National College of Ireland

<BSHCSD4>

<Software Development>

<Academic Year 2022/2023>

<James Butler>

<x20129211>

<x20129211@student.ncirl.ie>

<BetterU>

Technical Report

Contents

Executive Summary

BetterU is an application designed at helping you achieve a better you. To become the person, you have always wanted to be and to hold yourself accountable. This report is to document the functionality of BetterU and how through using it can help a user achieve their goals.

This report goes over everything to do with BetterU from the Requirements to the implementation and the testing. This report fully outlines all aspects of BetterU and its capabilities.

1.0 Introduction

1.1. Background

I have chosen to undertake this project as it is on a topic that has a particular interest to me. The topic is self-improvement. I believe the journey of self-improvement is a very important journey to help become the person you want to be. I want this application to be able to facilitate people in their self-improvement journeys to become better versions of themselves. That's where the name BetterU comes from. It's a play on 'Better You'. I picked this name as the goal of this application is to help create a better you. I think self-improvement has been gaining traction in recent times and I would like to facilitate others in their journeys.

1.2. Aims

This project aims to create an application that will help the end user to be able to stay on track on their journey of self-improvement. It is for people of all ages. It helps users be able to be able to make better decisions regarding their weight and their habits and make it easier to see their progression. BetterU has features to be able to see a user's progression and help the user stay on track. The ability to get a motivational quote will help the motivation of the user. The application is intuitive so anyone can navigate it easy.

1.3. Technology

BetterU is built using NetBeans and is made using the Java programming language. I also used SQL in creating BetterU to be able to interact with the database for the storage of information such as the user accounts and the information that can be inserted into BetterU.

I had originally set out to create this program in android studio and make it as an android application but due to limited resources of my computer and the resource heavy nature of android studio I had to switch to creating this project in the NetBeans IDE. I chose to stick with Java as it is the language, I had originally set out to use with android studio.

1.4. Structure

This report starts off with section 1. This section contains the Introduction, The Aims, Technology and this the structure of the report. These subsections help the reader be able become familiar with the project idea and lay the foundation for the further reading of the Technical Report.

2.0 System

2.1. Requirements Migration

This application was original set to be developed with Android studio and to be a mobile phone application. The reason this is not how it turned out to be is due to the heavy use of resource nature of Android studio. I am running an old machine and it could not keep up with Android studio. It made development near impossible as it was very slow to work with. I realised this and decided to tell my supervisor about the issue.

The solution that I came up with was to keep the application as a java project but use NetBeans instead to create a Java Swing application for the PC. I decided to do this as Java would have been the language, I used to create the Android Application on Android studio. This is why you may notice that my project is proposed as an Android project but came to fruition as a NetBeans PC project. I have chosen to use NetBeans as it is something I know my PC could run well enough to be able to create this project.

This change resulted in my non-functional requirements changing. My functional requirements have not changed. My non-functional requirements now reflect a PC program and not an Android Application.

2.2. Requirements

1. The application is easy to navigate.
2. The application should be able to handle a large user base.
3. The application can run on any pc system.
4. There is a database to store information from the user.
5. The application should be able to run smoothly.

2.2.1. Functional Requirements

1. The system must allow a user to register and be able to login to the system upon opening the program.
2. The application should then validate the users' credentials and provide an error message for invalid login details.
3. The system should have session management so users cannot see each other's information.
4. The System should have a main menu with all the features of the application on it.
5. The System should have a calorie tracker where a user can input their meals and the calorie content of them meals. The user can also update, delete, or print those meals.
6. The system should have a BMI calculator to calculate the users BMI and view previous calculations.
7. The system should have a habit tracker to allow a user to be able to record their habits.
8. The system should have a calorie calculator to calculate how much calories needed for the user and view previous calculations.

9. The system should allow a user to set reminders to do various activities and see history of reminders.
10. The system should allow a user to be able to pull in a quote from an API. Quote should be saved to database and user be able to view previous quotes.
11. The system should generate stats, suggestions and graphs with the information entered into the other functions of the application.
12. The system should have a change password function to allow a user to change their password.
13. The system should allow a user to log out.

2.2.1.1. Use Case Diagram

2.2.1.2. Requirement 1 User Registration and Login

2.2.1.3. Description & Priority

This is where the user upon opening the program has the option to login and or register if they do not already have an account. This is number 1 priority as it is the door to the application. It is the first thing a user is met with upon opening the program. It is essential for the overall system as it provides the user with an account to be able to use the program and for the data that they enter the system to be associated with that username. This gives this requirement the utmost priority.

2.2.1.4. Use Case

Requirement Number 1 on List of Functional Requirements.

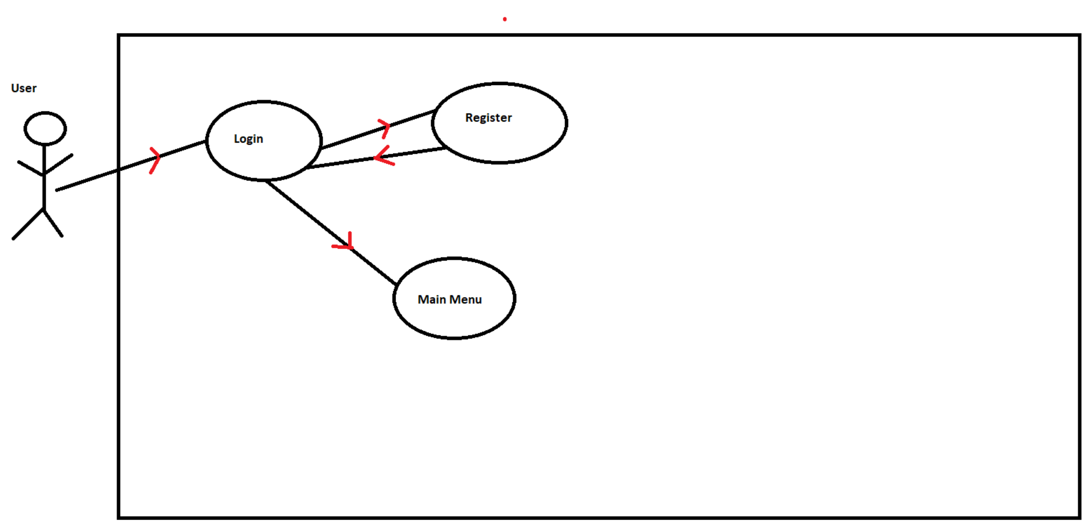
Scope

The scope of this use case is to allow a user to be able to login to the system or else to be able to create a user on the system by using the register window. The user registration asks for a username and to type the password twice to confirm the password before it is submitted to the database. The user can then go back to the login screen and login.

Description

This use case describes when a user first opens the program. They have the options of being able to register or login. If the user does not have an account they must register and if they do have an account they can login to the system.

Use Case Diagram



Flow Description

Precondition

The user has BetterU and is ready to open it.

Activation

This use case starts when the user opens up BetterU

Main flow

1. The user opens the application.
2. Upon opening the application there is a login screen with fields to enter a username and password. There is also a button to register.
3. The user enters their username and password and presses login.
4. The system checks the database to verify the user's credentials.
5. If the credentials are valid the system creates a session for the user and then grants the user access
6. The user is met with the main menu.

Alternate flow

A1:

1. The user opens the application.
2. The login screen is shown with input for the username and password and a button to register.
3. The user then clicks on the register button and the registration form opens.
4. The system then shows a registration form with input for a username, password and then to re type the password.
5. The user fills out the information needed and submits the registration form.

6. The system then makes sure that the username selected is unique. If not, they are prompted to try again. If the passwords do not match the user is also prompted to try again.
7. If the validation is successful a new user account is created, and the information is stored in the database.
8. The user can now go back to the login page.
9. The use case continues at step one of the main flow.

Exceptional flow

E1: Exceptional Flow (Registration – Username Taken or Passwords Don't match)

1. User opens the application.
2. The user clicks on registration.
3. The user enters the required information and presses register.
4. The system makes sure that the username is unique and that the passwords match.
5. If the username is taken the system displays a message that the username has already been taken.
6. If the passwords do not match the user is prompted by the system saying the passwords do not match.
7. The user can ensure the passwords match.
8. Once the entered information passes the validation the account is now created for the user
9. The use case can continue from step one in the main flow.

E2: Exceptional Flow (Login – Incorrect Details)

1. User opens the application.
2. The application presents a login screen with fields for the username and password.
3. User enters their details and presses the submit button.
4. The system verifies the details with the information that is in the database.
5. If the entered information is not correct the system prompts the user saying the information is not correct.
6. User is prompted to try again.
7. User can retry the login process.
8. Use case continues from step 1 of the main flow.

Termination

User presses login and is granted access to the account.

Post condition

The user is then presented with the main menu.

List further functional requirements here, using the same structure as for Requirement1.

2.2.1.5. Requirement 2 Validate Users Credentials

2.2.1.6. Description & Priority

This is one of the most crucial aspects of the application. The application must verify the user's credentials before giving the user access to the system. If the credentials are incorrect the system, the system shows a dialogue box to tell the user to try again. This prevents unauthorised access to the system. This has very high priority as without a proper validation system the information in the system could be seen by someone who is not supposed to. Having proper validation also ensures that the system has adequate usability and could affect the whole user experience.

2.2.1.7. Use Case

Requirement Number 2 on List of Functional Requirements.

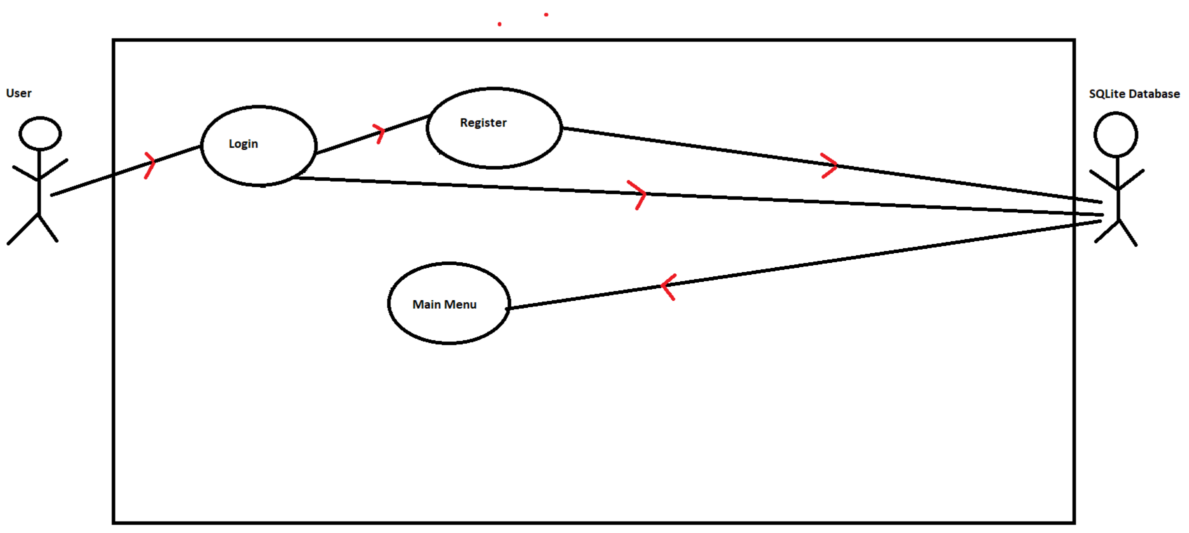
Scope

This use case focuses on the verification of the users' credentials this consists of a username and a password. This ensures a user is who they say they are. The system validates the credentials against the credentials stored in the SQLite database. If the credentials do not match the system returns an error message telling the user that they have entered invalid credentials and to try again. For registration there should be validation to ensure that username is unique and that the password and the re-typed password match.

Description

This use case describes when a user enters their details into the system checks with the database that they have an account. If the user is registering the uniqueness of the username is to be checked and that the passwords typed match each other.

Use Case Diagram



Flow Description

Precondition

The Login page open and ready to be used.

Activation

Login

The user starts the login process by entering their login credentials into the username and password fields. The user then submits the credentials for validation by pressing the login button.

Registration

From the Login page the user clicks on the register button. Once the registration form is open the user then enters the required information into the username and 2 password fields. The user then submits this information by pressing the register button.

Main flow

1. User opens the login page.
2. User enters their login credentials.
3. The user clicks the login button.
4. The system validates the entered details.
5. If the details are valid the user is brought to the main menu of the application.
6. If the credentials are not valid the user is prompted with an error message telling the user to try again.

Registration

1. The user opens the login page and clicks registration.
2. The user enters the required information (Username, Password, Re-type Password)
3. The user submits the details entered by pressing the register button.
4. If the username is unique and the passwords match a new user is entered into the system. The user can then log into the system using these credentials.
5. If the information does not match an error message is displayed depending on if the username is already taken or else if the passwords don't match.

Alternate flow

A1: Invalid credentials During Login

1. User opens the login page a
2. If the credentials are invalid an error message is displayed to say try again.

A2: Invalid Registration Information

1. The user clicks on the register button.
2. The user enters their desired username and password and is made to re-type the password.
3. If username is taken the system prompts the user that the username has already been taken
4. If the passwords don't match the system tells the user that the passwords don't match.

Exceptional flow

E1: Exceptional Flow (Validation Errors during Registration or Login)

1. If there are errors when entering login or registration info for example blank fields the program prompts the user that the fields cannot be left empty.

Termination

User presses login and is granted access to the account.

Post condition

The user is then presented with the main menu.

2.2.1.8. Requirement 3 Session Management

2.2.1.9. Description & Priority

This use case is also very important as it sets the user of the systems session so they can only see their own information and not anyone else's. It isolates the user from all the other users to ensure data privacy. For example, a user cannot see another users calories or statistics. The whole point of this is to ensure data privacy.

2.2.1.10. Use Case

Requirement Number 3 on List of Functional Requirements.

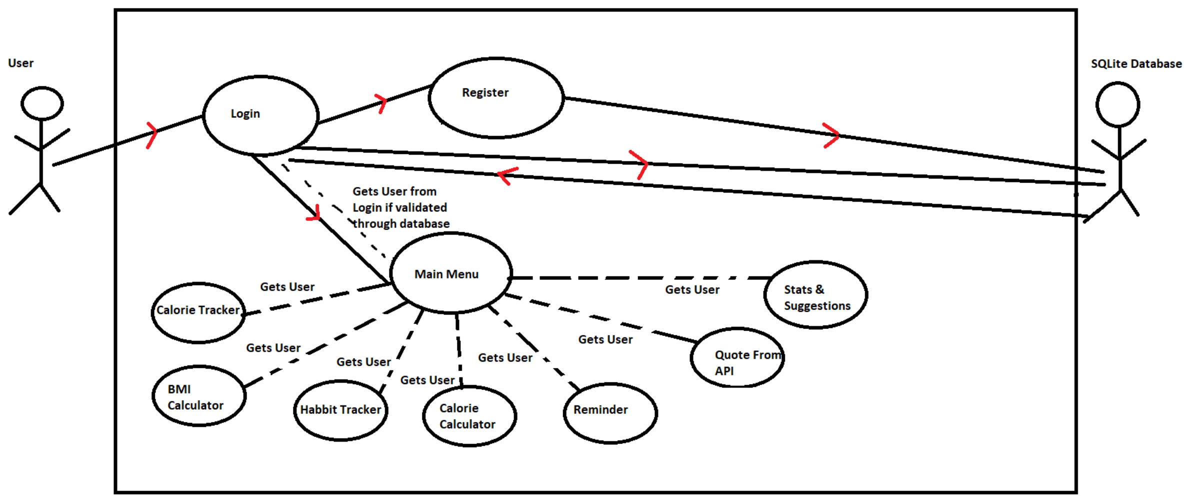
Scope

The scope for session management focuses on being able to establish a session for each user that logs in. This is to ensure privacy of data and isolation within the system. Some of the functionalities included in this are the authentication of a user, session initialisation and identification. Session management in this system ensures that that users can only access and interact with their own information. This keeps the system secure.

Description

Session management in this system is very important. It works by passing the username of the user into the main menu upon successful login into the system. The user is assigned to the variable LoggedInUser which is passed onto all the other functionalities of the system upon opening them. The system constantly knows who is logged in and this creates the session management.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system.

Activation

User enters their credentials, and they are verified. Session management kicks in and creates a session for that user. While the user is logged in and the application is on the session remains active.

Main flow

1. User logs into the system with valid credentials
2. The system checks the users' credentials against the database and then grants access.
3. The system creates a session for the user.
4. User interacts with the system by accessing their information and modifying their information.
5. The system makes sure that a user can only see and interact with their own information.
6. User uses the system with their session.
7. The user logs out and the session closes.

Alternate flow

A1: Invalid credentials During Login

1. User enters invalid credentials during login.
2. System detects the invalid credentials.
3. The system shows an error message asking the user to try again.
4. User repeats process with valid credentials.
5. The main flow continues from step 2.

Exceptional flow

E1: User tries to see other user information.

1. User tries to access another user information.
2. The user opens one of the functions.
3. The user can only see their own information.

Termination

User presses the logout button or just closes the system.

Post condition

The user is back to the login screen.

2.2.1.11. Requirement 4 Main Menu

2.2.1.12. Description & Priority

The main menu serves as the central hub for my system. This is where users can access all the functionalities of my system. The main menu allows a user to choose and interact with the functions that are included in the system. Some of the features on the main menu is buttons to be able to open the Calorie Tracker, BMI calculator, Calorie Calculator, Habit tracker, Random Quote from API, Set Reminders, and stats & suggestions. The main menu is the central gateway of the application and provides users with a way to easily navigate the system. It is a very important core part of my system.

2.2.1.13. Use Case

Requirement Number 4 on List of Functional Requirements.

