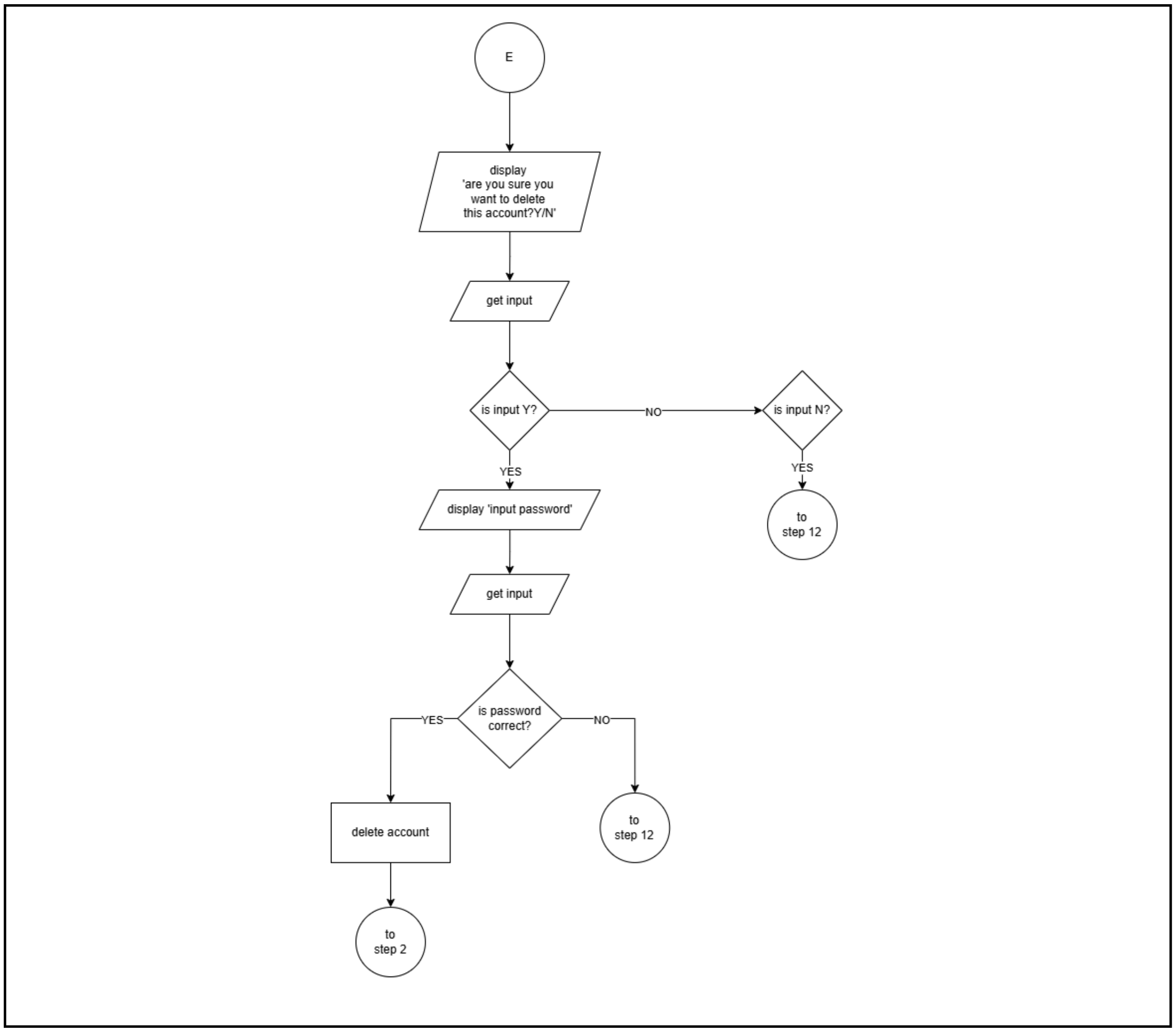


Figure 1: MedTrack system flowchart

Within Figure 1 is the program's flowchart. Listed below are the basic symbols and meaning of each symbol in the flowchart as a guide to further explain the program.

Arrow = Flow, indicates where the next step is
Oval = Terminator, indicates the start and end of the program.
Rectangle = Process, indicates an operation in the code.
Diamond = Decision, indicates a decision within the code.
Rhombus = Data, an input/output in the code.
Circle = Connector, connects 2 parts of the flowchart together.

Pseudocode

This section will display the pseudocode of our proposed project. You may use this to read through what the program does, even if you are not completely familiar with code, as this uses English to describe what the code does.

```
1.  start
2.      display 'nutrition & health tracker'
3.      display '1. log in'
4.      display '2. sign up'
5.      display '3. exit'
6.      get input
7.          if input is 1
8.              display 'enter username & password'
9.              get input
10.             check if file exist: read file if yes; display 'no accounts exists" if no
11.             verify account: continue if account details are correct; return to step 2 if
    account details are incorrect
12.             display 'account management'
13.             display '1. update personal details'
14.             display '2. input meals'
15.             display '3. input vitals'
16.             display '4. check improvements'
17.             display '5. delete account'
18.             display '6. log out'
19.             get input
20.                 if input is 1
21.                     display '1. update name'
22.                     display '2. update age'
23.                     display '3. update height'
24.                     display '4. update weight'
25.                     display '5. update activity level'
26.                     display '6. update daily calorie goal'
27.                     display '7. exit'
28.                     get input
29.                         if choice is 1
30.                             display 'input new name'
31.                             get input
32.                             save to file
33.                             return to step 21
34.                         if choice is 2
35.                             display 'input new age'
36.                             get input
```

```
37.                         save to file
38.                         return to step 21
39.                     if choice is 3
40.                         display 'input new height'
41.                         get input
42.                         save to file
43.                         return to step 21
44.                     if choice is 4
45.                         display 'input new weight'
46.                         get input
47.                         save to file
48.                         return to step 21
49.                     if choice is 5
50.                         display 'input new activity level'
51.                         get input
52.                         save to file
53.                         return to step 21
54.                     if choice is 6
55.                         display 'input new daily calorie goal'
56.                         get input
57.                         save to file
58.                         return to step 21
59.                     if choice is 7
60.                         return to step 12
61.             if input is 2
62.                 display 'input food'
63.                 get input
64.                 display 'food's calories'
65.                 get input
66.                 display 'Meal type (breakfast/lunch/dinner/snacks)'
67.                 get input
68.                 save inputs to file
69.                 subtract calorie inputted to total calorie goal remaining for the day
70.                 display remaining calorie for the day
71.                 return to step 12
72.             if input is 3
73.                 display '1. log resting heart rate'
74.                 display '2. log O2 levels'
75.                 display '3. log blood pressure'
76.                 display '4. exit'
77.                 get input
78.                     if choice is 1
```

```
79.                 display 'input rhr'
80.                 get input
81.                 save to file
82.                 return to step 73
83.             if choice is 2
84.                 display 'input O2 level'
85.                 get input
86.                 save to file
87.                 return to step 73
88.             if choice is 3
89.                 display 'input blood pressure'
90.                 get input
91.                 save to file
92.                 return to step 73
93.             if choice is 4
94.                 return to step 12
95.     if input is 4
96.         display '1. Show resting heart rate change'
97.         display '2. Show O2 level change '
98.         display '3. Show blood pressure change'
99.         display '4. exit'
100.        get input
101.            if choice is 1
102.                compare current week's average hr to previous week's average
103.                display difference in percentage
104.                return to step 96
105.            if choice is 2
106.                compare current week's average O2 level to previous week's average
107.                display difference in percentage
108.                return to step 96
109.            if choice is 3
110.                compare current week's average bp to previous week's average
111.                display difference in percentage
112.                return to step 96
113.            if choice is 4
114.                return to step 12
115.    if input is 5
116.        display 'are you sure you want to delete this account?Y/N'
117.        get input
118.            if input is Y
```

```
119.                            display 'input password'
120.                            get input
121.                            password is correct?: If yes, delete
     account file; if no, return to step 12
122.                        if input is N
123.                            return to step 12
124.                if input is 6
125.                    return to step 2
126.        if input is 2
127.            display 'enter account name'
128.            display 'create account password'
129.            create new account file
130.            display 'input name'
131.            get input
132.            save to file
133.            display 'input age'
134.            get input
135.            save to file
136.            display 'input height'
137.            get input
138.            save to file
139.            display 'input weight'
140.            get input
141.            save to file
142.            display 'input activity level'
143.            get input
144.            save to file
145.            display 'input daily calorie goal'
146.            get input
147.            save to file
148.            return to step 2
149.        if input is 3
150.    stop program
```

Figure 2: Pseudo code of MedTrack

As seen in Figure 2, from steps 2–5, is the initial display of the program's options. Step 5 is for the user's input. From steps 6–125 is for the "log-in" option, steps 126–148 for the "sign up" option, and steps 149 for the "exit" option.

When selecting "log in" the user is presented with steps 7–18, which shows the available options when logging in. step 19 then gets the user's input on what option is chosen.

Under the “update personal details” (steps 21–94) allow users to update their account’s details. Steps 21–27 shows the user 7 additional options. Under the option “Update name” are steps 29–33 which retrieves a new name from the user, then updates the file according to the new name. The same is done for options “update age” (steps 34–38), “update height” (steps 39–43), “update weight” (steps 44–48), “update activity level” (steps 49–53), and “update daily calorie goal” (steps 54–58), which all update their corresponding values within the user’s account. Option 7 “exit” (steps 93–94) returns the user to step 12.

Under the “input meals” option, steps 61–71. Prompts the user to input what meal they had, how many calories it had, what kind of meal it was, and saves the information to the user’s file.

Under the “Input vitals” option are steps 72–94. Steps 73–76 asks the user what option they will choose, and Step 77 gets the user input. Under “log resting heart rate” are steps 78–82, under “log O2 levels” are steps 83–87, and under “log blood pressure” are steps 88–92. All of these options retrieve the related data from the user and save them into the file. For the “exit” option (step 92–93), it returns the user to step 12.

Under “check improvements” are steps 95–114. Steps 96–100 displays the options under “check improvements” These options are “Show resting heart rate change” (steps 101–104), “Show O2 level change” (steps 105–108), and “Show blood pressure change” (steps 109–112). All of these options first take the average of their corresponding value’s average for that week and the previous week, compare the two values, and display the difference in percentage. The option “exit” (steps 113–114) returns the user to step 12.

Under the “delete account” option are steps 115–123. Steps 116–117 first confirms if the user wishes to delete their account, step 117 then retrieves the user’s input. If the user chooses to proceed, steps 118–121 asks the user for their account’s password. Step 121 checks if the passwords match, if they do, then it deletes the account’s file, if they do not match, it returns the user to step 12. If the user wishes to back out of the account deletion, they can choose “N” at step 117, if they do so, they will be returned to step 12 by steps 122–123.

Under “log out” (Steps 124–125) it simply returns the user to step 2.

Now for the “sign up” option, steps 126–148. When selecting this option at Step 6, steps 126–148 will be called. These steps go through the creation of the account file, and retrieving the basic information from the user that will be then saved to the file. Once it is all done, the user will be redirected to step 2 by step 148.

When selecting the “exit” option at step 6. Steps 149–150 will be called. Step 150 simply closes the program.

Data Dictionary

The data dictionary is a table which contains all the data that will be needed for the program to function. This allows us to see all the data types and what they are used for. Useful for referencing.

Within the table are the names of the variables, their memory size, data types that they will be assigned, and the description of that they are for and how they are used in the code.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description

1. Username	1 byte per character	String	This data is what's assigned as the file name and identifier of the account.
2. Password	1 byte per character	String	This data is used for confirmation of the identity of the user.
3. Name	1 byte per character	String	This is the name of the user.
4. Age	4 bytes	Integer	This is the age of the user, –which can be updated in case of change.
5. Height	8 bytes	Float	This is the height of the user in cm, which can be updated in case of change.
6. Weight	8 bytes	Float	This is the weight of the user in kg, which can be updated in case of change.
7. Activity level	1 byte per character	String	This is the activity level of the user, which can be updated in case of change.
8. Calorie Goal	4 bytes	Integer	This is the daily calorie goal of the user which will be used as a reference each day to display the remaining calories to be consumed. This can also be updated as the user desires.
9. Food	1 byte per character	String	This is the name of the food that the user consumed. It serves as a journal of what the user has been eating.
10. Calories	4 bytes	Integer	This is the caloric value of the food that the user consumed. This data will be required each time the user logs a food data.

11. Meal Type	1 byte per character	String	This is when the user ate the food he consumed. This can be either breakfast, lunch, dinner, or snack.
12. Resting Heart Rate	4 bytes	Integer	This is the resting heart rate of the user. Ideally measured daily as soon as the user wakes up. This is to show the cardiovascular improvements of the user.
13. O2 level	4 bytes	Integer	This is the oxygen saturation of the user. Ideally measured daily with the use of a pulse oximeter. This will be used to monitor the heart and lung's efficiency in delivering oxygen to your body.
14. Blood Pressure	4 bytes	Integer	This is the blood pressure of the user. Ideally measured daily as soon as the user wakes up. This will be used by the user to find early signs of health problems.

Code

This section contains the code of the program.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <ctime>
#include <string>
#include <vector>
#include <filesystem>
#include <chrono>
#include <sstream>
```

```
#include <iomanip>
#include <map>
using namespace std;

struct User { //All user data
    string username;
    string password;
    string name;
    int age;
    double height;
    double weight;
    string activityLvl;
    int dailykcalGoal;
    int kcalRemaining;
    string lastResetDate;
};

struct Meals { //Meal data
    string food;
    int calories;
    string mealType;
    string date;
};

struct Vitals { //Vitals
    int RHR;
    int o2Lvl;
    string bloodPressure;
    string date;
};

//Function declarations
void startUI();
void signup();
void saveAcc(const User& user);
void login();
bool verifyAcc(const string& username, const string& password);
void accManager(const string& username);
void updateDetails(const string& username);
void mealLog(const string& username);
void saveMeal(const string& username, const Meals& meal);
void vitalLog(const string& username);
void saveVitals(const string& username, const Vitals& vital);
void deleteAccount(const string& username);
vector<Meals> loadMealData(const string& username);
vector<Vitals> loadVitalsData(const string& username);
void checkImprovements(const string& username);
void HRDifference(const vector<Vitals>& vitals);
void O2Difference(const vector<Vitals>& vitals);
void BPDifference(const vector<Vitals>& vitals);
```

```
bool updateDay(const string& username);
User loadData(const string& username);
string getCurrentDate();
void WeeklyAverages(const string& username);
void MealWeeklyAverages(const string& username);
void VitalWeeklyAverages(const string& username);
string getWeekFromDate(const string& date);

int main() { //Runs the log in start UI
    startUI();
    return 0;
}

void startUI() { //Main UI
    int choice;

    do {
        cout << "\n----WELCOME TO MEDTRACK----\n";
        cout << "1. Log-in\n";
        cout << "2. Sign-up\n";
        cout << "3. Exit\n";
        cout << "Choice: ";
        cin >> choice;
        cin.ignore();

        if(!cin.fail()){
            switch(choice){
            case 1:
                login();
                break;
            case 2:
                signup();
                break;
            case 3:
                cout << "Thank you for using MedTrack!\n";
                break;
            default:
                cout << "Invalid input.\n";
           }
        } else {
            cin.clear();
            cin.ignore(100, '\n');
            cout << "Invalid input! Please choose a number from 1 - 3.\n";
        }

    }while(choice != 3);
}

void signup() { //Sign-up UI
    User newUser;
```

```
cout << "\n----SIGN UP----\n";
cout << "\nEnter username: ";
getline(cin, newUser.username);

//Turns username into filename
string filename = "users/" + newUser.username + ".csv";

//check if file exists
ifstream file(filename);
if(file.is_open()) {
    cout << "Username already exists. Log-in or create a new username.\n";
    file.close();
    return;
}
file.close();

cout << "\nEnter password: ";
getline(cin, newUser.password);

cout << "\nEnter name: ";
getline(cin, newUser.name);

cout << "\nEnter age: ";
cin >> newUser.age;
while(cin.fail() || newUser.age <= 0) {
    cin.clear();
    cin.ignore(100, '\n');
    cout << "Age must exceed 0!\nEnter a valid age: ";
    cin >> newUser.age;
}
cin.ignore();

cout << "\nEnter height(in cm): ";
cin >> newUser.height;
while(cin.fail() || newUser.height <= 0) {
    cin.clear();
    cin.ignore(100, '\n');
    cout << "Height must exceed 0!\nEnter a valid height: ";
    cin >> newUser.height;
}
cin.ignore();

cout << "\nEnter weight(in kg): ";
cin >> newUser.weight;
while(cin.fail() || newUser.weight <= 0) {
    cin.clear();
    cin.ignore(100, '\n');
    cout << "Weight must exceed 0!\nEnter a valid weight: ";
    cin >> newUser.weight;
    }
    cin.ignore();

    cout << "\nEnter activity level(sedentary/light/moderate/active): ";
    getline(cin, newUser.activityLvl);

    cout << "\nEnter daily calorie goal: ";
    cin >> newUser.dailykcalGoal;
    while(cin.fail() || newUser.dailykcalGoal <= 0) {
        cin.clear();
        cin.ignore(1000, '\n');
        cout << "Calorie goal must exceed 0!\nEnter a valid calorie goal: ";
        cin >> newUser.dailykcalGoal;
    }
    cin.ignore();

    newUser.kcalRemaining = newUser.dailykcalGoal;
    newUser.lastResetDate = getCurrentDate();

    saveAcc(newUser);
    cout << "\nAccount created successfully!\n";
}

void saveAcc(const User& user) { //Creates account file containing details
    filesystem::create_directory("users"); //Creates directory in current directory of the project file
    string filename = "users/" + user.username + ".csv";
    ofstream file(filename); //Inputs values into the file
        if(file.is_open()) {
            file << "username" << "," << user.username << "," << "\n"
                 << "password" << "," << user.password << "," << "\n"
                 << "name" << "," << user.name << "," << "\n"
                 << "age" << "," << user.age << "," << "\n"
                 << "height" << "," << user.height << "," << "\n"
                 << "weight" << "," << user.weight << "," << "\n"
                 << "activity level" << "," << user.activityLvl << "," << "\n"
                 << "daily calorie goal" << "," << user.dailykcalGoal << "," << "\n"
                 << "remaining calories" << "," << user.kcalRemaining << "," << "\n"
                 << "last reset date" << "," << user.lastResetDate << "," << "\n";
            file.close();
        }
}

void login() { //Gets log in details
    string username, password;

    cout << "\n----LOG IN----\n";
    cout << "Enter username: ";
    getline(cin, username);
    cout << "Enter password: ";
    getline(cin, password);
    if(verifyAcc(username, password)) {
        cout << "\nLogged in succesfully!\n";
        updateDay(username);
        accManager(username);
    } else {
        cout << "\nInvalid username or password.\n";
    }
}

bool verifyAcc(const string& username, const string& password) { //Checks if file exists and details are correct
    string filename = "users/" + username + ".csv"; //Turns username into filename
    ifstream file(filename);
    if(!file.is_open()) { //Checks if file exists
        return false;
    }

    string line;
    string savedPassword;

    while(getline(file, line)) { // Reads file line by line, stores in 'line' variable
        stringstream ss(line);
        string label, value;

        getline(ss, label, ','); //Reads until first comma, stores as labe
        getline(ss, value, ','); //Reads until the second comma, stores as value

        if(label == "password") { //If the label is password, gets value
            savedPassword = value;
            break;
        }
    }

    file.close();
    if (savedPassword == password) { //Compares the saved password from inputted password
        return true;
    }
    else {
        return false;
    }
}

void accManager(const string& username) { //Shows the actions the user can take
    int choice;

     do {
        cout << "\n----ACCOUNT MANAGEMENT----\n";
        cout << "1. Update personal details\n";
        cout << "2. Input meals\n";
        cout << "3. Input vitals\n";
        cout << "4. Check improvements\n";
        cout << "5. Delete account\n";
        cout << "6. Log out\n";
        cout << "Choice: ";
        cin >> choice;
        cin.ignore();

        if (!cin.fail()){
            switch(choice) {
                case 1:
                    updateDetails(username);
                    break;
                case 2:
                    mealLog(username);
                    break;
                case 3:
                    vitalLog(username);
                    break;
                case 4:
                    checkImprovements(username);
                    break;
                case 5:
                    deleteAccount(username);
                    return;
                case 6:
                    cout << "\nLogged out successfully!\n";
                    break;
                default:
                    cout << "\nInvalid choice. Try again.\n";
                }
        } else {
            cin.clear();
            cin.ignore(100, '\n');
            cout << "Invalid input! Please choose a number from 1 - 3.\n";
        }
    }while(choice != 6);
}