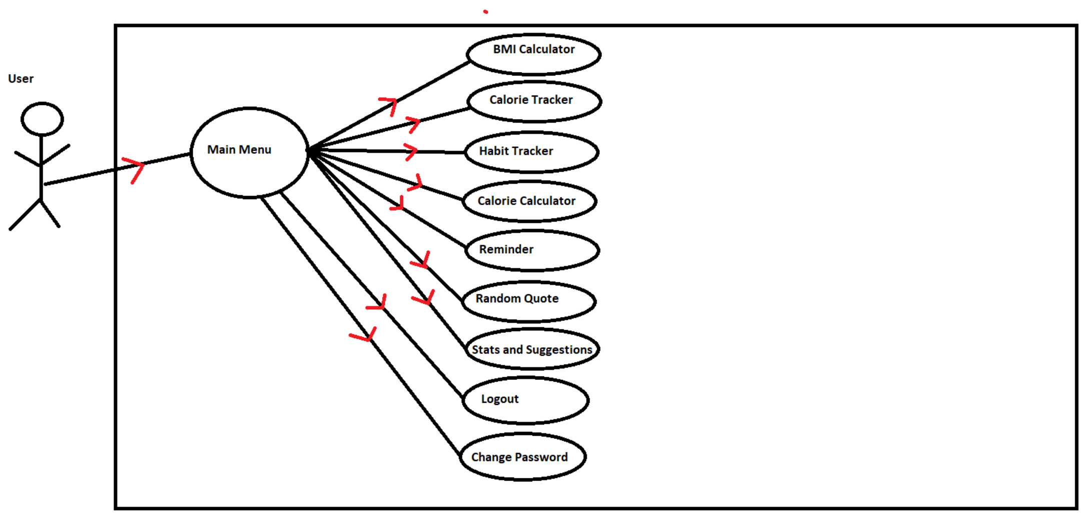
Scope

The scope for the main menu is for it to be easily used and understood by the user. It is the navigational epicentre of my application it is what is to be used by the user to open all the functionalities of my system.

Description

The main menu is where the user accesses all the functionality of my system. The main menu is also the first place the user is passed from the login screen. It is in the main menu class that the user gets assigned to the LoggedInUser variable, so the main menu is utmost importance. The main menu also passes the LoggedInUser to all the other functions.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system and gained access to the menu. This means that the user has been authenticated and the system has created a session for the user and the user is at the main menu.

Activation

User enters their credentials, and they are verified. Session management kicks in and creates a session for that user. User is then brought to the main menu.

Main flow

1. The user logs into the system successfully.
2. The system checks to make sure the users' credentials are correct.
3. The system then activates the main menu.
4. User selects a function from the main menu.
5. System opens the function.
6. User interacts with the feature.
7. After using the feature, the user can return to the main menu.

Termination

User presses the logout button and returns to the login screen or else the user closes the main menu window down which closes the whole application. Both these options end the user's session.

Post condition

The user is back to the login screen, or the system is fully closed altogether.

2.2.1.14. Requirement 5 Calorie Tracker

2.2.1.15. Description & Priority

This use case is for the calorie tracker function of my system. The calorie tracker allows a user to be able to track their meals. The calorie tracker takes in the user's meal, food, calories, and date of the record. The functionality is in the form of a crud format. This is an effective means for managing meal entries. Users can view their meals in the table in the calorie tracker. This functionality is high priority as it is one of the core parts of this system.

2.2.1.16. Use Case

Requirement Number 5 on List of Functional Requirements.

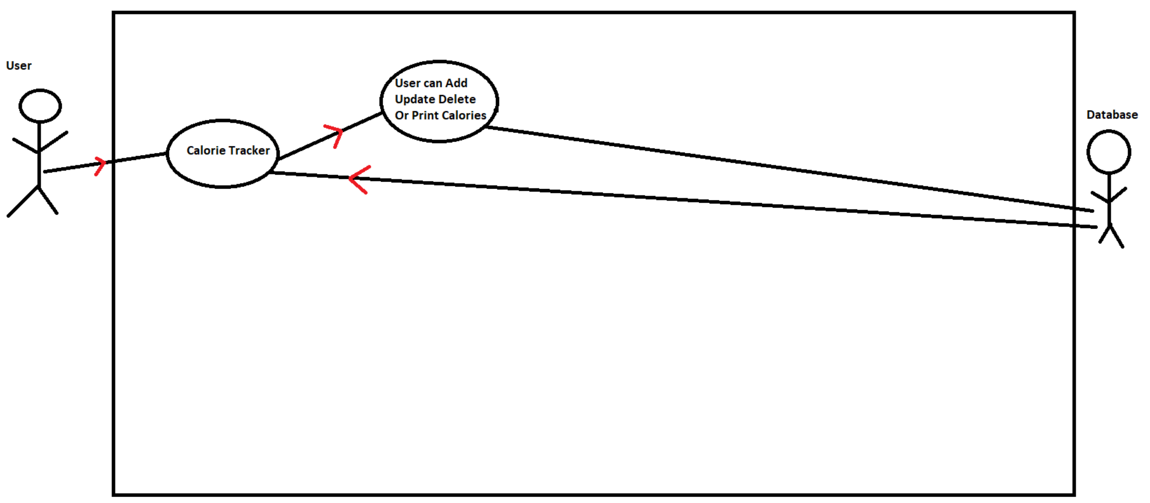
Scope

The scope for this requirement is a calorie tracker that allows a user to be able to input their meal, food, calories, and date. It should have functionality to update, delete and print the meal entries. It should only show the user their calories.

Description

This function allows a user to be able to see their calories consumed and help the user be able to hold themselves accountable for what they put into their body. The calorie tracker consists of 4 entries for the user. These entries are the Meal the Food the Calories and the Date. The user can see what is inputted into the system in the table in the calorie tracker.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the calorie tracker.

Activation

The activation is started when the user opens the calorie tracker from the main menu and then begins the process of inputting managing and interacting with the entries in the tracker.

Main flow

1. The user opens the calorie tracker feature.
2. The user fills in the information for their meal.
3. The user presses the add data button.
4. The system saves the meal entry to the database.
5. The system updates the table to show the users entered meal.
6. The user has options to update delete or print the entry.
7. If the user uses update they are able to modify the selected entry and then press the button to save it to the database.

8. If the user presses the delete button the selected entry is deleted from the database.
9. If the user wants to print the system presents a print screen where a user can select their printer and print
10. The user can navigate back to the main menu.

Alternate flow

A1: User chooses to not input a new entry. Instead, the user wants to view previous meals.

1. The system retrieves and shows the previous entries in the table.
2. The user can select a meal entry and can update delete or print it.

Exceptional flow

E1: User tries to delete or update without selecting an entry.

1. If the user selects to update or delete an entry that does not exist, the system shows an error to the user.

Termination

User presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.17. Requirement 6 BMI Calculator

2.2.1.18. Description & Priority

This use case is for the BMI calculator function of my system. The BMI Calculator allows a user to be able to calculate their BMI based on their weight, height age and gender. The feature also provides a classification of their BMI. For example, Overweight or Normal Weight. The function also tells a user their ideal weight. A user can also view previous BMI calculations. The user has the option to delete these previous calculations or else print them. This is a core feature of my application, and it has a high priority.

2.2.1.19. Use Case

Requirement Number 6 on List of Functional Requirements.

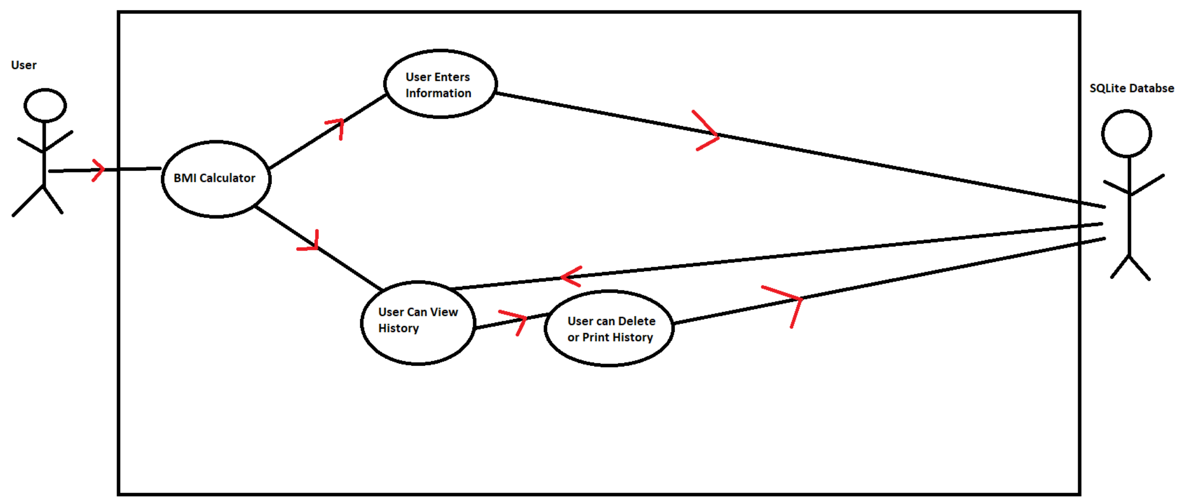
Scope

The scope of this use case is that the feature should be able to calculate the users BMI based on the input values of weight height gender and age. The BMI calculator should give the user a classification as well as the ideal weight the user should be. The user can also view previous calculations as they are stored in the database. The user can delete the previously saved calculations and they can also print them. A user will be only able to see their own data.

Description

This function allows a user to calculate their BMI based on their Age, Gender Height, and Weight. In the use case diagram below we can see that the user interacts with the BMI Calculator the user then enters their information and generates their BMI this all gets stored in the database. The user can view the history of the previous calculations. This page for viewing previous calculations is populated from the database the user can the delete or print their history. If they delete the history, it is deleted from the database.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the BMI Calculator.

Activation

The activation is started when the user opens the BMI Calculator from the main menu and then begins the process of inputting, managing, and interacting with the entries.

Main flow

1. The user opens the BMI calculator.
2. The user enters in their Weight, Height, Gender, and Age.
3. The user then clicks the calculate button.

4. The function calculates the BMI based on the entered values and displays a result along with a BMI classification and the ideal weight of someone with the entered parameters.
5. The BMI, classification, and Ideal weight along with the input values are stored to the database.
6. The user can press the clear button to reset the input fields for a new calculation.
7. The user can press the view button to view previous BMI calculations.

Alternate flow

A1: User chooses to not calculate BMI.

1. In step 3 of the main flow the user instead of pressing calculate decides to press the clear button instead as they do not want to calculate the BMI

Exceptional flow

E1: User tries to enter unrealistic values for age weight or height.

1. If the user enters a value that is unrealistic for the age, weight, or height the function will display a message to the user telling them to enter a realistic value.

Termination

User is done with the BMI calculator and presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.20. Requirement 7 Habit Tracker

2.2.1.21. Description & Priority

This use case is for the habit tracker function of my system. The habit tracker allows a user to be able to track their habits. The habit tracker takes in the user's habit name, Frequency, Goal, Progress, Notes, and date of the record. The functionality is in the form of a crud format. This is an effective means for managing habit entries. Users can view their habits in the table in the habit tracker. This functionality is high priority as it is one of the core parts of this system.

2.2.1.22. Use Case

Requirement Number 7 on List of Functional Requirements.

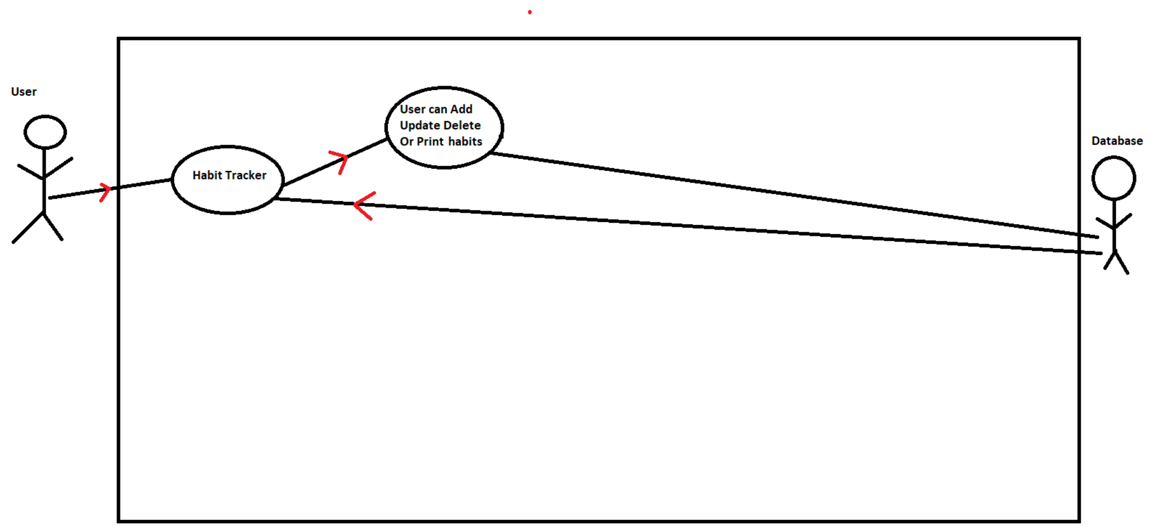
Scope

The scope for this requirement is a habit tracker that allows a user to be able to input their habit, frequency of habit, goal, progress, notes, and date. It should have functionality to update, delete and print the habit entries. It should only show the user their habits.

Description

In the below use case diagram, we can see that the user interacts with the habit tracker they can add, update, delete or print their habits. If a user updates, adds or deletes an entry it modifies the database, and the information is sent back to the Habit Tracker table which is updated.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the habit tracker.

Activation

The activation is started when the user opens the habit tracker from the main menu and then begins the process of inputting managing and interacting with the entries in the tracker.

Main flow

1. The user opens the habit tracker feature.
2. The user fills in the information for their habit.
   The user presses the add data button.
3. The system saves the meal entry to the database.

4. The system updates the table to show the users entered habit.
5. The user has options to update delete or print the entry.
6. If the user uses update, they can modify the selected entry and then press the button to save it to the database.
7. If the user presses the delete button the selected entry is deleted from the database.
8. If the user wants to print the system presents a print screen where a user can select their printer and print
9. The user can navigate back to the main menu.

Alternate flow

A1: User chooses to not input a new entry. Instead, the user wants to view previous habits.

3. The system retrieves and shows the previous entries in the table.
4. The user can select a habit entry and can update delete or print it.

Exceptional flow

E1: User tries to delete or update without selecting an entry.

2. If the user selects to update or delete an entry that does not exist, the system shows an error to the user.

Termination

User presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.23. Requirement 8 Calorie Calculator

2.2.1.24. Description & Priority

The calorie tracker feature allows a user to be able to determine how much calories they should consume in a day using factor such as age, height, weight, gender, and activity level. It also gives the user the ability to be able to manage and view their previous calculations. This helps users make informed decisions about their dietary control. This is a high priority as it is one of the core functions in this system.

Requirement Number 8 on List of Functional Requirements.

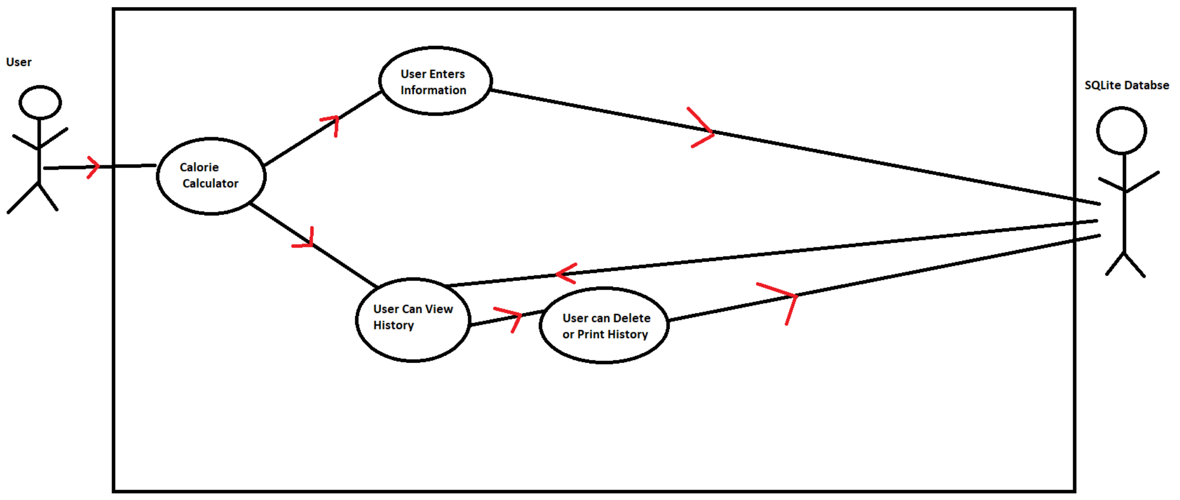
Scope

The scope for this requirement is a calorie calculator that allows a user to be able to calculate their recommended daily calorie intake based on their age, height, weight, activity level and gender. Users can view their previous calculations and delete specific entries and be able to print out a table of past results.

Description

In the below use case diagram, we can see that upon opening the calorie calculator the user has an option to view the history of previous calculations or they can enter information into the calorie calculator. Both options interact with the database. If the user enters information the entered information is sent to the database and if the user views information, they are pulling information from the database. When a user deletes a previous calculation, it is deleted from the database.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the Calorie Calculator.

Activation

The activation is started when the user opens the calorie calculator from the main menu.

Main flow

1. The user opens the calorie calculator.
2. The user enters their age weight height gender and activity level into the feature.
3. The user clicks the calculate button.

4. The system then calculates the recommended calorie intake based on this information.
5. The system then displays the result on a label.
6. The system then saves the input and calculation result to the database.
7. The user has the option to use the clear button which will reset the input fields.
8. The user can then click the view button which opens a new window which has a table populated with previous calculations done by the user and the input data that goes along with that data.

Alternate flow

A1: User presses the clear button.

1. The system clears all input fields allowing the user to input new values.

Exceptional flow

E1: User tries to enter unrealistic values for age weight or height.

1. If the user enters a value that is unrealistic for the age, weight, or height the function will display a message to the user telling them to enter a realistic value.

Termination

User presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.25. Requirement 9 Set Reminder

2.2.1.26. Description & Priority

The set reminder feature allows a user to be able to be able to create and manage reminders in this application. Users can enter the reminder text and select a date for the reminder from the calendar. The user can then use the two spinners to be able to set the time of the reminder. This feature allows a user to be able to set personalised reminders. This has a high priority as it plays a core function in my systems functionality.

Requirement Number 9 on List of Functional Requirements.

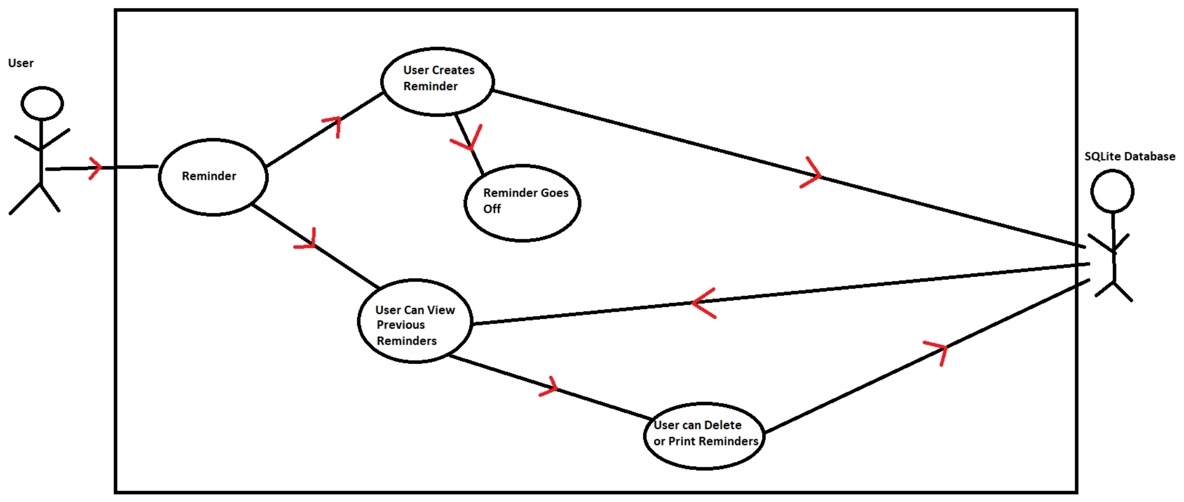
Scope

The scope of this feature is that it allows a user set reminder for various activities. The user can input the reminder text. They can also select a date and time for the reminder. The user can then view the history of previous reminders. The user can be able to delete specific reminders from the history. The user can also print off a list of reminders.

Description

In the below use case diagram, we can see that upon opening the reminder function the user has two options. They can create a reminder or else view previous reminders. If the user selects to view previous reminders, they are met with a table with the information from previous reminders this is populated by the database. From here a user can delete a selected entry or else print off the list of reminders. If the user wants to set a reminder, they full out the reminder text they select the date and then select the time for the reminder. When its time for the reminder to off the system will make a noise and alert the user of the reminder.

Use Case Diagram


User
User Creates Reminder
Reminder
Reminder Goes Off
SQLite Database
User Can View Previous Reminders
User can Delete or Print Reminders

Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the reminder option.

Activation

The activation is started when the user opens the reminder option from the main menu.

Main flow

1. The user opens the reminder functionality in the application.
2. The user enters the reminder text selects a date and specifies a time for the reminder to go off.
3. The user clicks the create reminder button.
4. The system then validates the input and creates the reminder with the selected details.

5. The system then creates the reminder with the specified details.
6. The system sets up the reminder to be triggered at the specified time.
7. The system then saves the reminder in the database.

Alternate flow

A1: User presses the view button.

1. The user clicks the view button.
2. The system then opens a window showing reminders that have been previously created.
3. User can review the reminder text date and the time.

Exceptional flow

E1: User presses the print button.

1. User presses the print button.
2. The system generates a printable document with all the reminders included in the table.
3. The system then sends the document to a printer.

Termination

User presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.27. Requirement 10 API Quote

2.2.1.28. Description & Priority

The Random quote feature allows a user to get a random quote from an API. The quote is presented to the user on a text field the quote is automatically saved to the database. The user can press the clear button if they would like to clear the text field. The user can press the view button to view previously saved quotes. In the view window the user can select quotes and delete them if they wish. The user can also print out the saved quotes.

Requirement Number 10 on List of Functional Requirements.

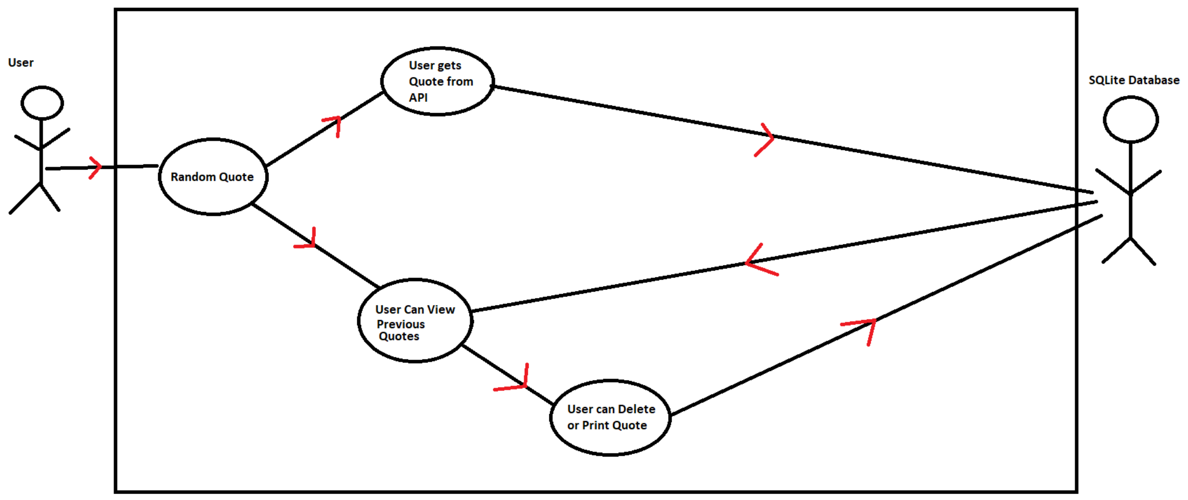
Scope

The system should allow a user to be able to get a random quote from an API. It should then save this quote to a database. After that the user can view saved quotes, delete a selected quote, and print the quote history.

Description

In the diagram below we can see that when a user opens up the program, they have two options they can either view previously generated quotes by the user or else generate a quote which will be saved by a database. If the user generates a quote, it is automatically saved to the database. The user can then click the view button and see any quote they have generated, and they can select the quote if they like and delete it. There is an option to print the quote as well. If the user deletes the quote the change is made to the database.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the random quote option.

Activation

The activation is started when the user opens the random quote option from the main menu.

Main flow

1. The user selects the random quote option from the main menu.
2. The system retrieves a random quote from the API.
3. The quote is displayed in a text field.
4. System saves quote to database.
5. User clicks view quotes option.
6. A new window opens that displays a table with previously saved quotes.
7. User closes the window.

Alternate flow

A1: Error making API request.

1. The user opens the random quote option from the main menu.

2. The system attempts to get a random quote from the API.
3. If there is an error in the API request the system displays a message saying error making API request.

Exceptional flow

E1: Error parsing JSON

1. The user opens the random quote option from the main menu.
2. The system attempts to get a random quote from the API.
3. If there is an error in parsing the JSON, the system displays a message saying error parsing JSON.

Termination

User presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.29. Requirement 11 Stats and Suggestions

2.2.1.30. Description & Priority

The stats and suggestions feature upon opening shows a bunch of stats and suggestion from all the other functions in the system. It shows these using SQL queries and nested SQL queries. This feature analyses and processes information that has been entered into other functions in this system. It retrieves data from the database and applies calculations and comparisons and shows the info to the user in meaningful way. This has a high priority as it generates significant insights for the user.

Requirement Number 11 on List of Functional Requirements.

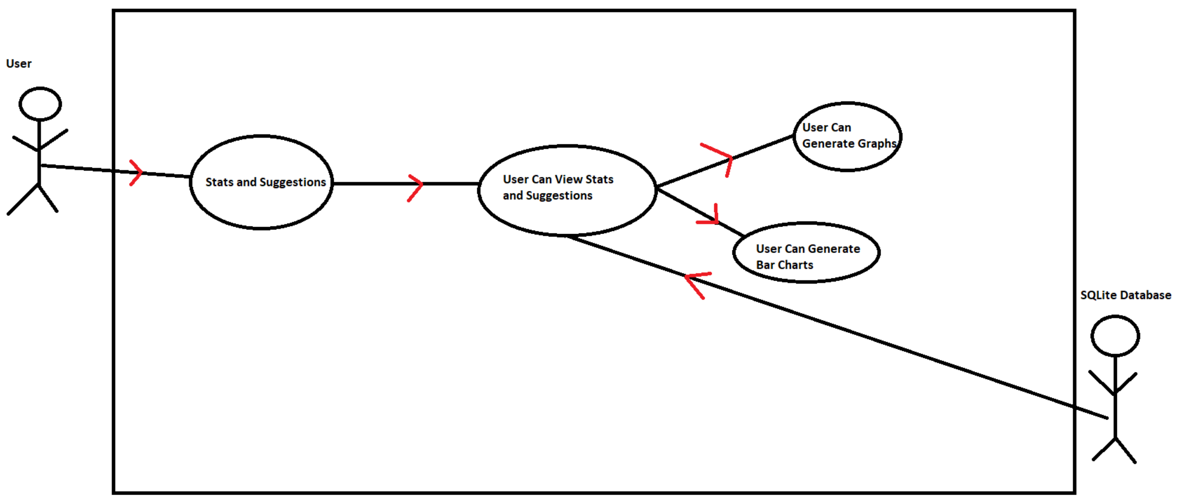
Scope

The scope of this use case is utilising existing data within the system to be able to provide insights recommendations and visual aids to the user. It is an integration of different data sources which applies the right statistical methods to be able to provide the user with results that are meaningful.

Description

In the diagram below we can see that upon the user opening the stats and suggestions feature the user is presented with some stats and statistics that is taken from the information entered into the rest of the features of the application in the database. The user can then generate some graphs based on the calorie consumption data or the BMI calculation data or else bar charts.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the stats and statistics.

Activation

The activation is started when the user opens the stats and statistics option from the main menu.

Main flow

1. The user selects stats and suggestions from the main menu.
2. The system gets the relevant data from the other functions in the application which are stored in the database.
3. The system then processes the data.
4. The system presents the stats and statistics to the user.
5. The user can then generate a graph for calorie consumption or else BMI data.
6. The user can also generate a bar for calorie consumption or else BMI data.
7. The user can go back to the main menu.

Alternate flow

A1: Not enough Data

1. If there isn't enough data for a statistic or suggestion the system will tell the user that there is a lack of data for that statistic or suggestion

Exceptional flow

E1: User tries to generate graph with only one calculation.

1. User presses generate graph button.
2. User is told there need to be more than one calculation for a graph to be generated.

Termination

User presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.31. Requirement 12 Change Password

2.2.1.32. Description & Priority

This use case allows a user to be able to change their password within the system. Upon the user selecting the change password option from the main menu a new window opens and presents the user with three fields one is for the current password the other two are for the new password and to re type the new password. The system preforms error checking on these fields to ensure that they are not blank and that the new password and the re type new passwords match. When the system verifies that the current password is correct, and the two new password fields match the password for the user is changed.

Requirement Number 12 on List of Functional Requirements.

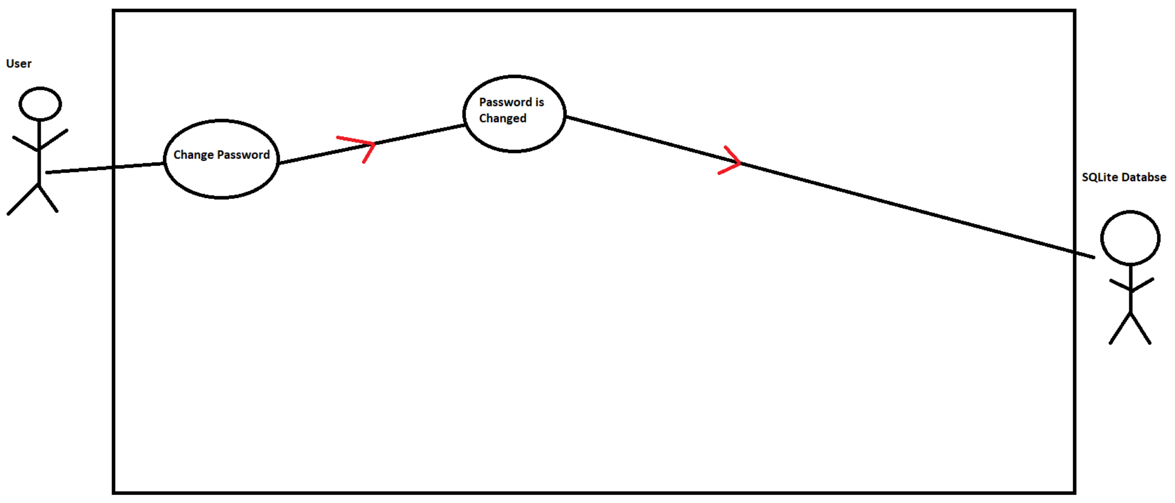
Scope

The scope of this use case is the functionality to be able to allow a user to change their password. It includes the error checking and database operation that are necessary to be able to perform this feature. The error handling ensures the current password inputs are correct and the new password entries are matching.

Description

This use case describes when a user changes their password using the change password feature. As seen below the user wants to change password. They then change the password which updates the database which has the user’s password stored.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to open the change password feature.

Activation

The activation is started when the user opens the option from the main menu.

Main flow

1. The user selects the change password option from the main menu.
2. The system opens a new window with three password fields these are current password new password and re-type new password.
3. The user enters their current password in the current password field.
4. The user enters their new desired password in the new password field.
5. The user re-enters the new password in the retype new password field.
6. User clicks the change password button.
7. The system verifies that the current password matches the user's current password in the database.
8. The system verifies that the new password and the retype new password match each other.
9. If both verifications pass the system updates the user's password with the new password
10. The system displays a message stating that the password has been updated.

Alternate flow

A1: If the current password field does not match the user's current password.

1. The system displays an error message indicating that the current password is not correct.
2. The user is prompted to try again.

Exceptional flow

E1: If the new password and the confirmed password do not match

1. The system displays an error message indicating that the passwords do not match.
2. The user is prompted to re-enter the new password and confirm the new password.

Termination

User presses the main menu button or just closes the window.

Post condition

The user is back to the main menu.

2.2.1.33. Requirement 13 Log Out

2.2.1.34. Description & Priority

This use case allows a user to log out when they press the logout button on the main menu.

Requirement Number 13 on List of Functional Requirements.

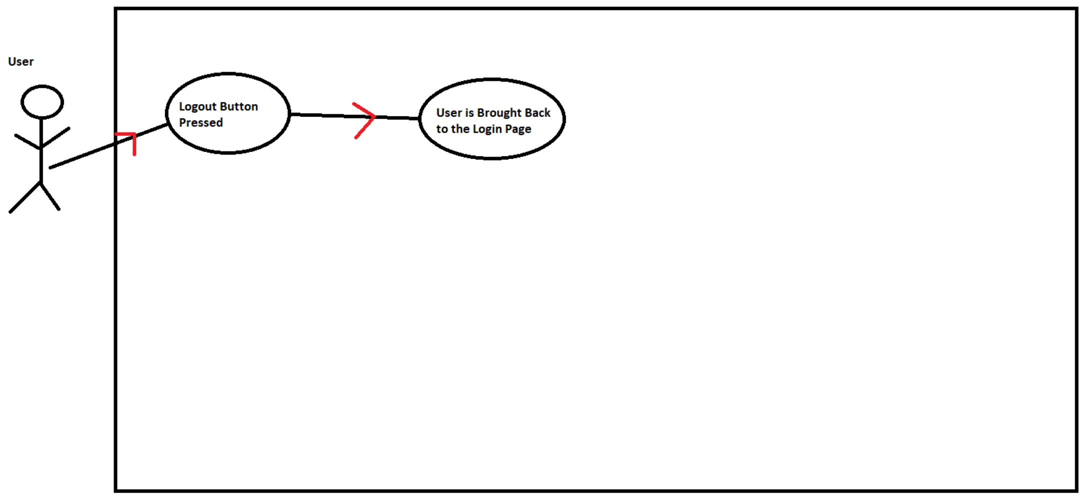
Scope

The scope of this use case is to log the user out when the logout button is pressed. The user is then brought to the login screen.

Description

This use case shows what happens when a user logs out. As seen in the diagram below the user is brought back to the log in screen upon pressing log out.

Use Case Diagram



Flow Description

Precondition

User has successfully logged into the system. On the main menu the user chooses to click the log out button.

Activation

The activation is started when the user presses the option from the main menu.

Main flow

1. User initiates the logout process by pressing the logout option from the main menu.
2. The system closes the users session.
3. The system redirects back to the login screen.
4. Login screen is displayed.

Post condition

The user is back to the Login Screen.

2.2.2. Data Requirements

For the data requirements of this system, I needed a database with quite a lot of tables to be able to perform the functions of the features of my application. I needed a table for Users, the BMI calculator, calorie calculator, calorie tracker, habit tracker, random quote and the create reminder features.

User must be able to retrieve the information stored in the database.

Database must be reliable.

User must be able to register and create username.

User must be able to create, read, update, and delete meal data.

User must be able to store and delete BMI data.

User must be able to store and delete reminder data.

User must be able to create, read, update, and delete data for habits.

User must be able to store and delete quote data.

User must be able to generate graphs on the BMI and calorie data.

2.2.3. User Requirements

User should be able to login and be able to register. The user is then taken to the main menu. The user can then choose one of the functions in the application. The functions are a calorie calculator, BMI calculator, calorie tracker, habit tracker, create reminder, get quote from API and stats and suggestions.

The user can use the features of this application and then view their stats and suggestions and create graphs.

A user must have a computer or suitable device that can use the JVM.