User loadData(const string& username) { //Reads file and stores the user datainto a structure
    User user;
    string filename = "users/" + username + ".csv";
    ifstream file(filename);
    if(file.is_open()) {
        string line;
        while(getline(file, line)) { //Looping to read every line in csv file
            stringstream ss(line);
            string label, value;

            getline(ss, label, ',');
            getline(ss, value, ',');
                if(label == "username") { //Storing of values in each field
                    user.username = value;}
                else if(label ==  "password") {
                    user.password = value;}
                else if(label ==  "name") {
                    user.name = value;}
                else if(label ==  "age") {
                    user.age = stoi(value);}
                else if(label ==  "height") {
                    user.height = stod(value);}
                else if(label ==  "weight") {
                    user.weight = stod(value);}
                else if(label ==  "activity level") {
                    user.activityLvl = value;}
                else if(label ==  "daily calorie goal") {
                    user.dailykcalGoal = stoi(value);}
                else if(label ==  "remaining calories") {
                    user.kcalRemaining = stoi(value);}
                else if(label ==  "last reset date") {
                    user.lastResetDate = value;}
        }
        file.close();
    }
    return user;
}

void updateDetails(const string& username) { //Reads user details file and allows the user to update it
    User user = loadData(username);
    int choice;

    do {
        cout << "\n----UPDATE PERSONAL DETAILS----\n";
        cout << "1. Update name\n";
        cout << "2. Update age\n";
        cout << "3. Update height\n";
        cout << "4. Update weight\n";
        cout << "5. Update activity level\n";
        cout << "6. Update daily calorie goal\n";
        cout << "7. Exit\n";
        cout << "Choice: ";
        cin >> choice;
        cin.ignore();

        if (!cin.fail()){
            switch(choice) { //Break; means that user can only update one at a time
                case 1:
                    cout << "Enter new name: ";
                    getline(cin, user.name);
                    break;
```

```
case 2:
    cout << "\nEnter new age: ";
    cin >> user.age;
    while(cin.fail() || user.age <= 0) {
        cin.clear();
        cin.ignore(100, '\n');
        cout << "Age must exceed 0!\nEnter a valid age: ";
        cin >> user.age;
    }
    cin.ignore();
    break;
case 3:
    cout << "\nEnter new height(in cm): ";
    cin >> user.height;
    while(cin.fail() || user.height <= 0) {
        cin.clear();
        cin.ignore(100, '\n');
        cout << "Height must exceed 0!\nEnter a valid height: ";
        cin >> user.height;
    }
    cin.ignore();
    break;
case 4:
    cout << "Enter new weight (kg): ";
    cin >> user.weight;
    while(cin.fail() || user.weight <= 0) {
        cin.clear();
        cin.ignore(100, '\n');
        cout << "Weight must exceed 0!\nEnter a valid weight: ";
        cin >> user.weight;
    }
    cin.ignore();
    break;
case 5:
    cout << "Enter new activity level: ";
    getline(cin, user.activityLvl);
    break;
case 6:
    cout << "Enter new daily calorie goal: ";
    cin >> user.dailykcalGoal;
    while(cin.fail() || user.dailykcalGoal <= 0) {
        cin.clear();
        cin.ignore(1000, '\n');
        cout << "Calorie goal must exceed 0!\nEnter a valid calorie goal: ";
        cin >> user.dailykcalGoal;
    }
    user.kcalRemaining = user.dailykcalGoal;
    cin.ignore();
    break;
case 7:
```

```
                break;
            default:
                cout << "Invalid choice! Please try again.\n";
            }
        } else {
            cin.clear();
            cin.ignore(100, '\n');
            cout << "Invalid input! Please choose a number from 1 - 3.\n";
        }

        if(choice >= 1 && choice <= 6) { //Saving details if user chose to update any detail
            saveAcc(user);
            cout << "Details updated successfully!\n";
        }
    } while(choice != 7);
}

string getCurrentDate() { //Function to return date in DD/MM/YYYY form
    time_t now = time(0);
    tm* ltm = localtime(&now);

    stringstream ss;
    ss << setw(2) << setfill('0') << ltm->tm_mday << "/"
       << setw(2) << setfill('0') << 1 + ltm->tm_mon << "/"
       << 1900 + ltm->tm_year;

    return ss.str();
}

void mealLog(const string& username) { // Gets meal info from user and calls the save meals function
    Meals meal;

    cout << "\n----MEALS----\n";
    cout << "Enter food: ";
    getline(cin, meal.food);

    while (1) {
        cout << "Enter calories: ";
        cin >> meal.calories;

        if (cin.fail() || meal.calories < 0) {
            cin.clear();
            cin.ignore(100, '\n');
            cout << "Invalid input!\n";
        } else {
            cin.ignore(100, '\n');
            break;
        }
    }
    cout << "Enter meal type(Breakfast/Lunch/Dinner/Snack): ";
    getline(cin, meal.mealType);

    meal.date = getCurrentDate();

    saveMeal(username, meal);

    User user = loadData(username);
    updateDay(username); //checks if day is diferent
    user = loadData(username);

    user.kcalRemaining -= meal.calories; //displays remaining calories, and if it's already exceeded
    if(user.kcalRemaining < 0) {
        cout << "Warning! You've exceeded your daily calorie goal by "
             << abs(user.kcalRemaining) << " calories.\n";}
    saveAcc(user);
    cout << "Meal logged successfully!\n";
    cout << "Remaining calories for today: " << user.kcalRemaining << endl;
}

void saveMeal(const string& username, const Meals& meal) { //Inserts meals into a new csv file
    string filename = "users/" + username + "_meals.csv";
    ofstream mealFile(filename, ios::app);
    if(!mealFile.is_open()) {  //If can't open file with the username
        cerr << "Error: Cannot open meal file for " << username << endl;
        return;
    }
    mealFile.seekp(0, ios::end); //Checks for header labels, puts one if there's none
    if(mealFile.tellp() == 0) {
        mealFile << "Date" << "," << "Food" << "," << "Calories" << "," << "MealType" << "\n";
    }
    mealFile << meal.date << ","
             << meal.food << ","
             << meal.calories << ","
             << meal.mealType << "\n";
    mealFile.close();
}

void vitalLog(const string& username) { //Asks for user vitals one by one and calls save vitals function
    int choice;
    Vitals vital;
    vital.date = getCurrentDate();

    do {
        cout << "\n----VITALS----\n";
        cout << "1. Log resting heart rate\n";
        cout << "2. Log O2 levels\n";
        cout << "3. Log blood pressure\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
```

```
cin >> choice;
cin.ignore();

Vitals vital;
vital.date = getCurrentDate();
vital.RHR = 0;
vital.o2Lvl = 0;
vital.bloodPressure = "";

if (!cin.fail()){
    switch(choice) {
        case 1:
            while (1) {
                cout << "Enter resting heart rate (bpm): ";
                cin >> vital.RHR;

                if (cin.fail() || vital.RHR < 0) {
                    cin.clear();
                    cin.ignore(100, '\n');
                    cout << "Invalid input!\n";
                } else {
                    cin.ignore(100, '\n');
                    break;
                }
            }
            saveVitals(username, vital);
            cout << "Resting heart rate logged successfully!\n";
            break;
        case 2:
            while (1) {
                cout << "Enter O2 level (%): ";
                cin >> vital.o2Lvl;

                if (cin.fail() || vital.o2Lvl < 0) {
                    cin.clear();
                    cin.ignore(100, '\n');
                    cout << "Invalid input!\n";
                } else {
                    cin.ignore(100, '\n');
                    break;
                }
            }
            saveVitals(username, vital);
            cout << "O2 level logged successfully!\n";
            break;
        case 3:
            cout << "Enter blood pressure (e.g., 120/80): ";
            getline(cin, vital.bloodPressure);
            saveVitals(username, vital);
            cout << "Blood pressure logged successfully!\n";
```

```
                break;
            case 4:
                break;
            default:
                cout << "Invalid choice! Please try again.\n";
        }
    } else {
        cin.clear();
        cin.ignore(100, '\n');
        cout << "Invalid input! Please choose a number from 1 - 3.\n";
    }
  } while(choice != 4);
}

void saveVitals(const string& username, const Vitals& newVital) { //Saves vitals info into csv file
  filesystem::create_directory("users");

  string filename = "users/" + username + "_vitals.csv";

  vector<Vitals> allVitals = loadVitalsData(username);

  bool foundToday = false; // Check if we already have an entry for today
  for(auto& vital : allVitals) {
    if(vital.date == newVital.date) { // Update existing entry with new data
      if(newVital.RHR != 0) vital.RHR = newVital.RHR;
      if(newVital.o2Lvl != 0) vital.o2Lvl = newVital.o2Lvl;
      if(!newVital.bloodPressure.empty()) vital.bloodPressure = newVital.bloodPressure;
      foundToday = true;
      break;
    }
  }

  if(!foundToday) {  // If no entry for today, add the new one
    allVitals.push_back(newVital);
  }

  ofstream vitalFile(filename); // Rewrite the entire file with updated data
  if(!vitalFile.is_open()) {
    cerr << "Error: Cannot open vitals file for " << username << "\n";
    return;
  }

  vitalFile << "Date" << "," << "RHR" << "," << "O2 Lvl" << "," << "BP" << "\n";

  for(const auto& vital : allVitals) { //Outputs all vitals in one line
    vitalFile << vital.date << ","
              << vital.RHR << ","
              << vital.o2Lvl << ","
              << vital.bloodPressure << "\n";
  }
```

```
    vitalFile.close();
}
void deleteAccount(const string& username) { //deletes user account if they wish
    char input;
    string password;

    cout << "\n----DELETE ACCOUNT----\n";
    cout << "Are you sure you want to delete this account? (Y/N): ";
    cin >> input;
    cin.ignore();

    if(input == 'Y' || input == 'y') { //Asks for passw. if user inputs Y/y
        cout << "Enter password to confirm: ";
        getline(cin, password);

        if(verifyAcc(username, password)) { //If password is correct, deletes all files
            string userFile = "users/" + username + ".csv";
            string mealFile = "users/" + username + "_meals.csv";
            string vitalFile = "users/" + username + "_vitals.csv";

            remove(userFile.c_str());
            remove(mealFile.c_str());
            remove(vitalFile.c_str());

            cout << "Account deleted successfully!\n";
        } else {
            cout << "Incorrect password! Account deletion cancelled.\n";
        }
    } else {
        cout << "Account deletion cancelled.\n";
    }
}

vector<Meals> loadMealData(const string& username) { //Reads meal file and stores value in a vector
    vector<Meals> meals;
    string filename = "users/" + username + "_meals.csv";
    ifstream file(filename);

    if(file.is_open()) {
        string line;
        getline(file, line);

        while(getline(file, line)) { //Reading line by line
            stringstream ss(line);
            string item;
            Meals meal;

            getline(ss, meal.date, ',');
            getline(ss, meal.food, ',');
            getline(ss, item, ',');
            if(!item.empty()) {
                meal.calories = stoi(item);
            } else {
                meal.calories = 0;
            }

            getline(ss, meal.mealType, ',');

            meals.push_back(meal); //Stores structure in a vector
        }
        file.close();
    }
    return meals;
}

vector<Vitals> loadVitalsData(const string& username) { //Reads vitals file and stores it in vector(Similar to
loadMealData)
    vector<Vitals> vitals;
    string filename = "users/" + username + "_vitals.csv";
    ifstream file(filename);

    if(file.is_open()) {
        string line;
        getline(file, line);

        while(getline(file, line)) {
            stringstream ss(line);
            string item;
            Vitals vital;

            getline(ss, vital.date, ',');

            getline(ss, item, ',');
            vital.RHR = item.empty() ? 0 : stoi(item);

            getline(ss, item, ',');
            vital.o2Lvl = item.empty() ? 0 : stoi(item);

            getline(ss, vital.bloodPressure, ',');

            vitals.push_back(vital);
        }
        file.close();
    }
    return vitals;
}

void checkImprovements(const string& username) { //Displays options for checking improvements, calls functions
```

```
    vector<Vitals> vitals = loadVitalsData(username);
    vector<Meals> meals = loadMealData(username);

    if(vitals.empty() && meals.empty()) {
        cout << "\nNo data available for analysis.\n";
        return;
    }

    int choice;
    do {
        cout << "\n----CHECK IMPROVEMENTS----\n";
        cout << "1. Show RHR difference\n";
        cout << "2. Show O2 lvl difference\n";
        cout << "3. Show BP difference\n";
        cout << "4. Show weekly averages\n";
        cout << "5. Exit\n";
        cout << "Choice: ";
        cin >> choice;
        cin.ignore();

        if (!cin.fail()){
                switch(choice) {
                case 1:
                    HRDifference(vitals);
                    break;
                case 2:
                    O2Difference(vitals);
                    break;
                case 3:
                    BPDifference(vitals);
                    break;
                case 4:
                    WeeklyAverages(username);
                    break;
                case 5:
                    break;
                default:
                    cout << "Invalid choice! Please try again.\n";
            }
        } else {
            cin.clear();
            cin.ignore(100, '\n');
            cout << "Invalid input! Please choose a number from 1 - 3.\n";
        }
    } while(choice != 5);
}

void HRDifference(const vector<Vitals>& vitals) { //Displays heart rate difference from first log
    if(vitals.size() < 2) { //checks for content
        cout << "Need at least 2 readings to show difference.\n";
        return;
    }
    Vitals first = vitals.front(); //Assigning the first log and the last log
    Vitals last = vitals.back();
    int difference = last.RHR - first.RHR; //Difference

    cout << "\n----RESTING HEART RATE DIFFERENCE----\n";
    cout << "First reading (" << first.date << "): " << first.RHR << " bpm\n";
    cout << "Last reading (" << last.date << "): " << last.RHR << " bpm\n";
    cout << "Difference: " << (difference > 0 ? "+" : "") << difference << " bpm\n"; //shows if positive or negative
change
}

void O2Difference(const vector<Vitals>& vitals) { //Shows O2 levels difference like in RHR difference
    if(vitals.size() < 2) {
        cout << "Need at least 2 readings to show difference.\n";
        return;
    }
    Vitals first = vitals.front();
    Vitals last = vitals.back();
    int difference = last.o2Lvl - first.o2Lvl;

    cout << "\n----O2 LEVEL DIFFERENCE----\n";
    cout << "First reading (" << first.date << "): " << first.o2Lvl << "%\n";
    cout << "Last reading (" << last.date << "): " << last.o2Lvl << "%\n";
    cout << "Difference: " << (difference > 0 ? "+" : "") << difference << "%\n";
}

void BPDifference(const vector<Vitals>& vitals) { //Shows Blood pressure difference like in RHR difference
    vector<Vitals> bpReadings;
    for(const auto& v : vitals) { //Checks for value since bp is a string type, not like O2 and RHR which have default
of 0(int)
        if(!v.bloodPressure.empty()) {
            bpReadings.push_back(v);
        }
    }

    if(bpReadings.size() < 2) {
        cout << "Need at least 2 blood pressure readings to show difference.\n";
        return;
    }

    Vitals first = bpReadings.front();
    Vitals last = bpReadings.back();

    cout << "\n----BLOOD PRESSURE DIFFERENCE----\n";
    cout << "First reading (" << first.date << "): " << first.bloodPressure << "\n";
    cout << "Last reading (" << last.date << "): " << last.bloodPressure << "\n";
}
```

```
bool updateDay(const string& username) { //CHecks if date changed
    User user = loadData(username);
    string today = getCurrentDate();
    bool dayChanged = false;

    if (user.lastResetDate != today) {
        user.kcalRemaining = user.dailykcalGoal;
        user.lastResetDate = today;
        saveAcc(user);
        dayChanged = true;
    }

    return dayChanged; //Returns true or false
}

void WeeklyAverages(const string& username) { //Displays choices for displaying weekly average
    int choice;
    do {
        cout << "\n----WEEKLY AVERAGES----\n";
        cout << "1. Meal averages\n";
        cout << "2. Vital averages\n";
        cout << "3. Exit\n";
        cout << "Choice: ";
        cin >> choice;
        cin.ignore();

        if (!cin.fail()){
            switch(choice) {
                case 1:
                    MealWeeklyAverages(username);
                    break;
                case 2:
                    VitalWeeklyAverages(username);
                    break;
                case 3:
                    break;
                default:
                    cout << "Invalid choice! Please try again.\n";
            }
        } else {
            cin.clear();
            cin.ignore(100, '\n');
            cout << "Invalid input! Please choose a number from 1 - 3.\n";
        }
    } while(choice != 3);
}

string getWeekFromDate(const string& date) { //Returns week string
    int year, month, day;
    char slash;
    stringstream ss(date); //Reads date in the arguments
    if (!(ss >> day >> slash >> month >> slash >> year)) {
        return "Unknown-Week";
    }

    if (month < 1 || month > 12 || day < 1 || day > 31 || year < 1900) {
        return "Invalid-Date";
    }

    tm timeinfo = {}; //Stores years and dates in a structure
    timeinfo.tm_year = year - 1900;
    timeinfo.tm_mon = month - 1;
    timeinfo.tm_mday = day;
    mktime(&timeinfo); //FUnction to calculate information

    int week = (timeinfo.tm_yday / 7) + 1;
    if (week < 1) week = 1;
    if (week > 53) week = 53;

    stringstream weekSS;
    weekSS << year << "-W" << setw(2) << setfill('0') << week;
    return weekSS.str();
}

void MealWeeklyAverages(const string& username) { //Calculates and displays weekly average of meals
    vector<Meals> meals = loadMealData(username);

    if(meals.empty()) { //Checks for content
        cout << "\nNo meal data available.\n";
        return;
    }

    map<string, vector<int>> weeklyCalories; //Maps values into weeks (Key: week string)

    for(const auto& meal : meals) { //Storing calories in a vector for each week
        string week = getWeekFromDate(meal.date);
        weeklyCalories[week].push_back(meal.calories);
    }

    cout << "\n----WEEKLY MEAL AVERAGES----\n";
    cout << "Week\t\tAvg Calories\tMeal Count\n";

    for(const auto& week : weeklyCalories) { //Computes average for each week
        double avgCalories = 0;
        for(int cal : week.second) {
            avgCalories += cal;
        }
        avgCalories /= week.second.size();

        cout << week.first << "\t" << fixed << setprecision(1) << avgCalories << " kcal\t"
```