2.2.4. Environmental Requirements

This system uses the java JDK 1.8. To use this system the user must have a JVM installed on their machine. There are no operating system constraints. The dependencies used in this application are: JSON, okhttp, httpclient and sqlite-jdbc

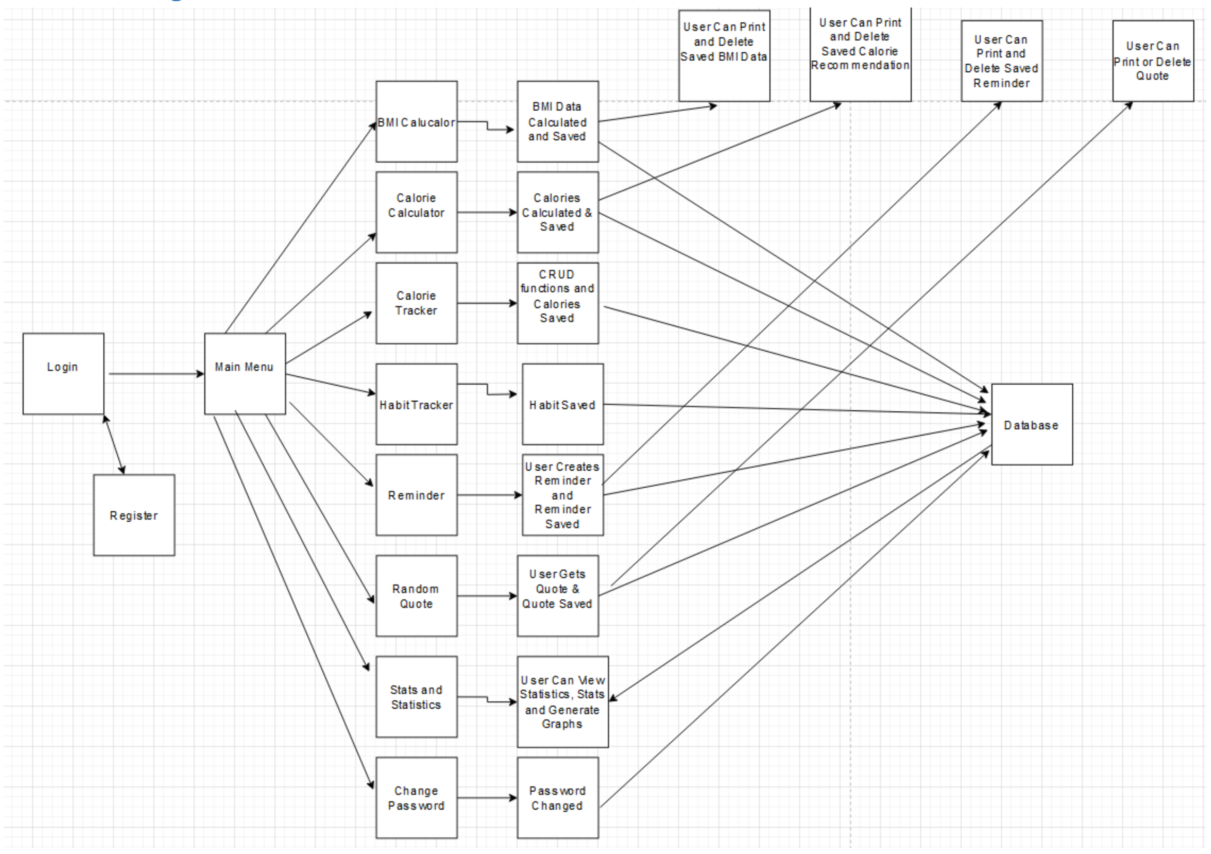
```
 <dependency>

     <groupId>org.json</groupId>

     <artifactId>json</artifactId>

     <version>20210307</version>

   </dependency>

</dependencies>


<dependency>

   <groupId>com.squareup.okhttp3</groupId>

   <artifactId>okhttp</artifactId>

   <version>4.9.1</version>

</dependency>
```

```
<dependency>
   <groupId>org.apache.httpcomponents</groupId>
 <artifactId>httpclient</artifactId>
   <version>4.5.13</version>
</dependency>

<dependency>
   <groupId>org.xerial</groupId>
   <artifactId>sqlite-jdbc</artifactId>
   <version>3.20.1</version>
</dependency>

<dependency>
   <groupId>org.jfree</groupId>
   <artifactId>jfreechart</artifactId>
   <version>1.5.4</version>
</dependency>
```

2.2.5. Usability Requirements

System must be easy to use and navigate. The system should be consistent in terms of speed and layout. The system should have adequate error handling. The system should be able to be used in a smooth manner with little to no loading times. The system should be compatible across many operating systems.

2.3. Design & Architecture



Login
Register
Main Menu
BMI Calucalor
Calorie Calculator
Calorie Tracker
Habit Tracker
Reminder
Random Quote
Stats and Statistics
Change Password
BMI Data Calculated and Saved
Calories Calculated & Saved
CRUD functions and Calories Saved
Habit Saved
User Creates Reminder and Reminder Saved
User Gets Quote & Quote Saved
User Can View Statistics, Stats and Generate Graphs
Password Changed
User Can Print and Delete Saved BMI Data
User Can Print and Delete Saved Calorie Recommendation
User Can Print and Delete Saved Reminder
User Can Print or Delete Quote
Database

The design and architecture of this project is based on a Java NetBeans application built on JDK 1.8. This system consists of a Java application that interacts with an SQLite database. To create this system, I used Java swing to be able to provide the user with an appropriate UI to be able to use the system intuitively.

The SQLite database is designed to be able to store information the user enters the program. I decided to go with an SQLite database as it was the one that suited my needs best. SQLite is lightweight and is self-contained. This made it a great chose for implementation into my project as there was no need for a database server. SQLite also provides other benefits such as the fact it is cross platform. It will work on Windows, Mac, Linux, and Android and IOS. SQLite is also very reliable and stable with a great proven track record.

After deciding to use SQLite I started to make my tables. Below are the tables that are used in my application.

Accounts Table:

```
CREATE TABLE Accounts (

   User VARCHAR(20),

   Pass VARCHAR(20),

);
```

This table is to store the user's information upon registering in the system. This table is also use for the login screen to allow a user to be logged in. Upon the user entering their details the system checks to see if their account exists in the database.

Calorie Tracker Table:

```
CREATE TABLE CalorieTrack (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   MealName VARCHAR(150),
   Food VARCHAR(150),
   Calories INTEGER NOT NULL,
   Date DATE,
   User VARCHAR(20) NOT NULL
);
```

This table is to store the information that is entered into the Calorie Tracker function of the system. The system stores the ID, which is auto generated. The meal name, the food, the calories, and the date. The user is determined by what user logged into the system initially.

Habit Tracker Table:

```
CREATE TABLE HabbitTrack (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   HabbitName VARCHAR(150),
   Frequency VARCHAR(150),
   Goal VARCHAR(150),
   Progress VARCHAR(150),
   Notes VARCHAR(150),
   Date DATE,
   User VARCHAR(20) NOT NULL
);
```

This table is used to store the information of the users' habits in the habit tracker function of the system. It stores the ID, which is automatically generated. The habit name, the frequency of the habit, the goal of the habit, the progress of the habit, the notes from the user and date are stored. The user is determined by what user logged into the system initially.

BMI Calculator Table

```
CREATE TABLE CalculatorBMI (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   Weight REAL,

   Height REAL,

   Age INTEGER NOT NULL,

   Gender VARCHAR(150),

   BMI REAL,

   Classification VARCHAR(250),

   Ideal_Weight REAL,

   Date DATE,

   User VARCHAR(20) NOT NULL

);
```

This table is used to store information from when a user uses the BMI Calculator part of the system. This table stores an ID for the record which is auto generated it also stores inputs from the user such as Weight, height, age, and gender. It then stores what the function generates based on this information. This is the BMI, classification, Ideal Weight. The date is automatically entered by the system. The user is determined by what user logged into the system initially.

Calorie Calculator Table:

```
CREATE TABLE CalorieCalculator (

   id INTEGER PRIMARY KEY AUTOINCREMENT,

   Age INTEGER NOT NULL,

   Weight REAL,

   Height REAL,

   Gender VARCHAR(150),

   ActivityLevel VARCHAR(150),

   Result REAL,

   Date DATE,

   User VARCHAR(20) NOT NULL

);
```

This table stores the information from the calorie calculator part of my system. The ID here is auto generated by the database. The user input is saved such as the age, weight, height, gender. The system then generates the result and automatically adds the date. The user is determined by what user logged into the system initially.

Random Quote Table:

```
CREATE TABLE RandomQuote (
    id INTEGER PRIMARY KEY AUTOINCREMENT,

    Quote VARCHAR(255),

    Author VARCHAR(100),

    Date DATE,

    User VARCHAR(20) NOT NULL

);
```

This table stores the random quote that is got from an API for a user. Upon clicking get quote the quote is got and saved to the database. The ID is auto generated by the database. The quote and author are got from the API parsed and added to the database. The date is added by the system. The user is determined by what user logged into the system initially.

Reminder Table

```
CREATE TABLE Reminder (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    ReminderText VARCHAR(255),

    DateReminder VARCHAR(100),

    TimeReminder REAL,

    Date DATE,

    User VARCHAR(20) NOT NULL

);
```

In this table the information enter by the user in the reminder feature of my system are saved. ID is auto generated by the database. Reminder text, date reminder and time of reminder are all inputs provided by the user. The date is added by the system automatically. The user is determined by what user logged into the system initially.

The application also uses an external API to be able to get a random quote for the user. This works by using the okhttp dependency and the json dependency.

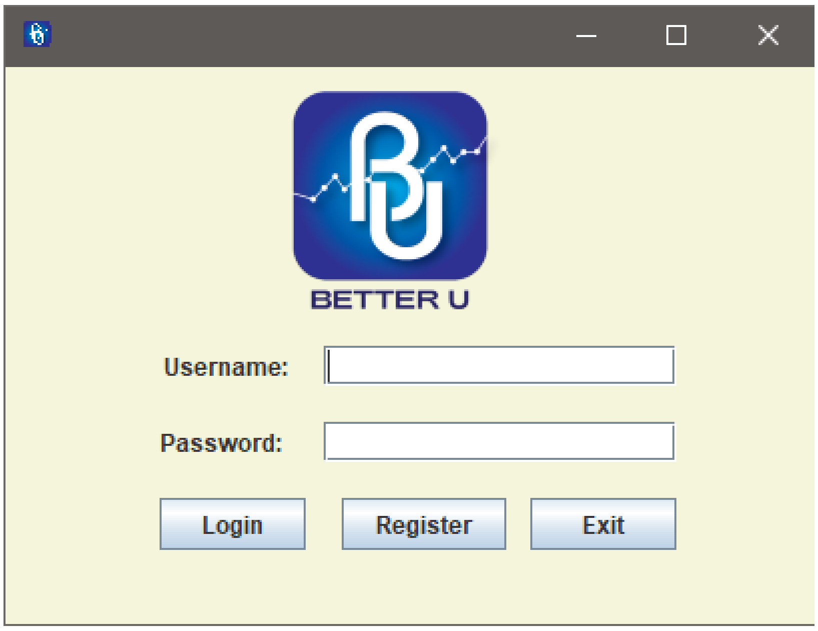
The charts in this system were generated by using the JfreeChart dependency.

2.4. Implementation

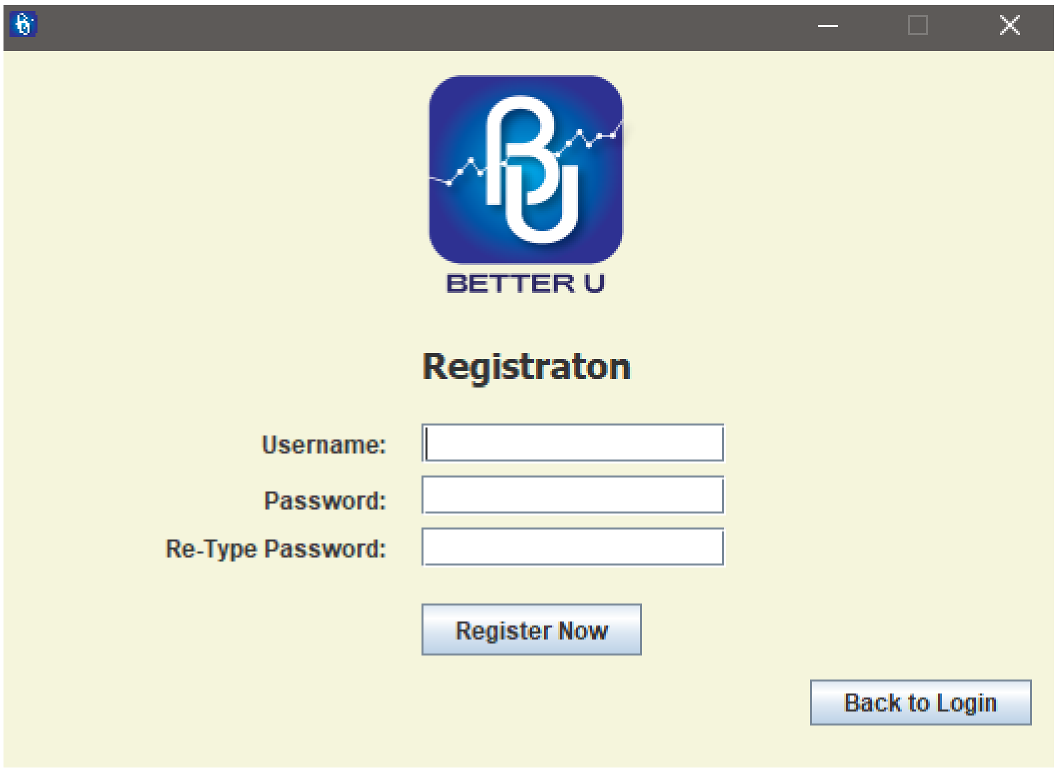
Describe the main algorithms/classes/functions used in the code. Consider to show and explain interesting code snippets where appropriate.

For this section, the implementation, I am going to go through the code and the functionalities of my system and explain each of them. I will also highlight the algorithms used when it comes to the functions of this project too. I will start from the beginning and work my way through the application.

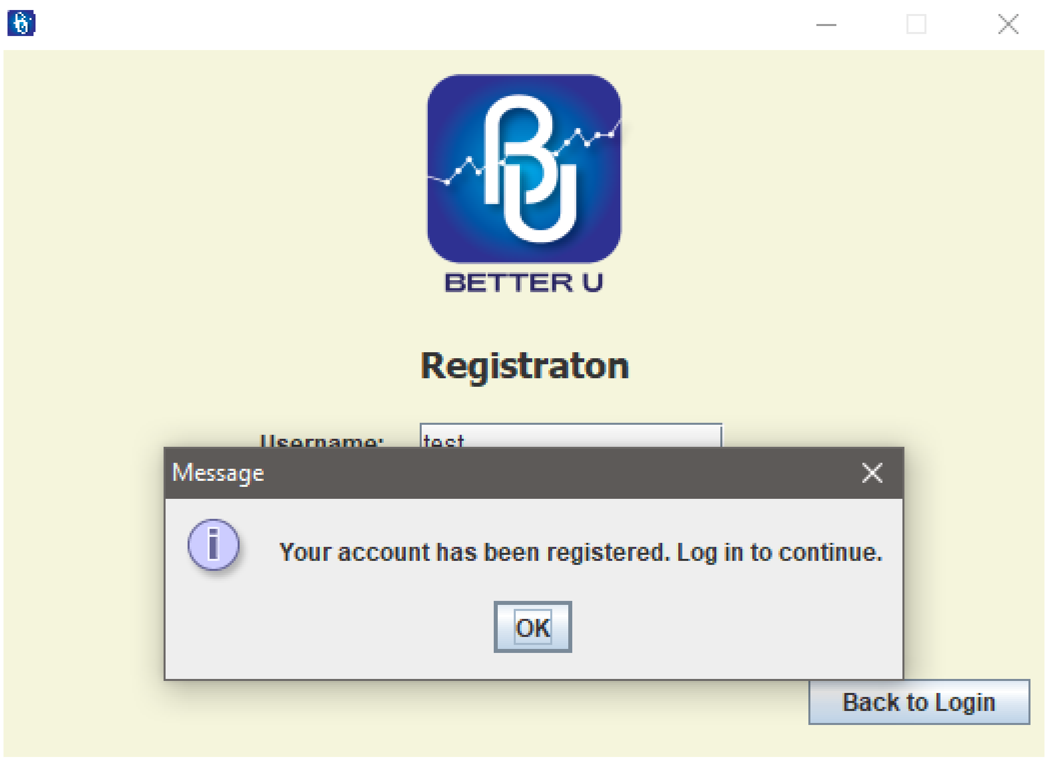
Upon opening the application, the user is met with a login page. Here a user has the option to log into the system or else to register themselves.



For demonstration purposes I will now register a new account. To do so the user would click the register button to be brought to the registration window.

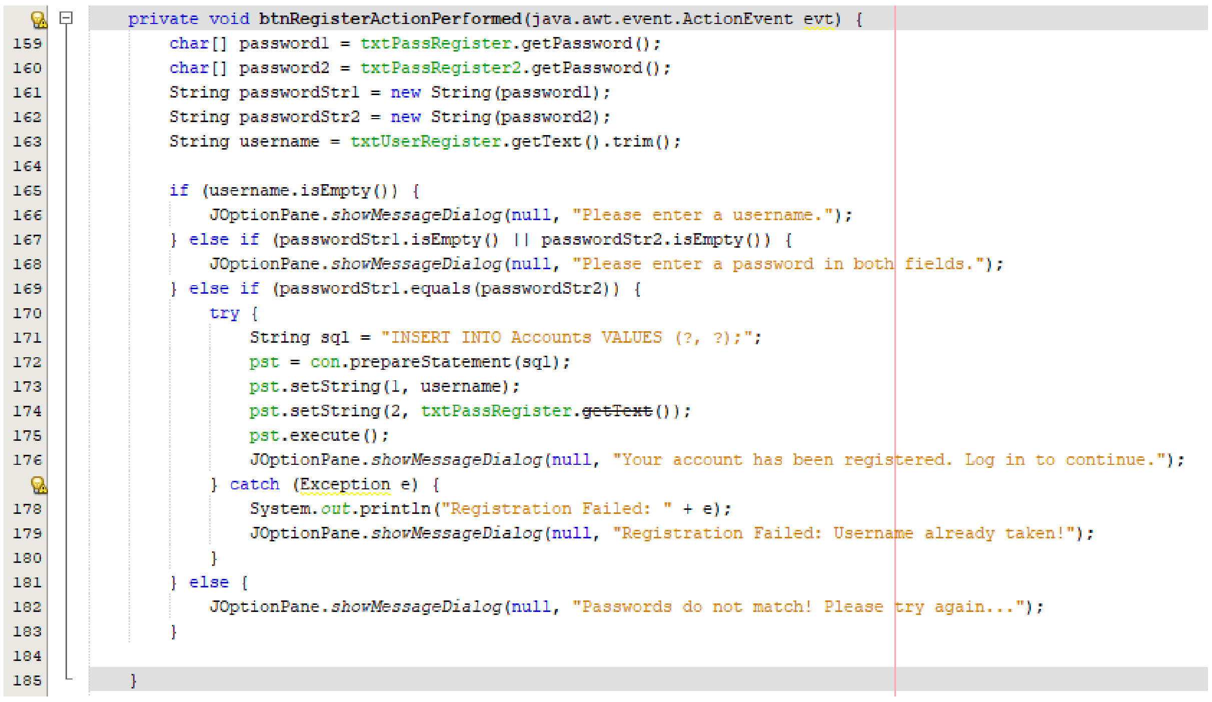


Here a user has the option to be able to create a username and a password I will begin my filling out this info and pressing register.