```
            << week.second.size() << " meals\n";
    }
}

void VitalWeeklyAverages(const string& username) { //Calculates and displays weekly average for vitals
    vector<Vitals> vitals = loadVitalsData(username); //Calls to read file

    if(vitals.empty()) { //Check for content
        cout << "\nNo vital data available.\n";
        return;
    }

    map<string, vector<int>> weeklyRHR; //Maps weeks into vector (Key: week string)
    map<string, vector<int>> weeklyO2;
    map<string, vector<string>> weeklyBP;

    for(const auto& vital : vitals) { //Sorts data by week into vectors mapped
        string week = getWeekFromDate(vital.date);
        if(!vital.RHR == 0) {
            weeklyRHR[week].push_back(vital.RHR);
        }
        if(!vital.o2Lvl == 0) {
            weeklyO2[week].push_back(vital.o2Lvl);
        }
        if(!vital.bloodPressure.empty()) {
            weeklyBP[week].push_back(vital.bloodPressure);
        }
    }

    cout << "\n----WEEKLY VITAL AVERAGES----\n";
    cout << "Week\t\tAvg RHR\tAvg O2\tBP Readings\n";

    for(const auto& week : weeklyRHR) { //Calculating weekly average
        double avgRHR = 0, avgO2 = 0;

        for(int rhr : week.second) {
            avgRHR += rhr;
        }
        avgRHR /= week.second.size();

        if(weeklyO2.find(week.first) != weeklyO2.end()) {
            for(int o2 : weeklyO2[week.first]) {
                avgO2 += o2;
            }
            avgO2 /= weeklyO2[week.first].size();
        }


        int bpCount = 0;
        if(weeklyBP.find(week.first) != weeklyBP.end()) { // Count BP readings
```

```
            bpCount = weeklyBP[week.first].size();
        }

        cout << week.first << "\t" << fixed << setprecision(1) << avgRHR << " bpm\t"
             << (avgO2 > 0 ? to_string((int)avgO2) + "%" : "N/A") << "\t"
             << bpCount << " readings\n";
    }
}
```

Figure 3. MedTrack C++ code

Seen above, in figure 3, is the final code for MedTrack. It is suggested that you first read through the pseudocode, or the flowchart, before reading through the C++ code. Here will be the brief summary of the code in figure 3 to serve as a guide when skimming through.

The first lines of code is for the inclusion of the built-in C++ libraries, after that up to line 68 are the initialization of all the structures, and variables that will be used in the main part of the code. All functions have the short declaration.

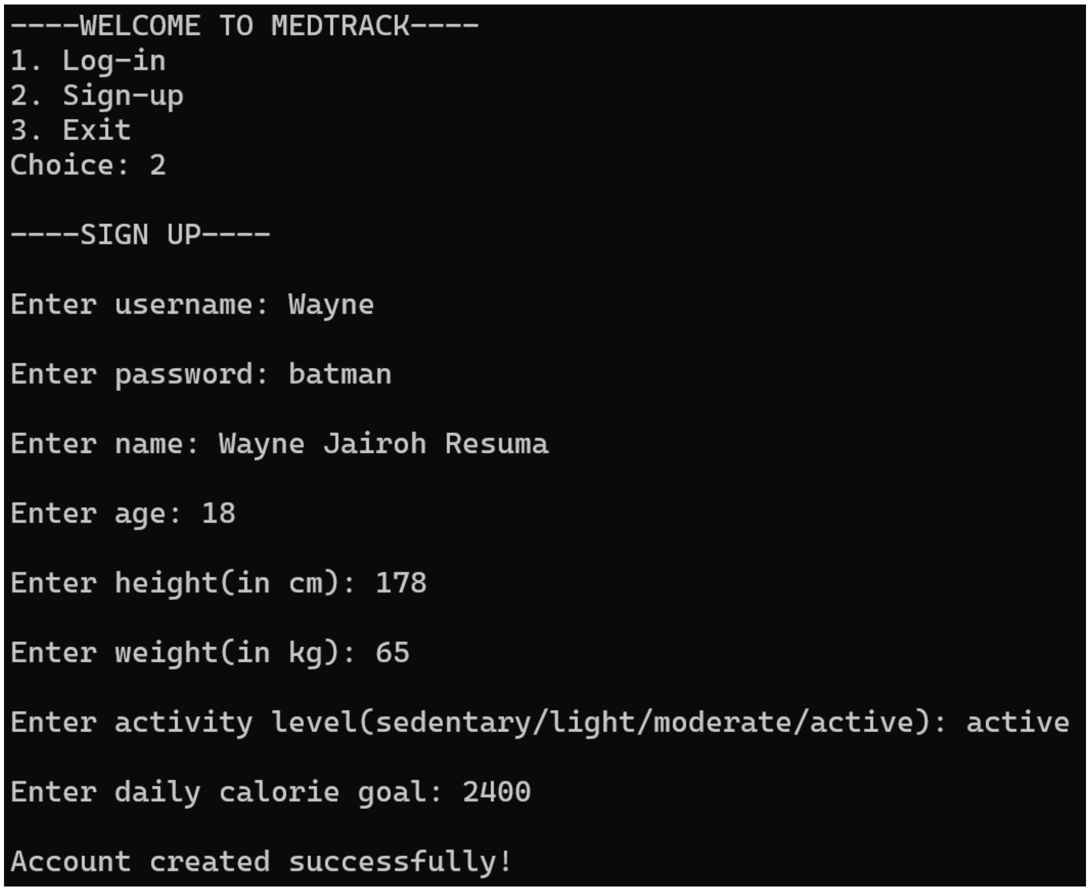
The main part of the code is only 4 lines long, most of all the program's functionality are handled by the functions declared in the previous lines of code. Each following section of code declares the function of the previously declared functions. Each function leading onto another function according to the pseudocode/flowchart seen in figures 1 and 2. Once the startUI() function is completed, the program closes.

Results and Discussion

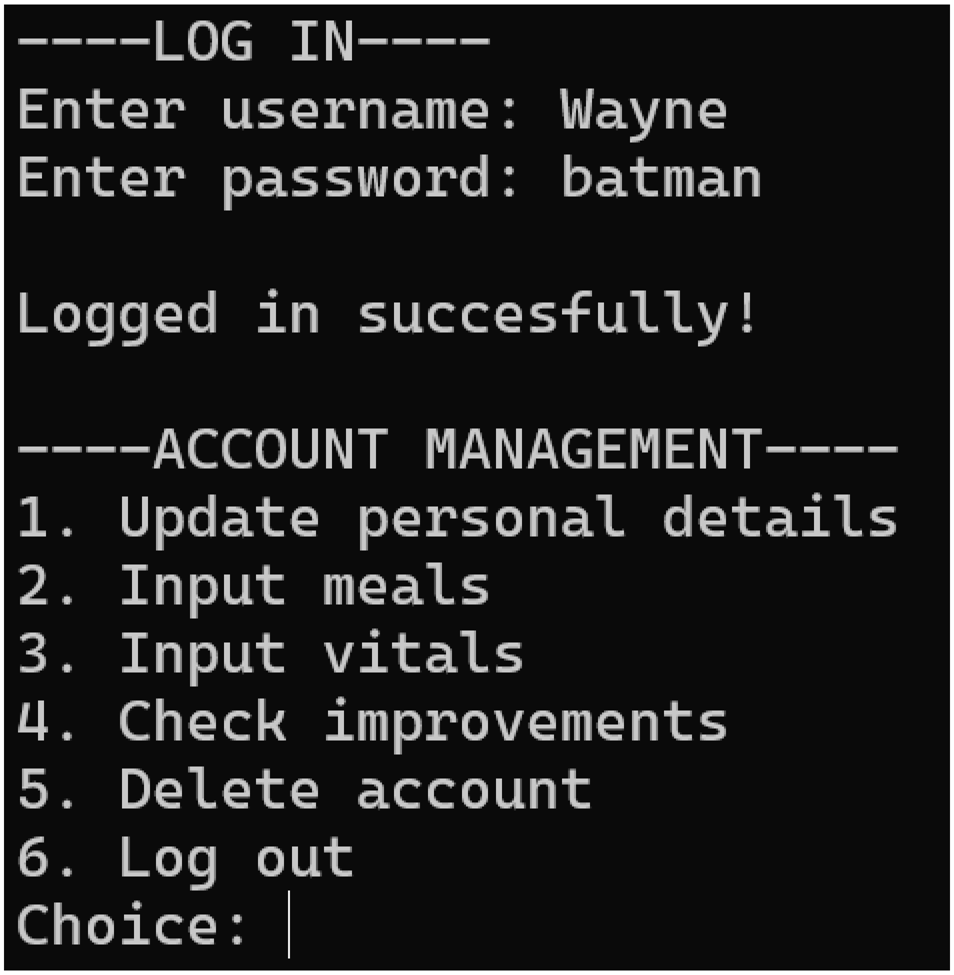
To evaluate the system's functionality with realistic data, a trial was conducted using health metrics collected from a local clinic. We obtained measurements for blood pressure, blood oxygen level, and resting heart rate from clinical equipment, and systematically input these data in our MedTrack system.

The testing successfully validated the system's core functionalities. The program successfully created user accounts and verified logins (Objective 1) with all personal data and clinical measurements being properly stored in CSV files. The data logging features (Objective 2) effectively processed daily entries for both vital signs and nutritional information. The weekly trend analysis (Objective 3) was able to show trends and differences, confirming the system's capability for consistent health monitoring.

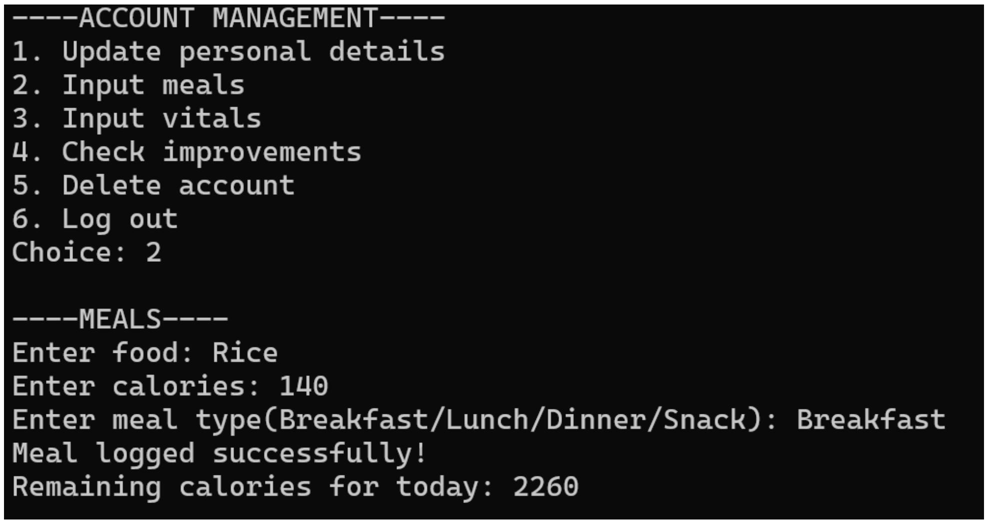
The results confirm that MedTrack provides a reliable foundation for personal health tracking. The system successfully meets core user needs through secure data management and consistent logging capabilities. Its effective performance with clinically sourced data demonstrates strong potential for practical use as a health monitoring tool.



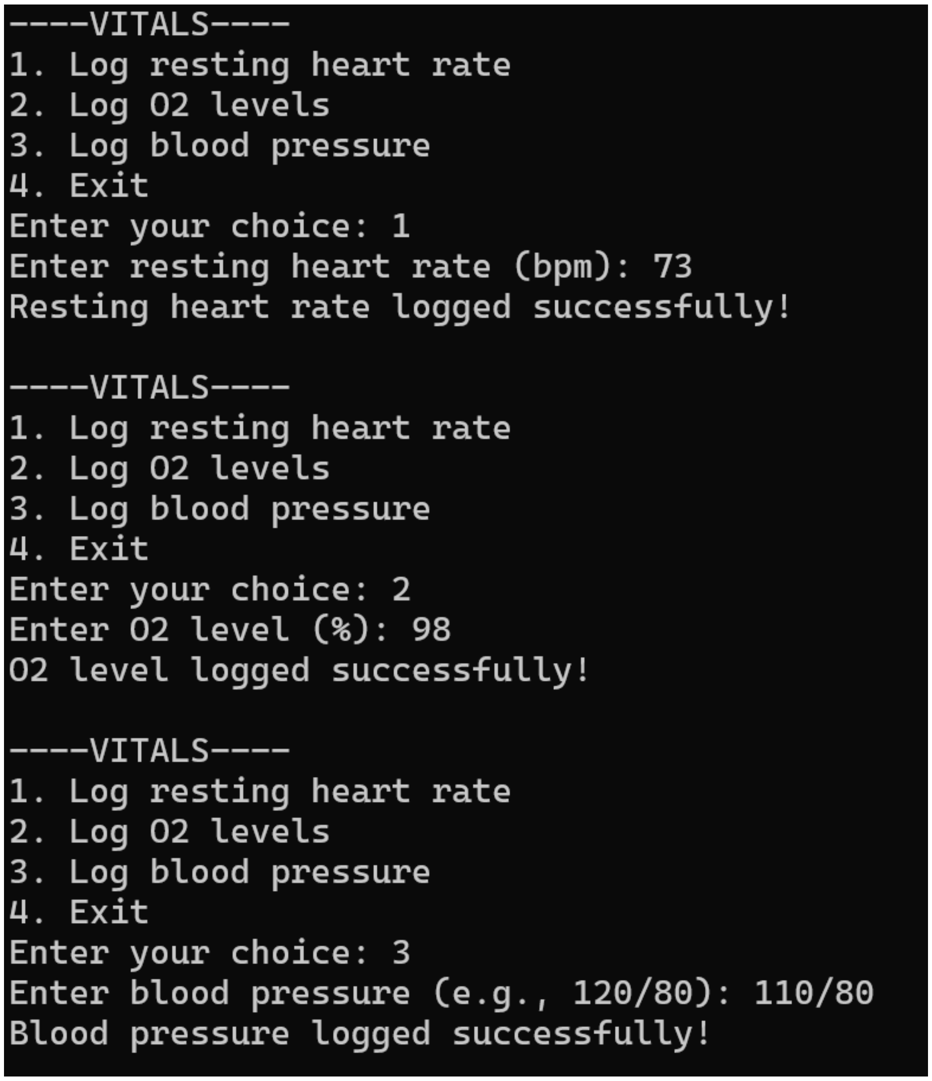
```
----WELCOME TO MEDTRACK----
1. Log-in
2. Sign-up
3. Exit
Choice: 2

----SIGN UP----

Enter username: Wayne

Enter password: batman

Enter name: Wayne Jairoh Resuma

Enter age: 18

Enter height(in cm): 178

Enter weight(in kg): 65

Enter activity level(sedentary/light/moderate/active): active

Enter daily calorie goal: 2400

Account created successfully!
```



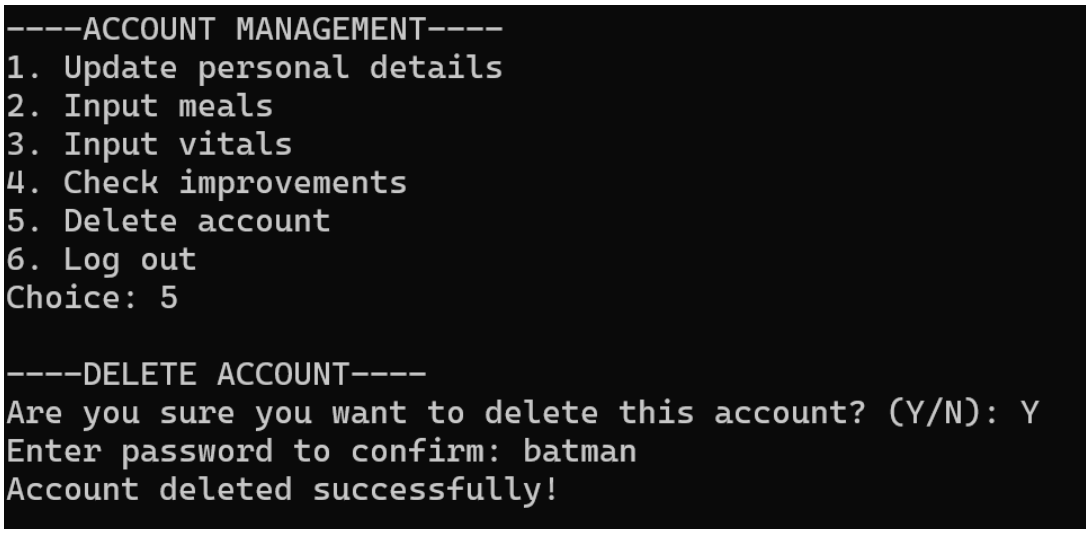
```
----LOG IN----
Enter username: Wayne
Enter password: batman

Logged in succesfully!

----ACCOUNT MANAGEMENT----
1. Update personal details
2. Input meals
3. Input vitals
4. Check improvements
5. Delete account
6. Log out
Choice: 
```



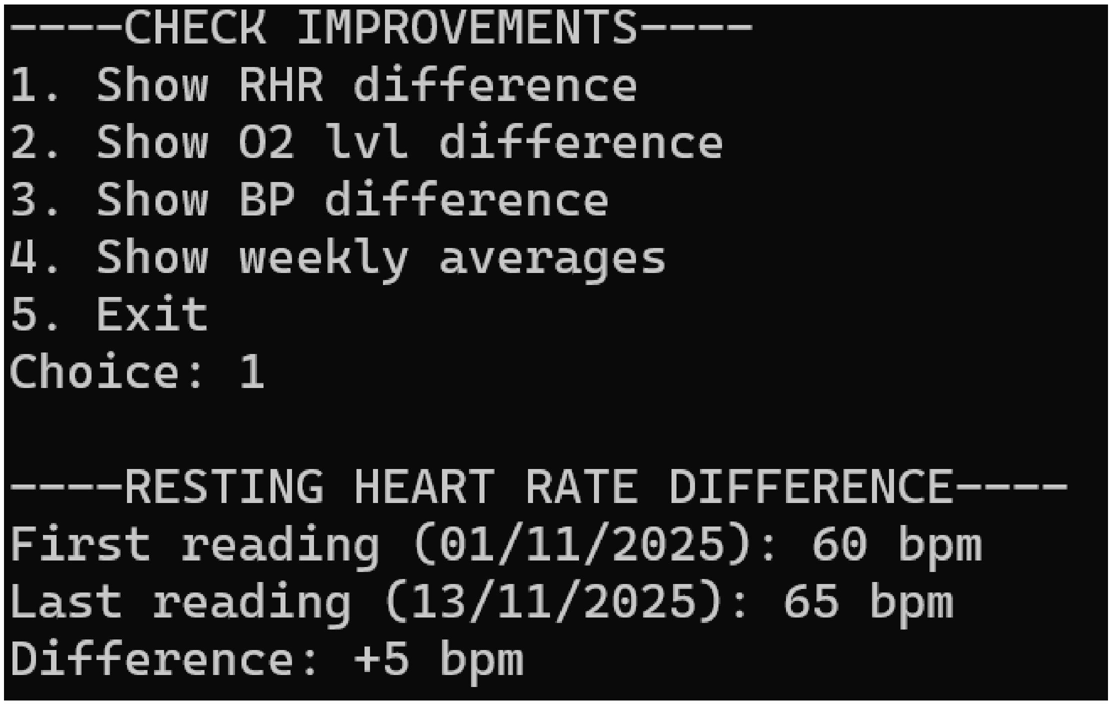
```
----ACCOUNT MANAGEMENT----
1. Update personal details
2. Input meals
3. Input vitals
4. Check improvements
5. Delete account
6. Log out
Choice: 2