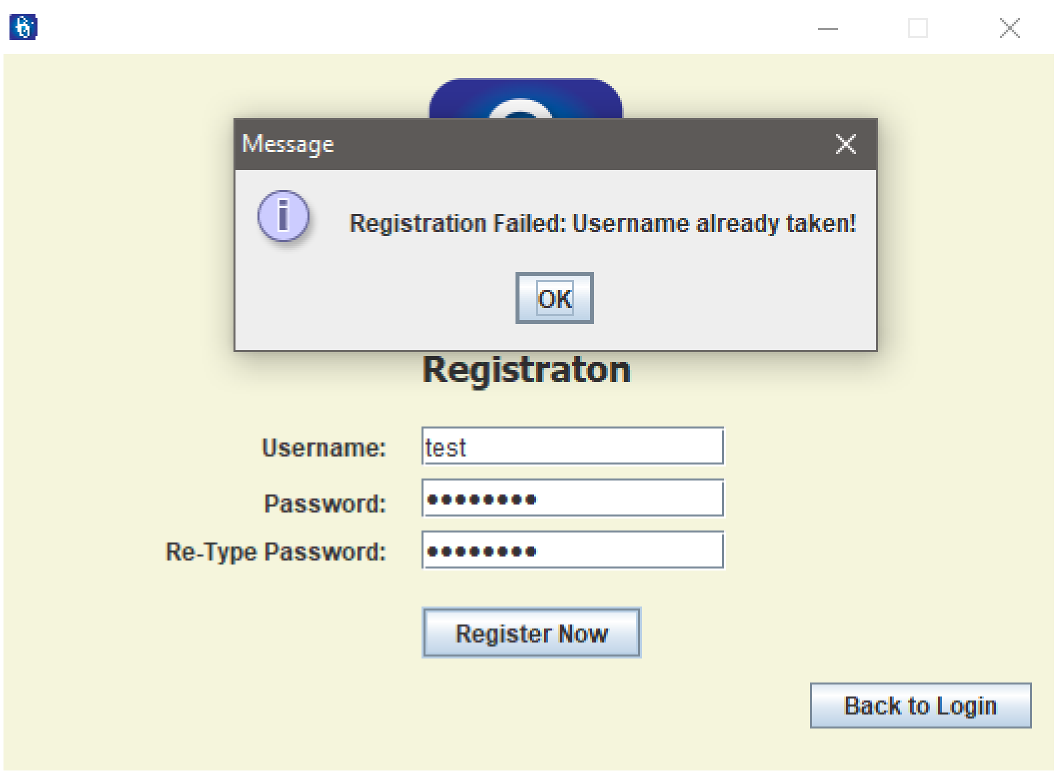
Your account has been registered. Log in to continue.

The user can now login to their newly created account. Below is the code for this registration function.



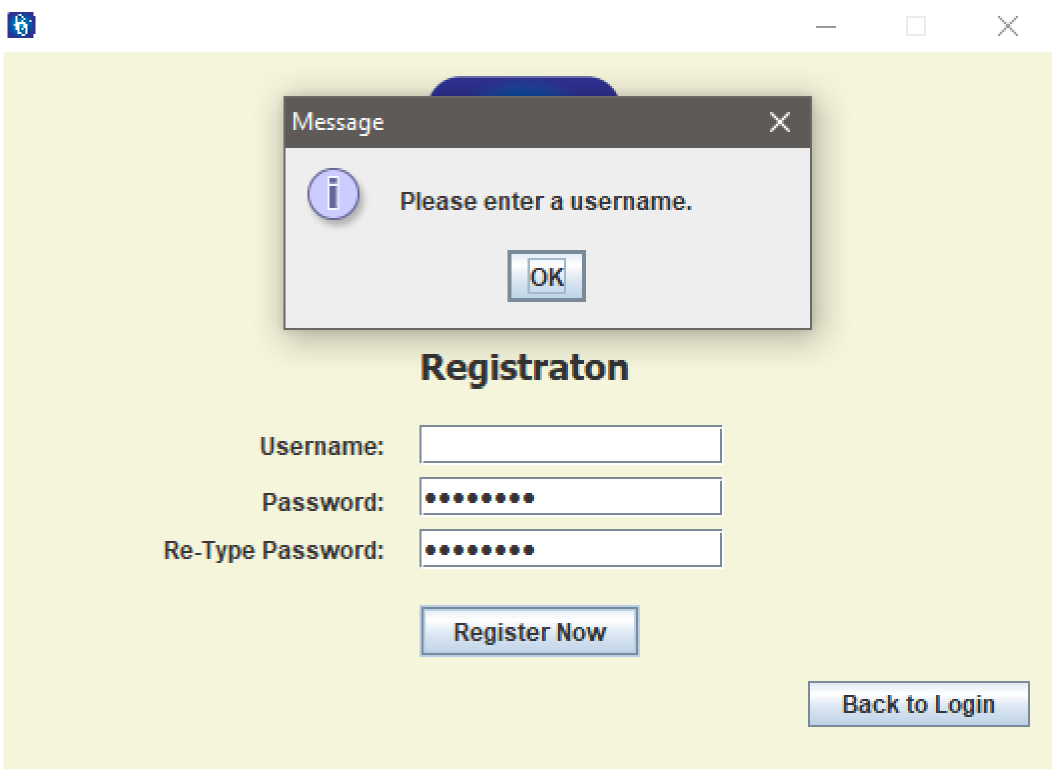
```
    private void btnRegisterActionPerformed(java.awt.event.ActionEvent evt) {
159     char[] password1 = txtPassRegister.getPassword();
160     char[] password2 = txtPassRegister2.getPassword();
161     String passwordStr1 = new String(password1);
162     String passwordStr2 = new String(password2);
163     String username = txtUserRegister.getText().trim();
164 
165     if (username.isEmpty()) {
166         JOptionPane.showMessageDialog(null, "Please enter a username.");
167     } else if (passwordStr1.isEmpty() || passwordStr2.isEmpty()) {
168         JOptionPane.showMessageDialog(null, "Please enter a password in both fields.");
169     } else if (passwordStr1.equals(passwordStr2)) {
170         try {
171             String sql = "INSERT INTO Accounts VALUES (?, ?);";
172             pst = con.prepareStatement(sql);
173             pst.setString(1, username);
174             pst.setString(2, txtPassRegister.~~getText~~());
175             pst.execute();
176             JOptionPane.showMessageDialog(null, "Your account has been registered. Log in to continue.");
            } catch (Exception e) {
178             System.out.println("Registration Failed: " + e);
179             JOptionPane.showMessageDialog(null, "Registration Failed: Username already taken!");
180         }
181     } else {
182         JOptionPane.showMessageDialog(null, "Passwords do not match! Please try again...");
183     }
184 
185 }
```

As seen in the above screenshot if the user passes the error checking their account information is inserted into the SQLite database it is done through the code and sql statement on line 156. I will now demonstrate the error handling in this.

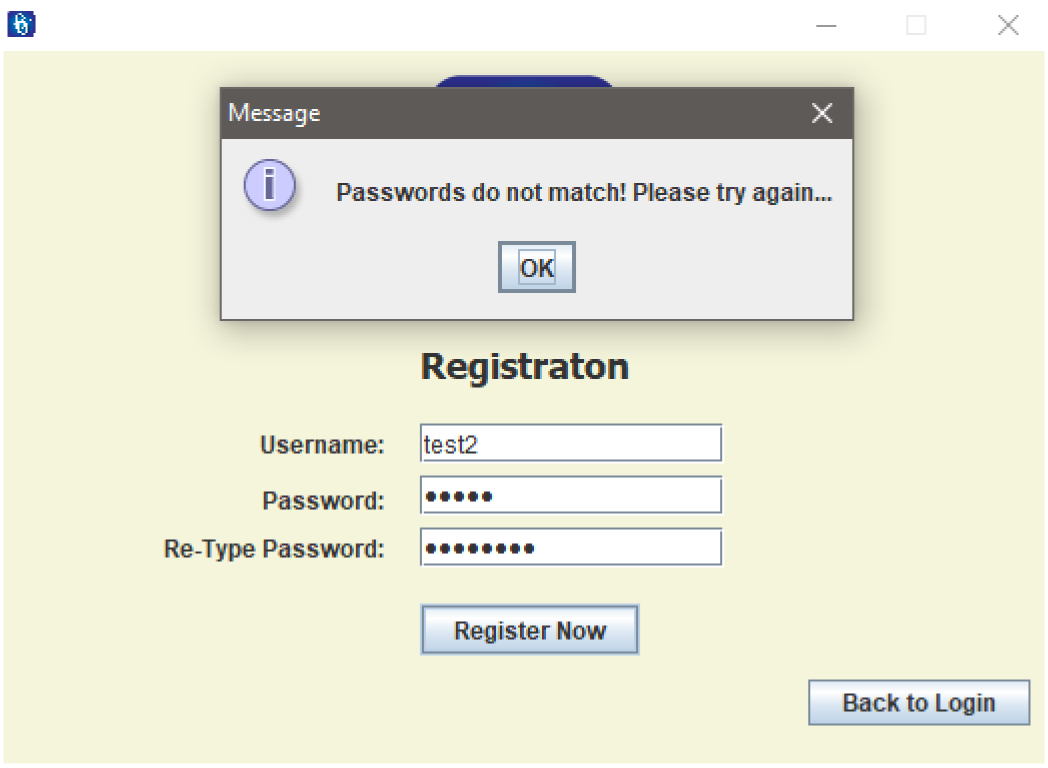

Message

Registration Failed: Username already taken!

This is what happens if the username is already taken by another user. This is executed if the try block fails. On line 155 For this try block to be executed there is preconditions. As seen on line 150-154 there is an if and two else if statements. These verify that there is something entered the username field and that the two passwords match I will put a screenshot of the two preconditions working below.


Message

Please enter a username.


Message

Passwords do not match! Please try again...

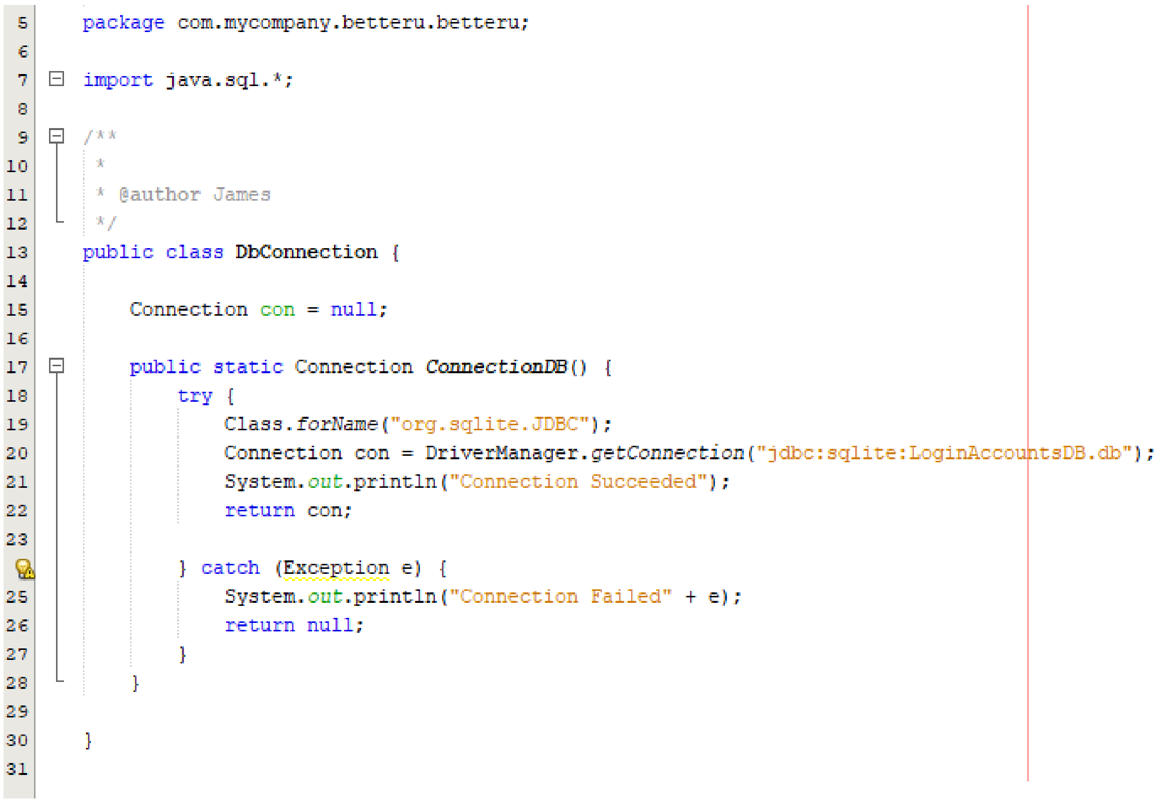
Registraton

Username: test2

Password: ●●●●●

Re-Type Password: ●●●●●●●●●

It is important to note that this all only works with a database connection being established. I will now show the database connection being established. It starts off with a file called DbConnection. I will put the code of this file below.

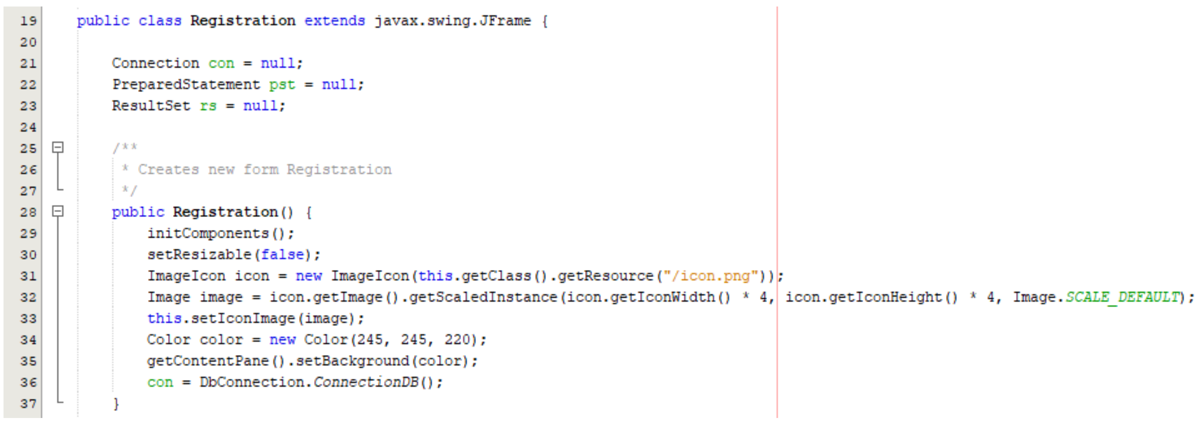


```
package com.mycompany.betteru.betteru;

import java.sql.*;

/**
 *
 * @author James
 */
public class DbConnection {

    Connection con = null;

    public static Connection ConnectionDB() {
        try {
            Class.forName("org.sqlite.JDBC");
            Connection con = DriverManager.getConnection("jdbc:sqlite:LoginAccountsDB.db");
            System.out.println("Connection Succeeded");
            return con;

        } catch (Exception e) {
            System.out.println("Connection Failed" + e);
            return null;
        }
    }

}
```

This is quite a small class but essential in this system. It has a simple try catch block that establishes the database connection it does this using the SQLite dependency from line

19 to 22 the connection is being made. The most important line is line 20 this connects to our LoginAccountsDB which is the name of the database for this system. If this fails, the catch block gets activated and prints an error message with the exception message.