----MEALS----
Enter food: Rice
Enter calories: 140
Enter meal type(Breakfast/Lunch/Dinner/Snack): Breakfast
Meal logged successfully!
Remaining calories for today: 2260
```



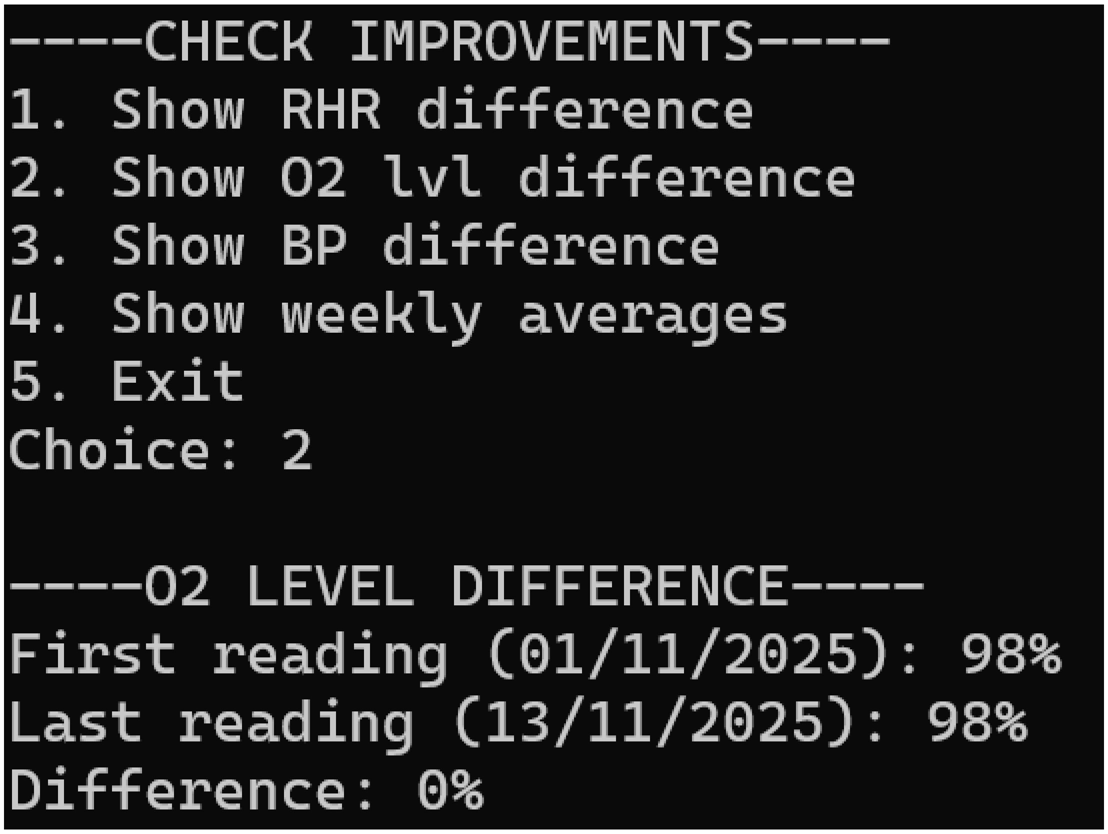
```
----VITALS----
1. Log resting heart rate
2. Log O2 levels
3. Log blood pressure
4. Exit
Enter your choice: 1
Enter resting heart rate (bpm): 73
Resting heart rate logged successfully!

----VITALS----
1. Log resting heart rate
2. Log O2 levels
3. Log blood pressure
4. Exit
Enter your choice: 2
Enter O2 level (%): 98
O2 level logged successfully!

----VITALS----
1. Log resting heart rate
2. Log O2 levels
3. Log blood pressure
4. Exit
Enter your choice: 3
Enter blood pressure (e.g., 120/80): 110/80
Blood pressure logged successfully!
```



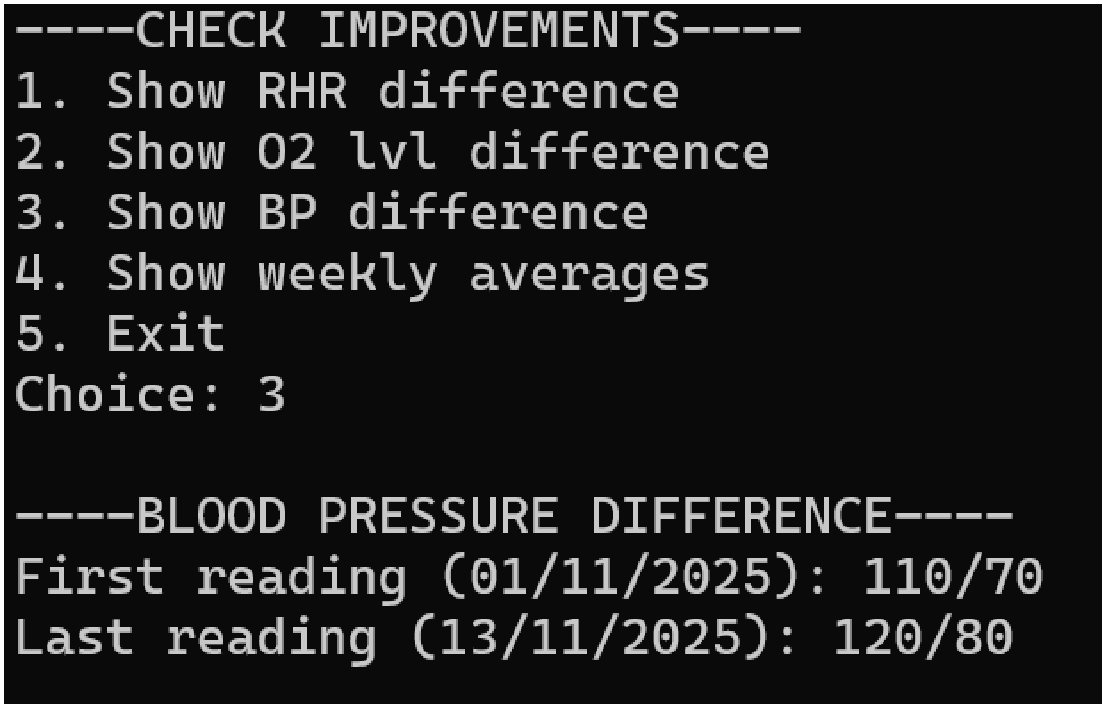
```
----ACCOUNT MANAGEMENT----
1. Update personal details
2. Input meals
3. Input vitals
4. Check improvements
5. Delete account
6. Log out
Choice: 5

----DELETE ACCOUNT----
Are you sure you want to delete this account? (Y/N): Y
Enter password to confirm: batman
Account deleted successfully!
```

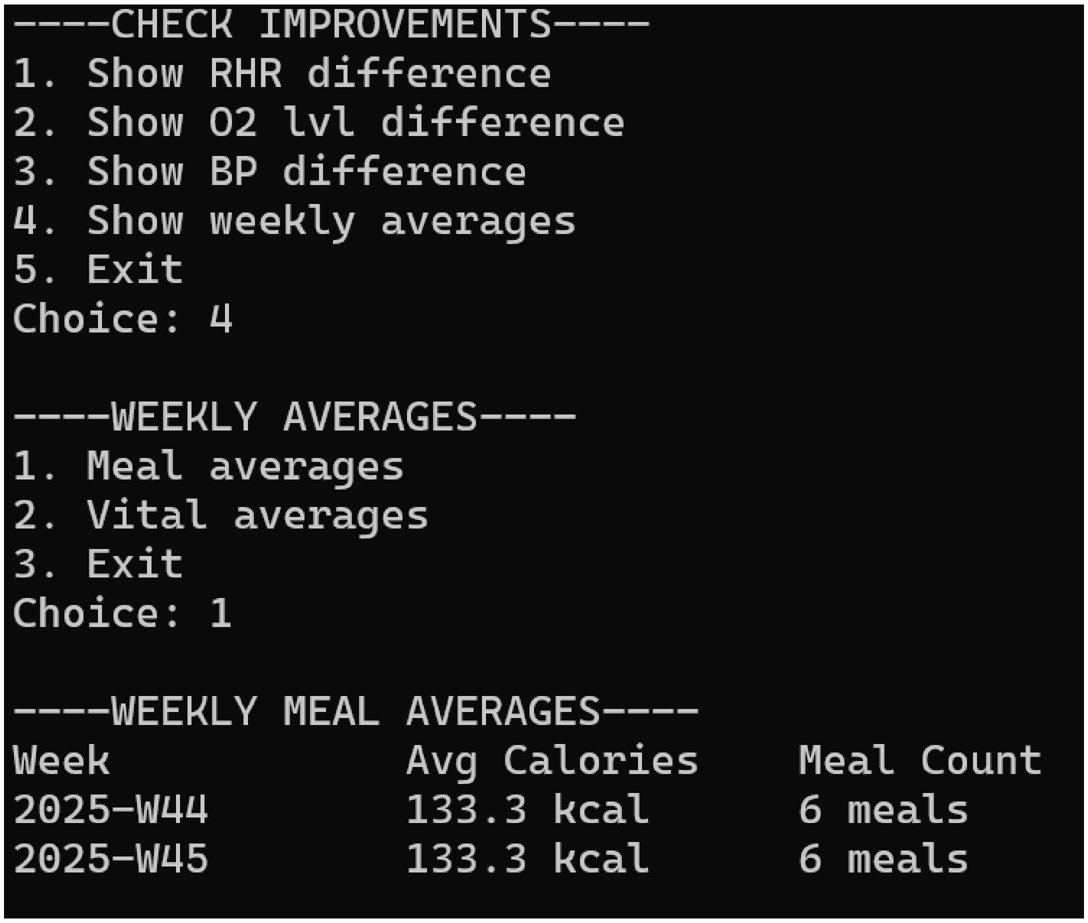


```
----CHECK IMPROVEMENTS----
1. Show RHR difference
2. Show O2 lvl difference
3. Show BP difference
4. Show weekly averages
5. Exit
Choice: 1

----RESTING HEART RATE DIFFERENCE----
First reading (01/11/2025): 60 bpm
Last reading (13/11/2025): 65 bpm
Difference: +5 bpm
```

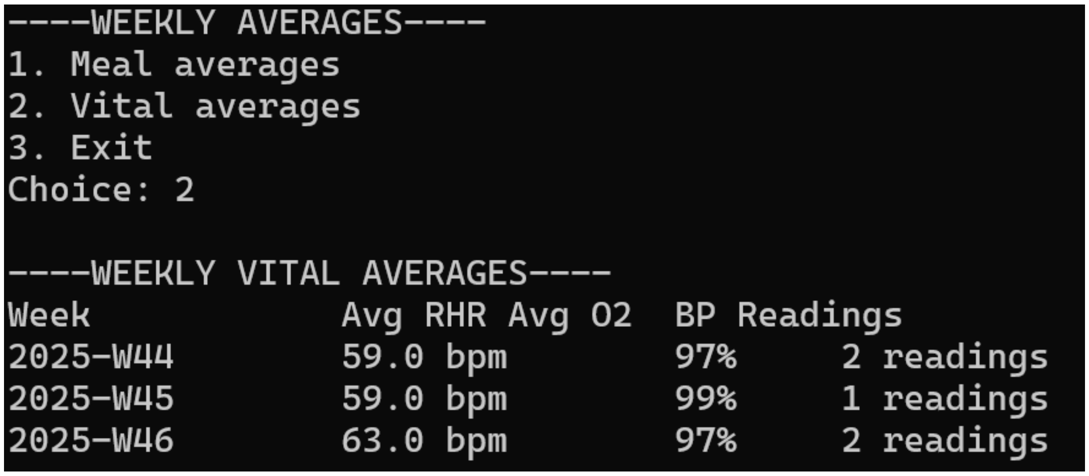


```
----CHECK IMPROVEMENTS----
1. Show RHR difference
2. Show O2 lvl difference
3. Show BP difference
4. Show weekly averages
5. Exit
Choice: 2

----O2 LEVEL DIFFERENCE----
First reading (01/11/2025): 98%
Last reading (13/11/2025): 98%
Difference: 0%
```


```
----CHECK IMPROVEMENTS----
1. Show RHR difference
2. Show O2 lvl difference
3. Show BP difference
4. Show weekly averages
5. Exit
Choice: 3

----BLOOD PRESSURE DIFFERENCE----
First reading (01/11/2025): 110/70
Last reading (13/11/2025): 120/80
```

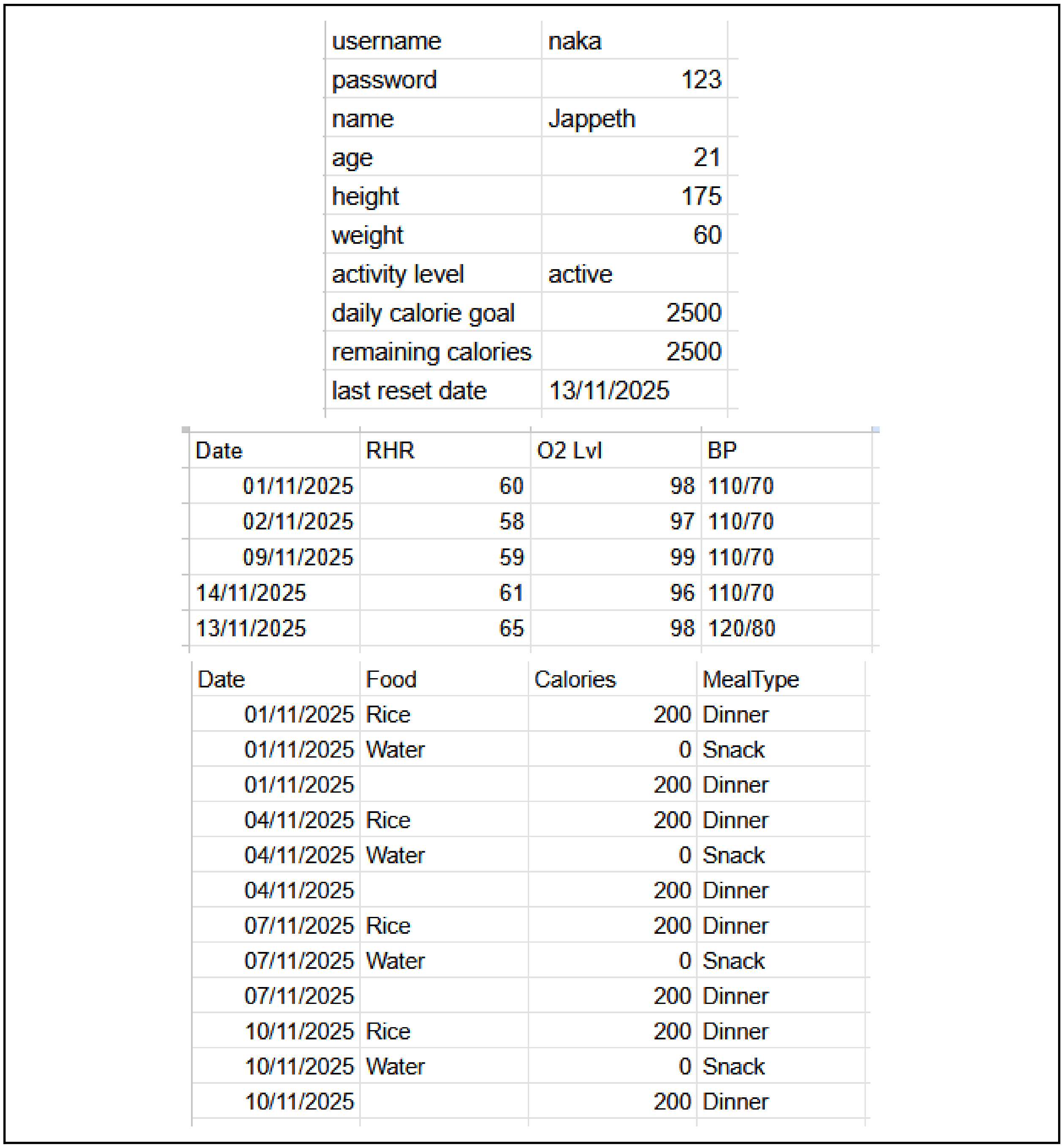


```
----CHECK IMPROVEMENTS----
1. Show RHR difference
2. Show O2 lvl difference
3. Show BP difference
4. Show weekly averages
5. Exit
Choice: 4

----WEEKLY AVERAGES----
1. Meal averages
2. Vital averages
3. Exit
Choice: 1

----WEEKLY MEAL AVERAGES----
Week            Avg Calories    Meal Count
2025-W44        133.3 kcal      6 meals
2025-W45        133.3 kcal      6 meals
```



```
----WEEKLY AVERAGES----
1. Meal averages
2. Vital averages
3. Exit
Choice: 2

----WEEKLY VITAL AVERAGES----
Week            Avg RHR Avg O2  BP Readings
2025-W44        59.0 bpm        97%     2 readings
2025-W45        59.0 bpm        99%     1 readings
2025-W46        63.0 bpm        97%     2 readings
```

Figure 4. Sample output of program



username	naka
password	123
name	Jappeth
age	21
height	175
weight	60
activity level	active
daily calorie goal	2500
remaining calories	2500
last reset date	13/11/2025

Date	RHR	O2 Lvl	BP
01/11/2025	60	98	110/70
02/11/2025	58	97	110/70
09/11/2025	59	99	110/70
14/11/2025	61	96	110/70
13/11/2025	65	98	120/80

Date	Food	Calories	MealType
01/11/2025	Rice	200	Dinner
01/11/2025	Water	0	Snack
01/11/2025		200	Dinner
04/11/2025	Rice	200	Dinner
04/11/2025	Water	0	Snack
04/11/2025		200	Dinner
07/11/2025	Rice	200	Dinner
07/11/2025	Water	0	Snack
07/11/2025		200	Dinner
10/11/2025	Rice	200	Dinner
10/11/2025	Water	0	Snack
10/11/2025		200	Dinner

Figure 5. Program file output

Conclusion

The MedTrack system was successfully developed as a functional tool for personal health monitoring. Testing confirmed that the system reliably manages user accounts and securely logs daily nutrition and vital signs (Objectives 1 & 2). Although the weekly trend analysis (Objective 3) could not be fully validated with long-term data, its code implementation is complete and ready for demonstration. The system's effective use of clinical data confirms its strong potential for real-world application, providing a solid foundation for promoting health awareness and proactive wellness management. Future work should focus on long-term testing and enhanced data visualization to fully validate its potential.

References

Ismail, L., Abul Hassan, M., AlHosani, A., et al. (2020). Requirements of health data management systems for biomedical care and research: Scoping review. Journal of Medical Internet Research, 22(7). https://doi.org/10.2196/17508

Ragupathi, T., Naveen Kumar, A., & Prasanna, S. (2022). Health monitoring system based on IoT. International Journal of Health Sciences, 6(S2). https://doi.org/10.53730/ijhs.v6nS2.7581

Coman, L.-I., Ianculescu, M., Paraschiv, E.-A., Alexandru, A., & Bădărău, I.-A. (2024). Smart Solutions for Diet-Related Disease Management: Connected Care, Remote Health Monitoring Systems, and Integrated Insights for Advanced Evaluation. *Applied Sciences*, *14*(6), 2351. https://doi.org/10.3390/app14062351

World Health Organization (2025) Noncommunicable diseases https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases

Iwaya, L. H., Ahmad, A., & Babar, M. A. (2020). Security and Privacy for MHealth and uHealth Systems: A Systematic Mapping study. IEEE Access, 8, 150081–150112. https://doi.org/10.1109/access.2020.3015962