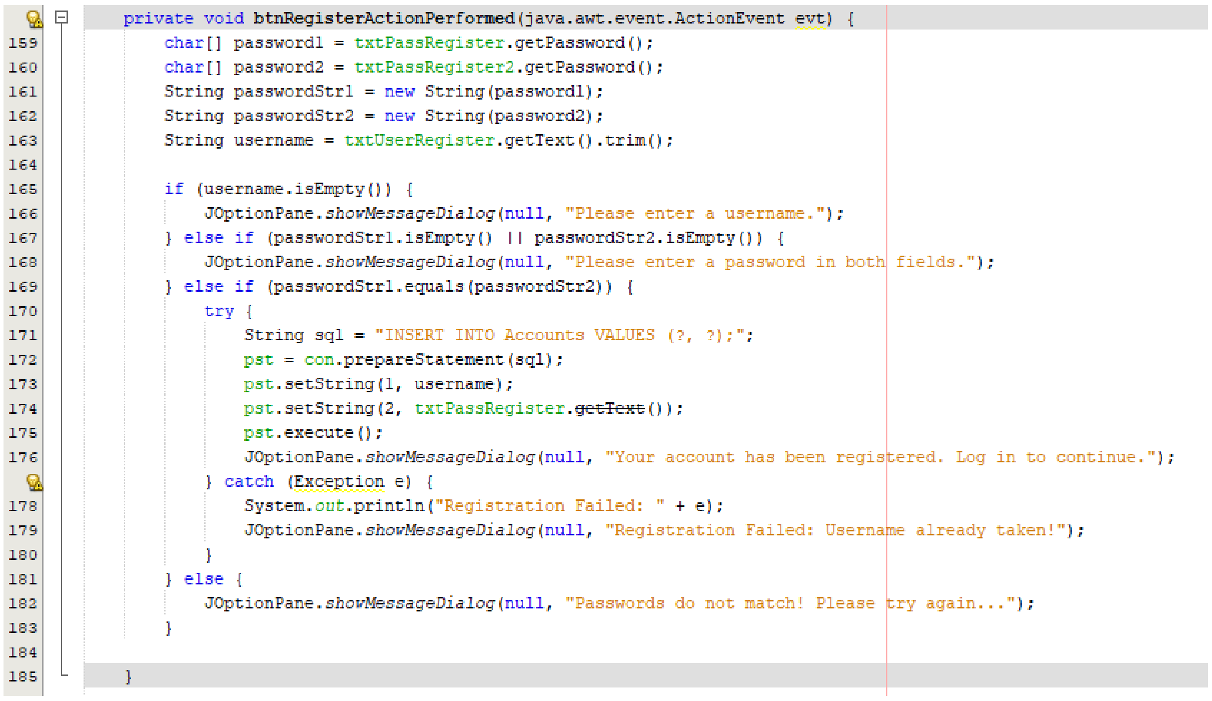
On the register page the database connection is established by the following code.



```
19  public class Registration extends javax.swing.JFrame {
20
21      Connection con = null;
22      PreparedStatement pst = null;
23      ResultSet rs = null;
24
25      /**
26       * Creates new form Registration
27       */
28      public Registration() {
29          initComponents();
30          setResizable(false);
31          ImageIcon icon = new ImageIcon(this.getClass().getResource("/icon.png"));
32          Image image = icon.getImage().getScaledInstance(icon.getIconWidth() * 4, icon.getIconHeight() * 4, Image.SCALE_DEFAULT);
33          this.setIconImage(image);
34          Color color = new Color(245, 245, 220);
35          getContentPane().setBackground(color);
36          con = DbConnection.ConnectionDB();
37      }
```

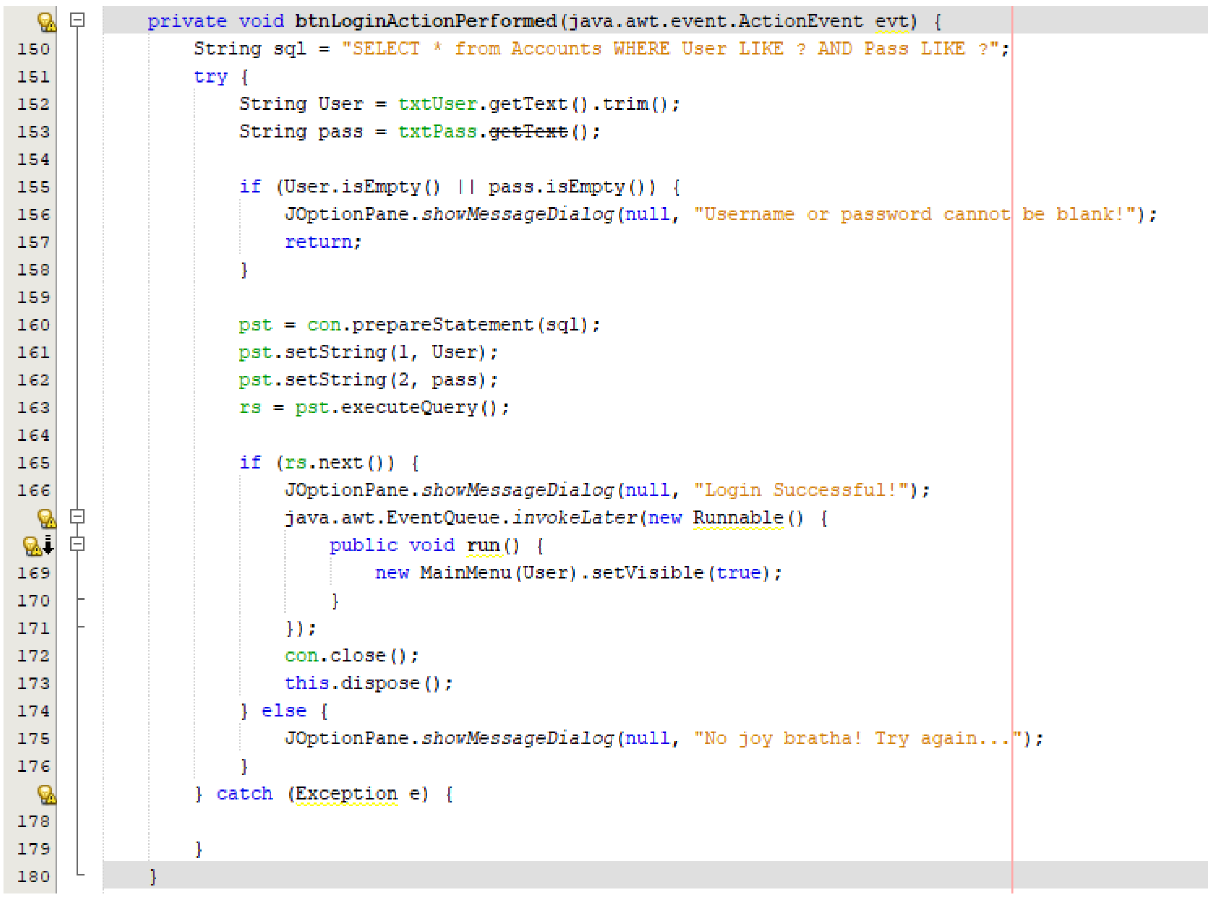
At the beginning lines 21 – 23 are global variables that are getting initialised to be able to allow the database connection. The next important bit of code is the constructor. It is line 36 this connects to the database using the DbConnection class.

If we look back at the register code, we can see the use of prepared statements this keeps the application secure and provides good performance and maintainability.



```
        private void btnRegisterActionPerformed(java.awt.event.ActionEvent evt) {
159         char[] password1 = txtPassRegister.getPassword();
160         char[] password2 = txtPassRegister2.getPassword();
161         String passwordStr1 = new String(password1);
162         String passwordStr2 = new String(password2);
163         String username = txtUserRegister.getText().trim();
164
165         if (username.isEmpty()) {
166             JOptionPane.showMessageDialog(null, "Please enter a username.");
167         } else if (passwordStr1.isEmpty() || passwordStr2.isEmpty()) {
168             JOptionPane.showMessageDialog(null, "Please enter a password in both fields.");
169         } else if (passwordStr1.equals(passwordStr2)) {
170             try {
171                 String sql = "INSERT INTO Accounts VALUES (?, ?);";
172                 pst = con.prepareStatement(sql);
173                 pst.setString(1, username);
174                 pst.setString(2, txtPassRegister.~~getText~~());
175                 pst.execute();
176                 JOptionPane.showMessageDialog(null, "Your account has been registered. Log in to continue.");
                } catch (Exception e) {
178                 System.out.println("Registration Failed: " + e);
179                 JOptionPane.showMessageDialog(null, "Registration Failed: Username already taken!");
180             }
181         } else {
182             JOptionPane.showMessageDialog(null, "Passwords do not match! Please try again...");
183         }
184
185     }
```

On line 172-175 we can see some prepared statements being used. Once a user has been registered the user can then close the registration page and then login to the system. The code for the login is below.

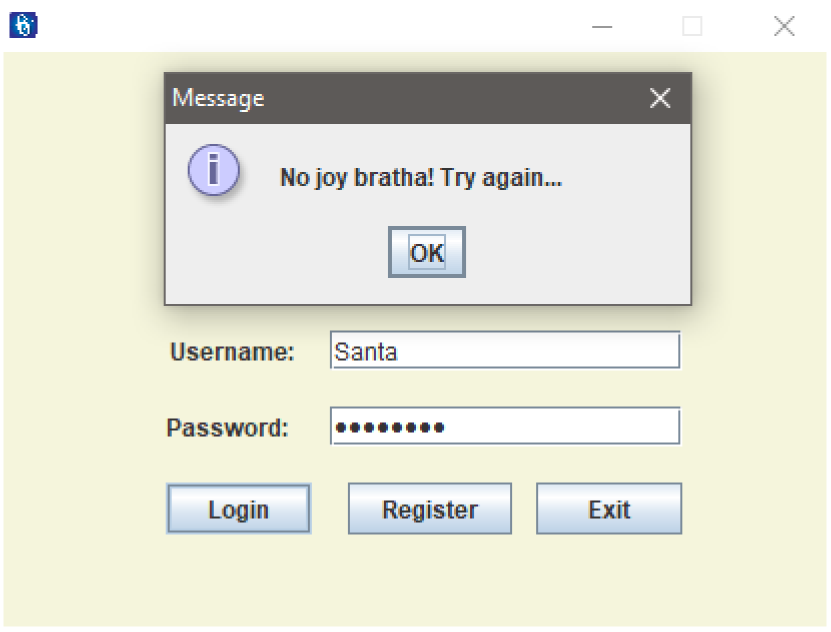


```
    private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
150     String sql = "SELECT * from Accounts WHERE User LIKE ? AND Pass LIKE ?";
151     try {
152         String User = txtUser.getText().trim();
153         String pass = txtPass.~~getText~~();
154 
155         if (User.isEmpty() || pass.isEmpty()) {
156             JOptionPane.showMessageDialog(null, "Username or password cannot be blank!");
157             return;
158         }
159 
160         pst = con.prepareStatement(sql);
161         pst.setString(1, User);
162         pst.setString(2, pass);
163         rs = pst.executeQuery();
164 
165         if (rs.next()) {
166             JOptionPane.showMessageDialog(null, "Login Successful!");
                java.awt.EventQueue.invokeLater(new Runnable() {
                    public void run() {
169                     new MainMenu(User).setVisible(true);
170                 }
171             });
172             con.close();
173             this.dispose();
174         } else {
175             JOptionPane.showMessageDialog(null, "No joy bratha! Try again...");
176         }
        } catch (Exception e) {
178 
179     }
180 }
```

This works by using a SQL select statement. The statement can be seen on line 150. It selects from the accounts where the username and password match one of the entries in the system. The try block then activates and the first line in the try block is grabbing the username typed into the username filed in the system and assigning it to a variable. The variable is User. This is very important as this is the foundation for the session management throughout the system. The lines 160-163 this is the use of prepared statements. Line 161 sets the first parameter in the prepared statement to the User variable. 162 sets the second parameter in the prepared statement to what is typed in the password field. The line 163 executes the prepared statement and then returns a result set. That's what the rs stands for. The rs.next makes sure that the result has at least one row. If there is a row that means the credentials matched a row in the accounts table which means the user exists on the system. This allows the code inside the if statement to run which opens the main menu and allows the User variable to be passed through. After this the connection to the database is closed as this is good practice.

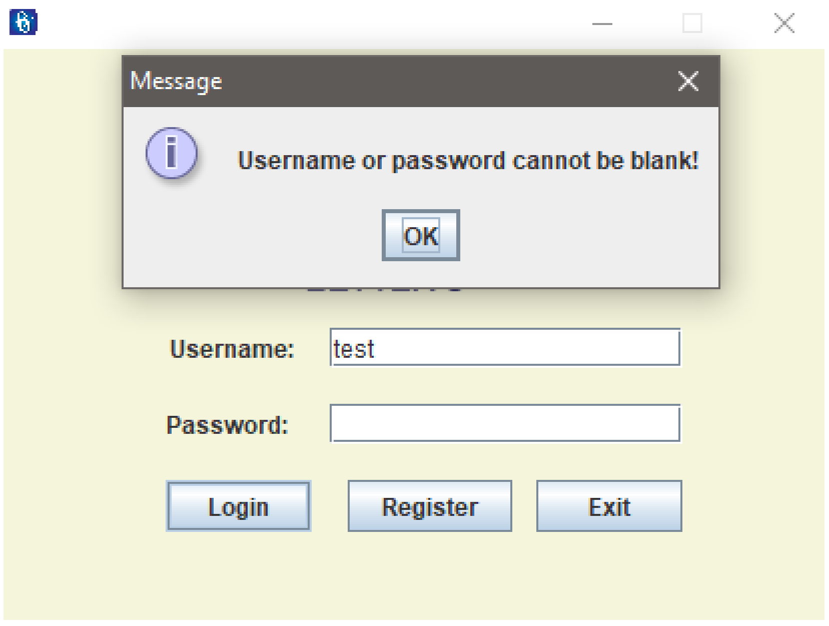
If the user does not exist in the database, the else statement runs letting the user know that they have had no joy in logging in and to try again.

So, we will try login with invalid credentials first and then we will login with valid credentials


Message

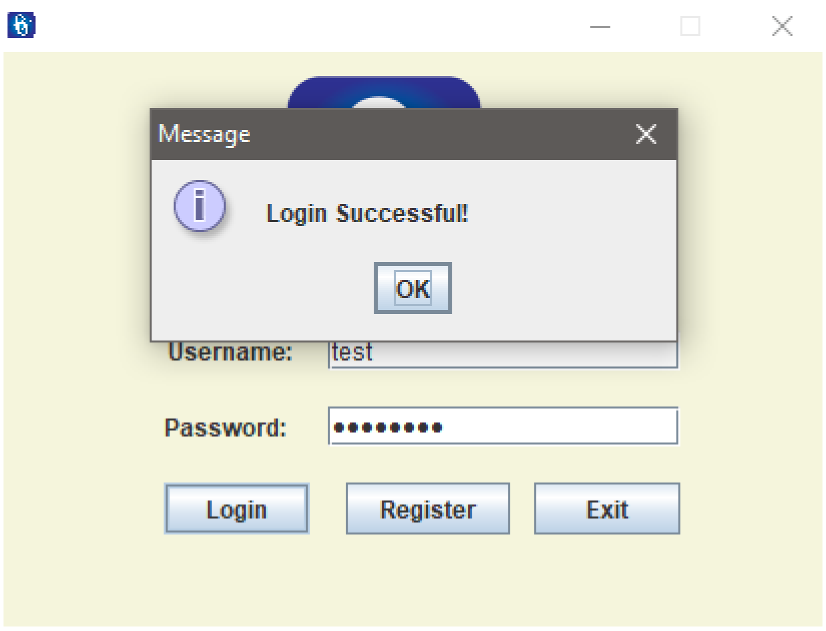
No joy bratha! Try again...

This is what happens when a user logs in with invalid credentials below is what happens when a user logs while leaving either the username or password blank.


Message

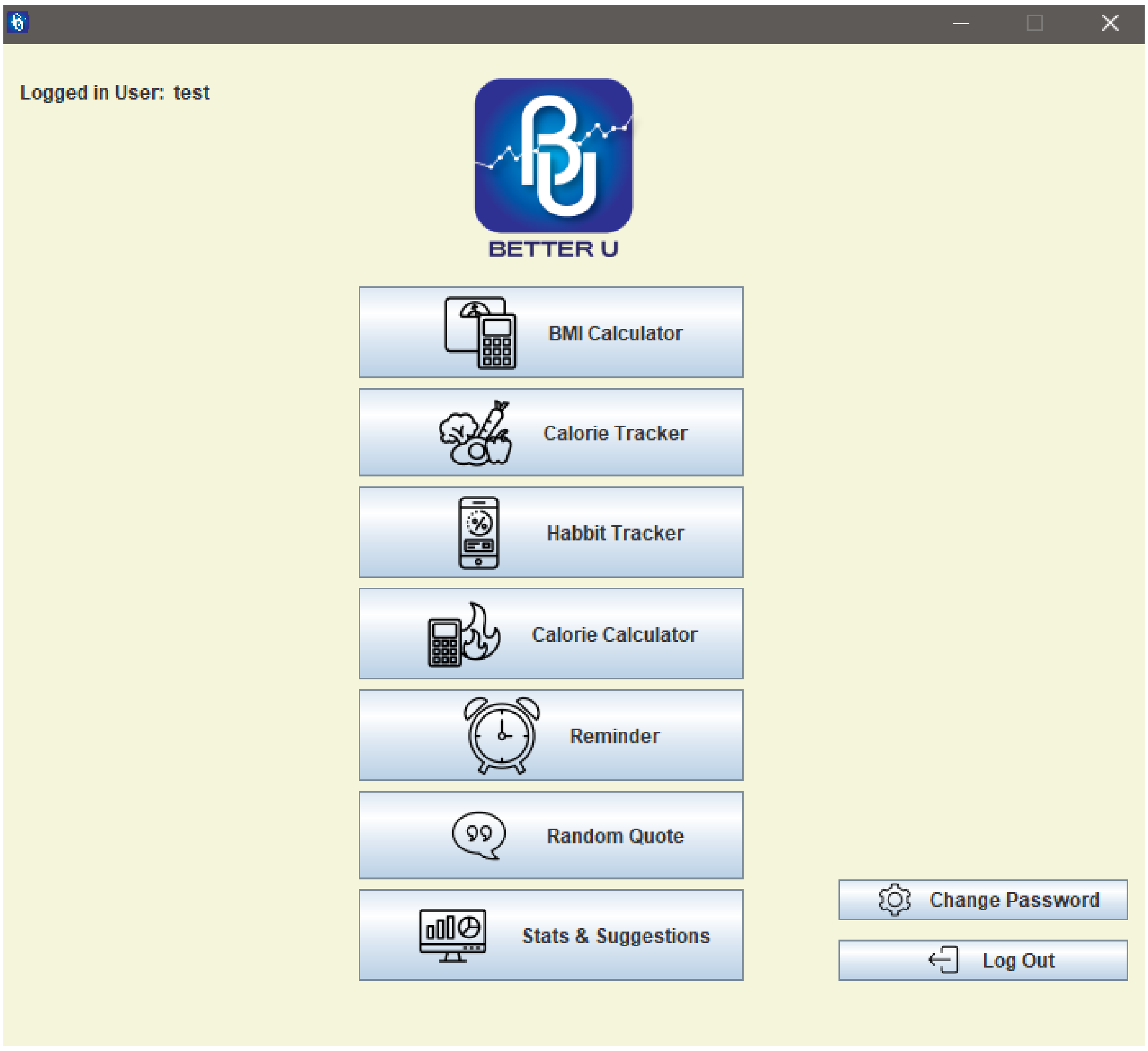
Username or password cannot be blank!

This will happen if the username or password field is left blank. Below is what happens when a user successfully logs in to the system.


Message

Login Successful!

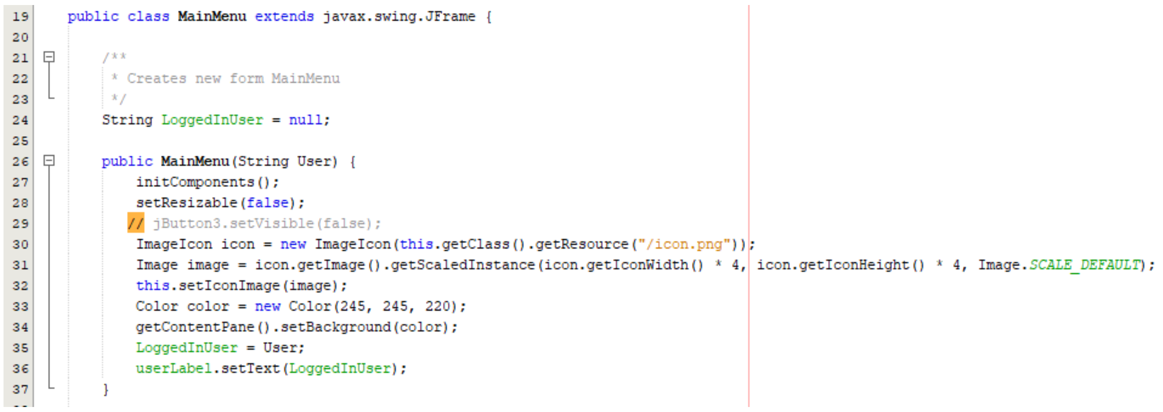
Upon pressing the ok button, the user is brought to the main menu.


Logged in User: test

In the main menu screen, we can see that there are many buttons. 7 in the centre and 2 in the right-hand corner. There is also some text in the top left-hand corner to indicate which user is using the system. In the centre the functions of the system can be seen. The BMI calculator, the calorie tracker, habit tracker, calorie calculator, reminder, random quote and stats and suggestions all have their own buttons and icon. If a user clicks one of these buttons they will be taken to that feature. On the right-hand side there is the change password feature and then the logout button.

The BMI calculator lets a user be able to calculate their BMI. The calorie Tracker allows a user to be able to track their calories. The habit Tracker allows a user to be able to track their habits. The calorie calculator allows user to be able to track their calories. The reminder button opens the reminder page which allows a user to be able to create a reminder. The random quote gets a random inspirational quote for a user form an API and the stats and suggestions generate statis and suggestions for the user bases on information they entered in the other features. It also allows a user to be able to generate graphs bases on the calorie tracker data and the BMI calculator data.

The main menu also plays a big role in the session management of the system. The User variable was passed through to the main menu upon a user logging into the system. This user value is assigned to a variable in the Main Menu.

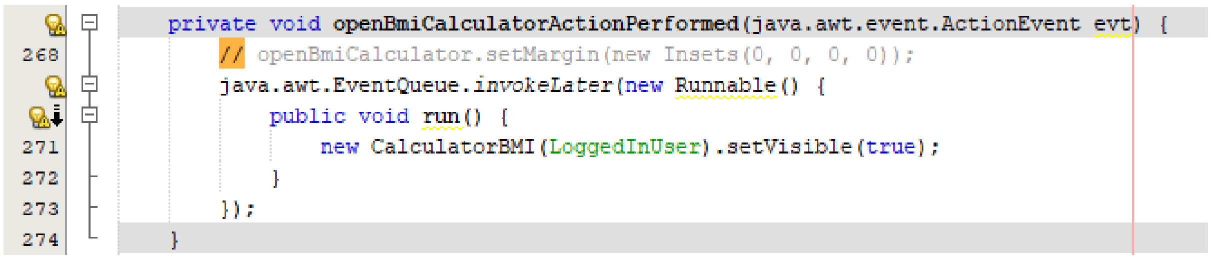


```
19  public class MainMenu extends javax.swing.JFrame {
20  
21      /**
22       * Creates new form MainMenu
23       */
24      String LoggedInUser = null;
25  
26      public MainMenu(String User) {
27          initComponents();
28          setResizable(false);
29         // jButton3.setVisible(false);
30          ImageIcon icon = new ImageIcon(this.getClass().getResource("/icon.png"));
31          Image image = icon.getImage().getScaledInstance(icon.getIconWidth() * 4, icon.getIconHeight() * 4, Image.SCALE_DEFAULT);
32          this.setIconImage(image);
33          Color color = new Color(245, 245, 220);
34          getContentPane().setBackground(color);
35          LoggedInUser = User;
36          userLabel.setText(LoggedInUser);
37      }
```

In line 26 we can see that the User is passed through into the Main Menu. There is a global variable established in this menu code that is to be used. This variable is on line 24 and it is the LoggedInUser variable. On line 35 the LoggedInUser is set to equal the User value. We now have the user who logged in passed into the main menu. This is then displayed by line 36 which sets the label at the top right of the screen to show which user is currently logged into the system.

This is the basis of the session management of the system. The LoggedInUser will be passed through the program as the user opens different features.

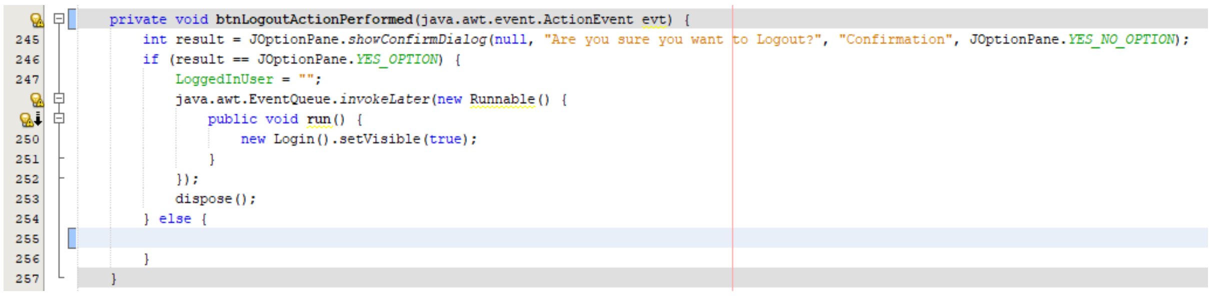
The main menu serves as the gateway to all the other features in the system. Below I will show one of the buttons from the 7 centre buttons. I will show one button as each of the 7 buttons work the same way.



```
    private void openBmiCalculatorActionPerformed(java.awt.event.ActionEvent evt) {
268     //  openBmiCalculator.setMargin(new Insets(0, 0, 0, 0));
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
271             new CalculatorBMI(LoggedInUser).setVisible(true);
272         }
273     });
274 }
```

Here we can see that the button works by opening the class that is called CalculatorBMI. This class has no main method so it can only be opened from here. In line 271 we can see that the LoggedInUser is being passed into the CalculatorBMI class that's being opened. This process is repeated for the other 6 buttons in the centre.

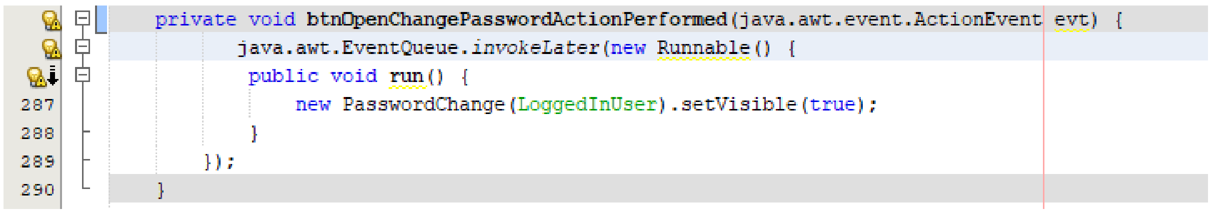
I will now show the two buttons that are on the bottom right-hand corner of the screen. These buttons are the change password button and the logout button. I will put the code for the logout button below.



```
    private void btnLogoutActionPerformed(java.awt.event.ActionEvent evt) {
245     int result = JOptionPane.showConfirmDialog(null, "Are you sure you want to Logout?", "Confirmation", JOptionPane.YES_NO_OPTION);
246     if (result == JOptionPane.YES_OPTION) {
247         LoggedInUser = "";
            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
250                 new Login().setVisible(true);
251             }
252         });
253         dispose();
254     } else {
255
256     }
257 }
```

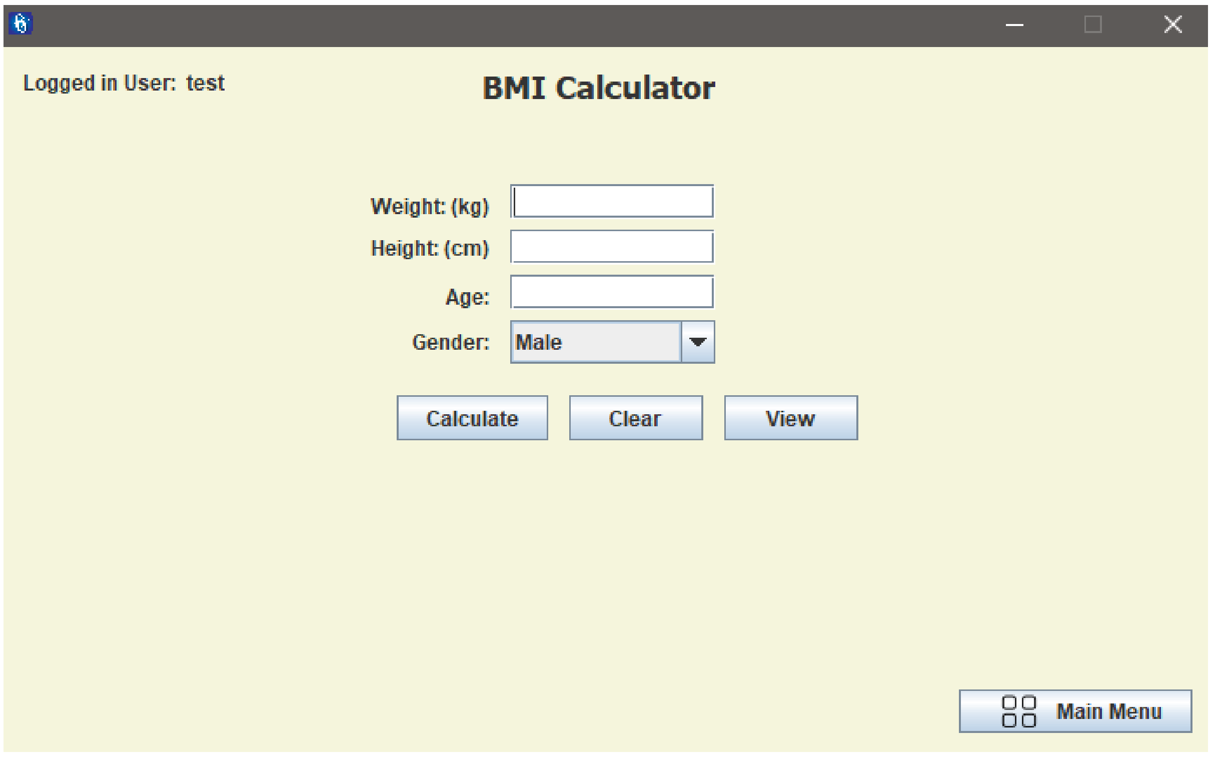
As seen in the above screenshot if a user selects to logout the user is asked if they are sure if they want to logout. It gives them a yes or no option. This can be seen on line 245. If a user presses No the confirmation dialogue closes but if the user presses the yes option, the code from within the if statement on line 246 gets triggered. The code inside the if statement brings the user back to the login page but does not pass the LoggedInUser value through so the user is back to the start.

I will now go through the change password button. This is like the code for the main 7 buttons for opening the features of the application. The code can be seen below.



```
    private void btnOpenChangePasswordActionPerformed(java.awt.event.ActionEvent evt) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
287             new PasswordChange(LoggedInUser).setVisible(true);
288         }
289     });
290 }
```

Here the LoggedInUser value is being passed through. I will now go through the main features of my system. I will begin by starting from the top of the main menu. The first feature is the BMI calculator. To open the BMI calculator the user just must click the BMI Calculator button. Upon clicking the BMI calculator a new window is opened and the BMI calculator can be seen.

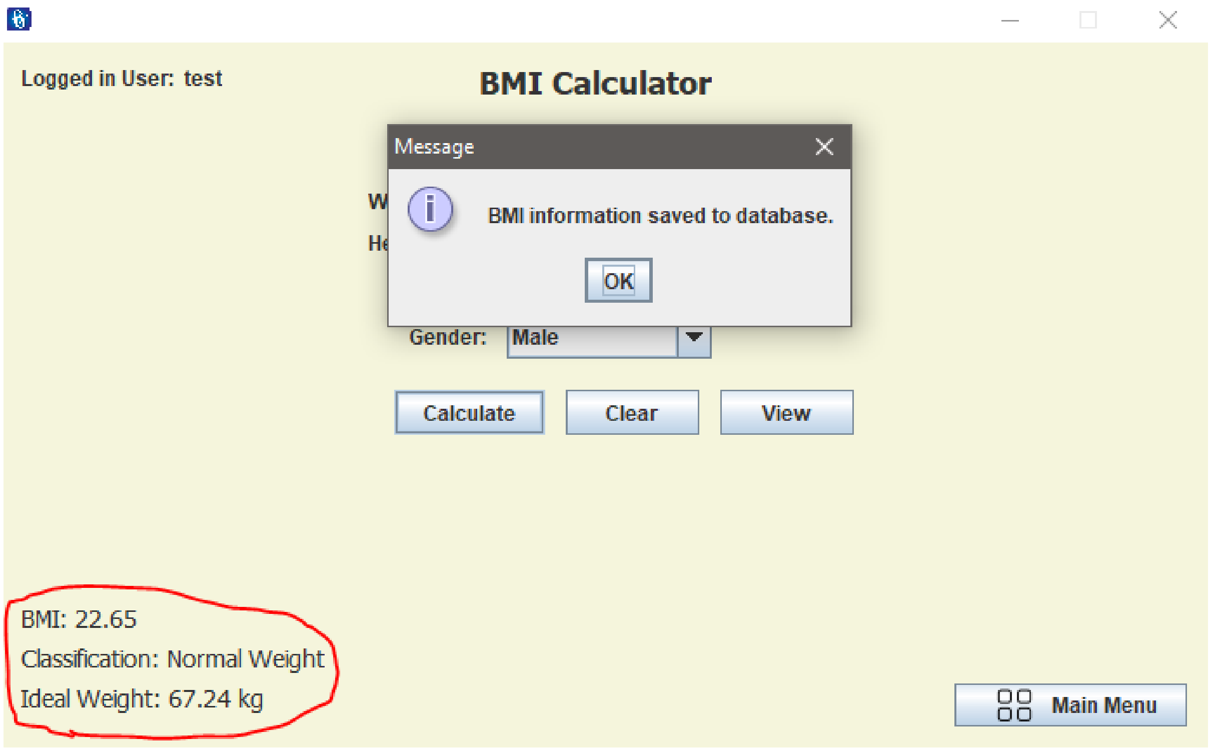

Logged in User: test

BMI Calculator

Weight: (kg)
Height: (cm)
Age:
Gender: Male

The BMI calculator calculates the users BMI by taking in their weight, height, age, and gender. When a user enters in these values the calculate button can be pressed to calculate the users BMI. The result is displayed on the bottom left of the window. The user can also clear the textfields by pressing clear. After this the user can press view which opens a new window for the user to be able to view previous BMI calculation and be able to delete past calculations.

I will show you what happens when a calculation is created.


Logged in User: test

BMI Calculator

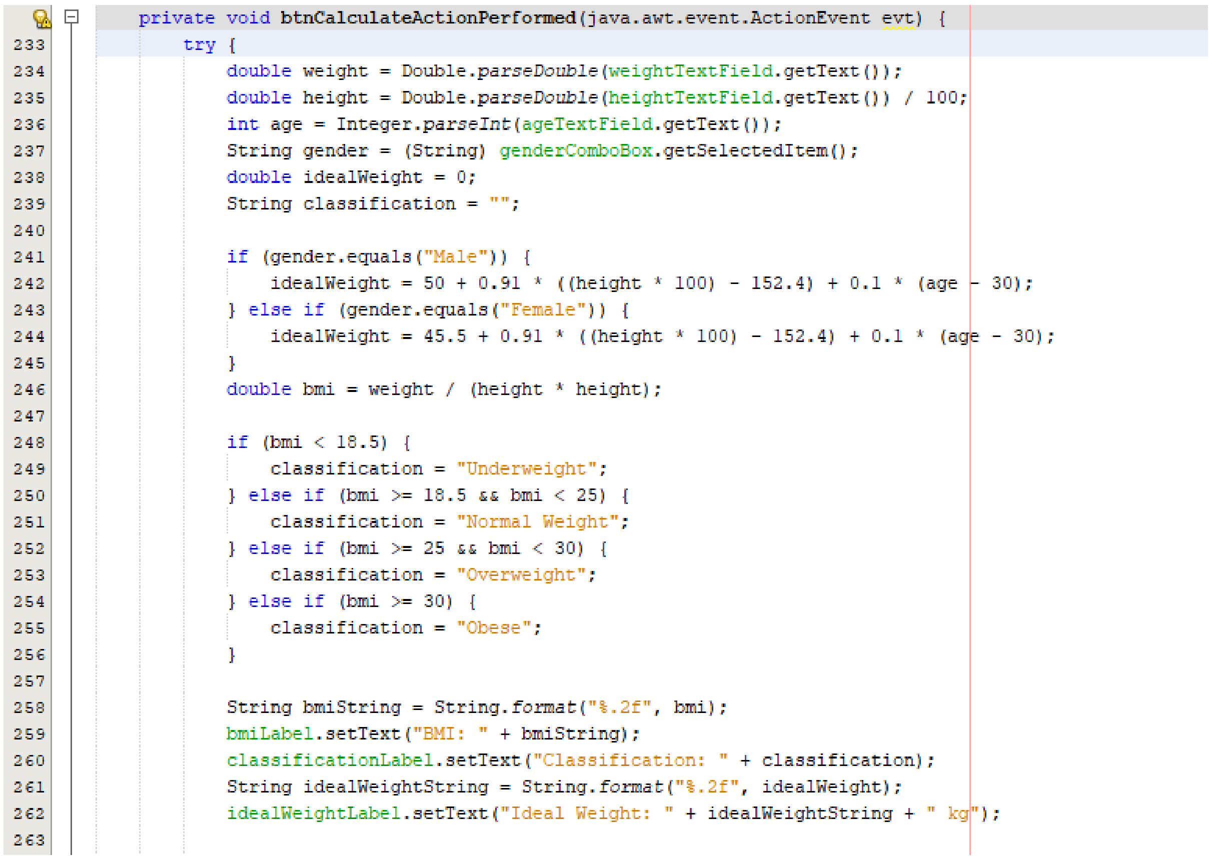
Message

BMI information saved to database.

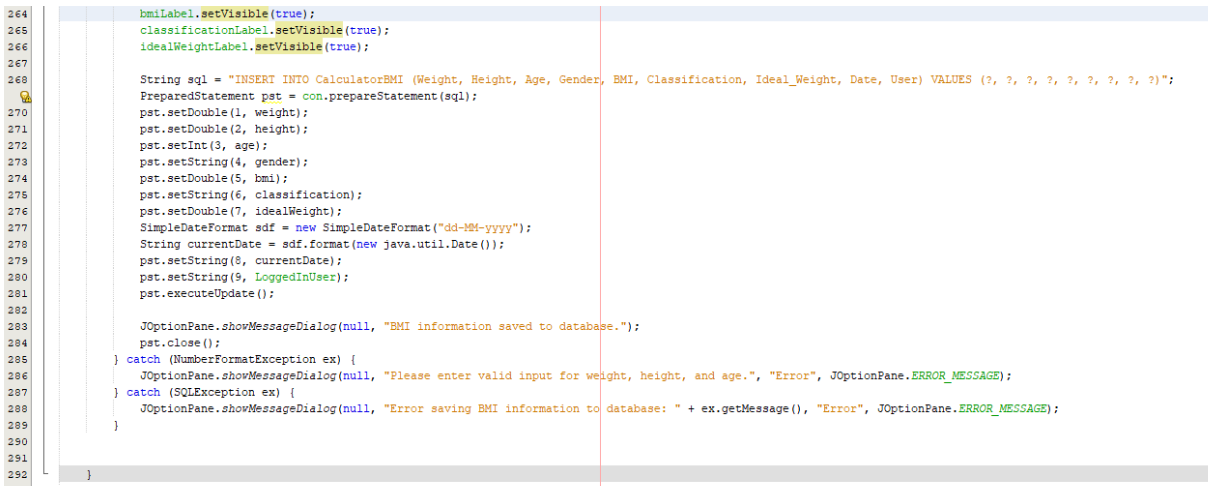
Gender: Male

BMI: 22.65
Classification: Normal Weight
Ideal Weight: 67.24 kg

As seen a message comes up saying that that the BMI information has been saved to the database. In the bottom left corner, the calculation for the BMI has been produced. It gives a calculation, classification, and Ideal Weight for the user. The user input along with the Three generated results are saved to the database. I will now put the code for the calculate button below.



```
        private void btnCalculateActionPerformed(java.awt.event.ActionEvent evt) {
233         try {
234             double weight = Double.parseDouble(weightTextField.getText());
235             double height = Double.parseDouble(heightTextField.getText()) / 100;
236             int age = Integer.parseInt(ageTextField.getText());
237             String gender = (String) genderComboBox.getSelectedItem();
238             double idealWeight = 0;
239             String classification = "";
240
241             if (gender.equals("Male")) {
242                 idealWeight = 50 + 0.91 * ((height * 100) - 152.4) + 0.1 * (age - 30);
243             } else if (gender.equals("Female")) {
244                 idealWeight = 45.5 + 0.91 * ((height * 100) - 152.4) + 0.1 * (age - 30);
245             }
246             double bmi = weight / (height * height);
247
248             if (bmi < 18.5) {
249                 classification = "Underweight";
250             } else if (bmi >= 18.5 && bmi < 25) {
251                 classification = "Normal Weight";
252             } else if (bmi >= 25 && bmi < 30) {
253                 classification = "Overweight";
254             } else if (bmi >= 30) {
255                 classification = "Obese";
256             }
257
258             String bmiString = String.format("%.2f", bmi);
259             bmiLabel.setText("BMI: " + bmiString);
260             classificationLabel.setText("Classification: " + classification);
261             String idealWeightString = String.format("%.2f", idealWeight);
262             idealWeightLabel.setText("Ideal Weight: " + idealWeightString + " kg");
263
```



```
264             bmiLabel.setVisible(true);
265             classificationLabel.setVisible(true);
266             idealWeightLabel.setVisible(true);
267
268             String sql = "INSERT INTO CalculatorBMI (Weight, Height, Age, Gender, BMI, Classification, Ideal_Weight, Date, User) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
                PreparedStatement pst = con.prepareStatement(sql);
270             pst.setDouble(1, weight);
271             pst.setDouble(2, height);
272             pst.setInt(3, age);
273             pst.setString(4, gender);
274             pst.setDouble(5, bmi);
275             pst.setString(6, classification);
276             pst.setDouble(7, idealWeight);
277             SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
278             String currentDate = sdf.format(new java.util.Date());
279             pst.setString(8, currentDate);
280             pst.setString(9, LoggedInUser);
281             pst.executeUpdate();
282
283             JOptionPane.showMessageDialog(null, "BMI information saved to database.");
284             pst.close();
285         } catch (NumberFormatException ex) {
286             JOptionPane.showMessageDialog(null, "Please enter valid input for weight, height, and age.", "Error", JOptionPane.ERROR_MESSAGE);
287         } catch (SQLException ex) {
288             JOptionPane.showMessageDialog(null, "Error saving BMI information to database: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
289         }
290
291
292     }
```

As seen in line 234-237 the input is being taken in from the user to perform the calculations for BMI.

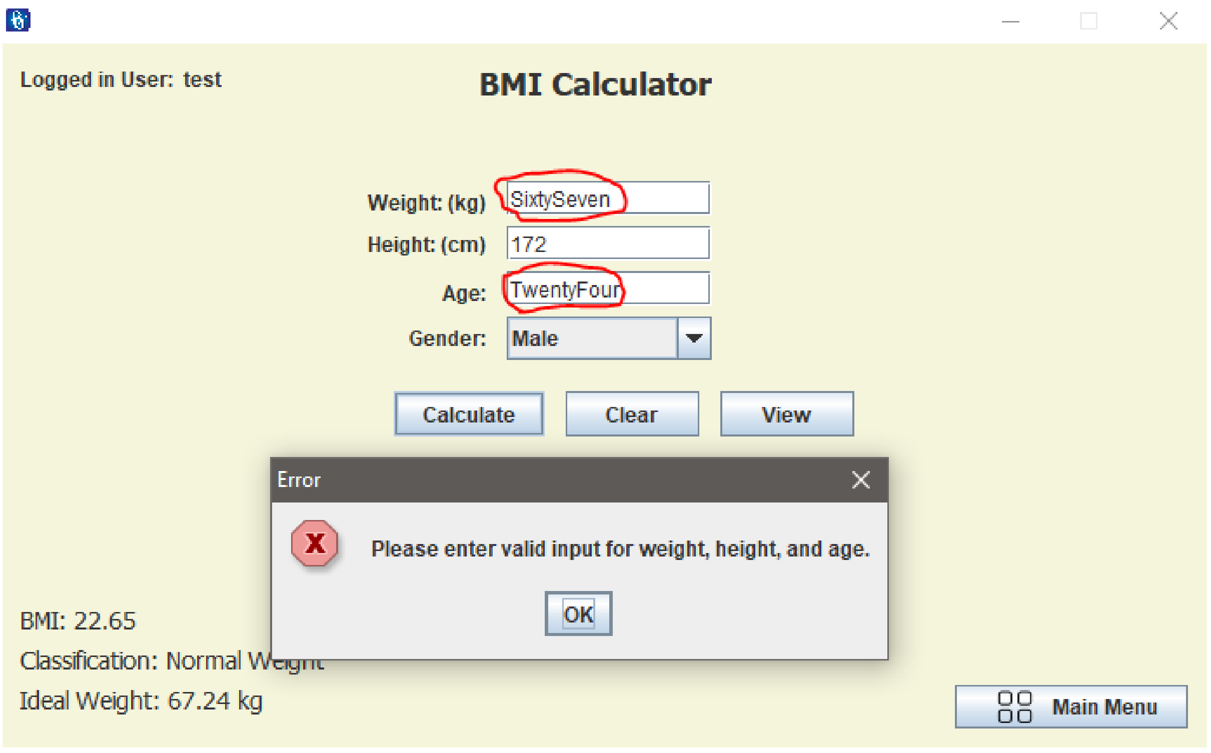
The if statement from line 241 to 246 sets out the different calculations for the ideal weight of a user depending on if they are a man or a woman. It is calculated using the height of the user and the gender. There are two equations and depending on if you are a man or a woman your height is plugged in to generate the users' ideal calories. Line 246 sets out the equation for BMI. This is calculated by dividing the weight of the user in kilograms by the square of the height of the user in meters. The height is taken off the user in centimetres but is divided by 100 in line 235 so the system sees the height in meters.

Once the BMI is calculated there is a classification attributed to the BMI. This is done in line 248-256 depending on the BMI value the classification can either be Underweight, Normal Weight, Overweight, or Obese. The if statements control the outcome.

Lines 258-266 set the labels to print out the results from the BMI calculations. The labels are originally hidden but become visible when there is a calculation.

Lines 268-281 control the saving of the data to the database. Line 268 has the SQL statement uses an INSERT to insert the data into the database. It is a prepared statement, and it is executed on line 269. The rest of the lines are just plugging the values into the query. This is then done on line 281. Line 283 and 284 show a message to the user that the data has been saved to the database and then it closes the connection.

So far, all the code discussed has been in a try catch block. The code explained was in the try block. There are two catch blocks. The first one catches a number format exception. This produces an error if the user does not enter a valid input for weight, height, and age. This can be seen working below.


Logged in User: test

BMI Calculator

Weight: (kg) SixtySeven
Height: (cm) 172
Age: TwentyFour
Gender: Male

Error

Please enter valid input for weight, height, and age.

BMI: 22.65
Classification: Normal Weight
Ideal Weight: 67.24 kg

As seen in the above screenshot the system prompts the user to enter a valid input for weight, height, and age.

The next catch block catches the SQL exception. If there is a SQL exception it displays a message saying error saving BMI information to database.

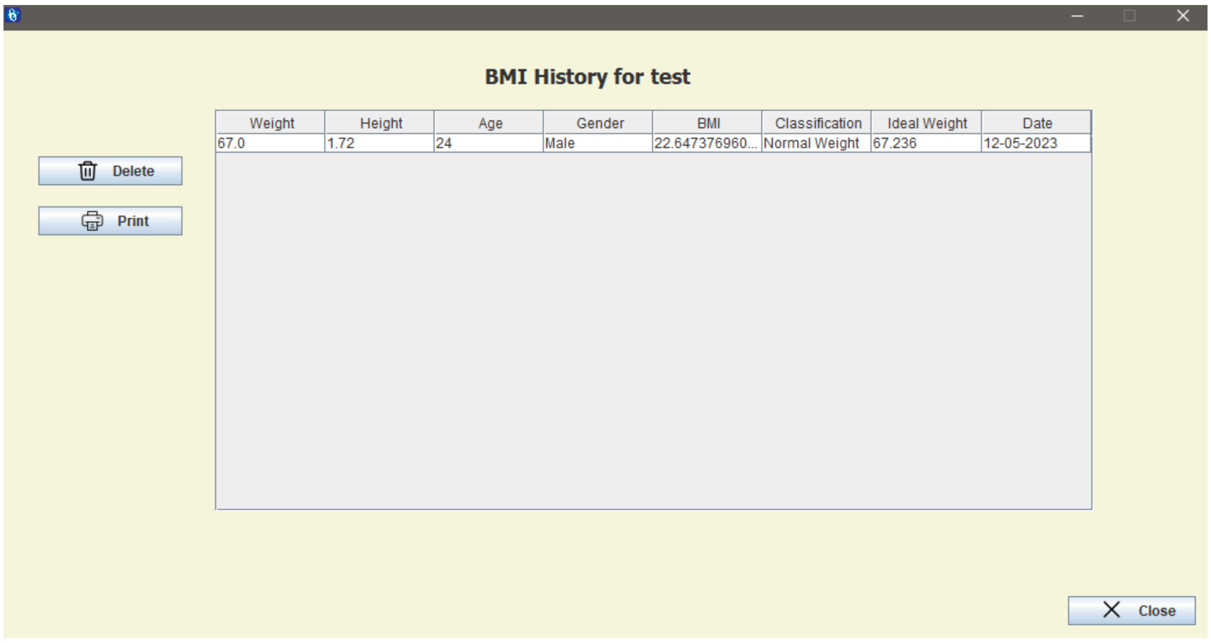
I will now show the clear button which resets all the labels for results and the textfields for the user's input.

```java
    private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {
        weightTextField.setText("");
        heightTextField.setText("");
        ageTextField.setText("");
        bmiLabel.setText("BMI: ");
        classificationLabel.setText("Classification: ");
        idealWeightLabel.setText("Ideal Weight: ");


    }
```

The next button is the view button which opens a new window where the user can view previous calculations. Below is the code for the view button.

```java
private void btnViewActionPerformed(java.awt.event.ActionEvent evt) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new ViewCalculatorBMI(LoggedInUser).setVisible(true);
        }
    });
}
```

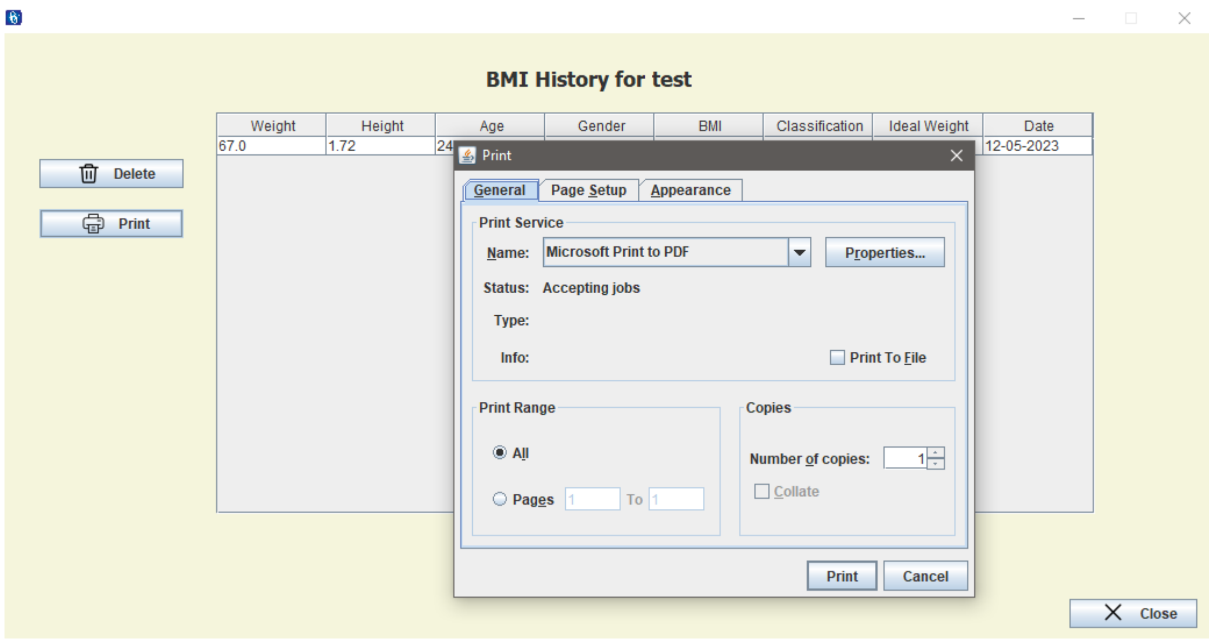
This code opens the view class and passes the LoggedInUser value for session management. Upon clicking this button the views sees this window.



BMI History for test

Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date
67.0	1.72	24	Male	22.647376960...	Normal Weight	67.236	12-05-2023

Here the user can see the information entered the database when the user preformed the calculation. The user has two options here and that is to either delete an entry or print all entries. As seen the information that was used for the calculation and the result are stored here. The print button prints the contents of the table if the user presses the print button it looks like this.



BMI History for test

Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date
67.0	1.72	24					12-05-2023

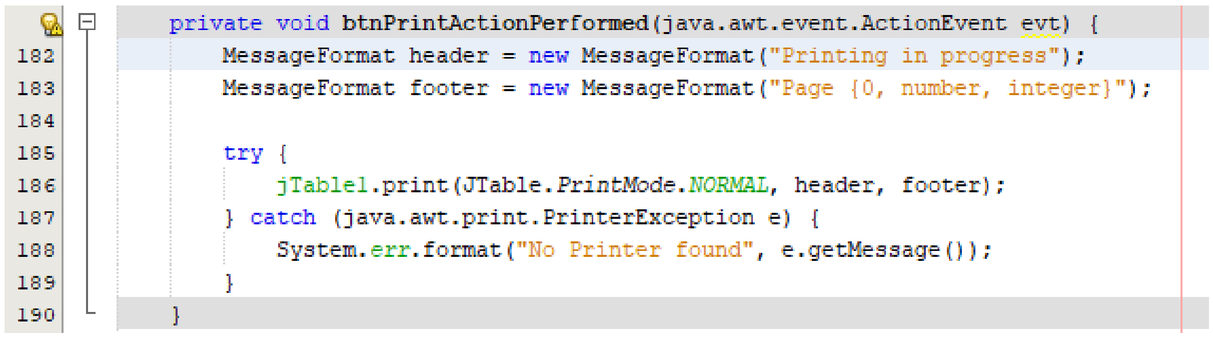
Print — General | Page Setup | Appearance

Print Service
Name: Microsoft Print to PDF
Status: Accepting jobs
Type:
Info:
Print To File: ☐

Print Range: ● All ○ Pages 1 To 1

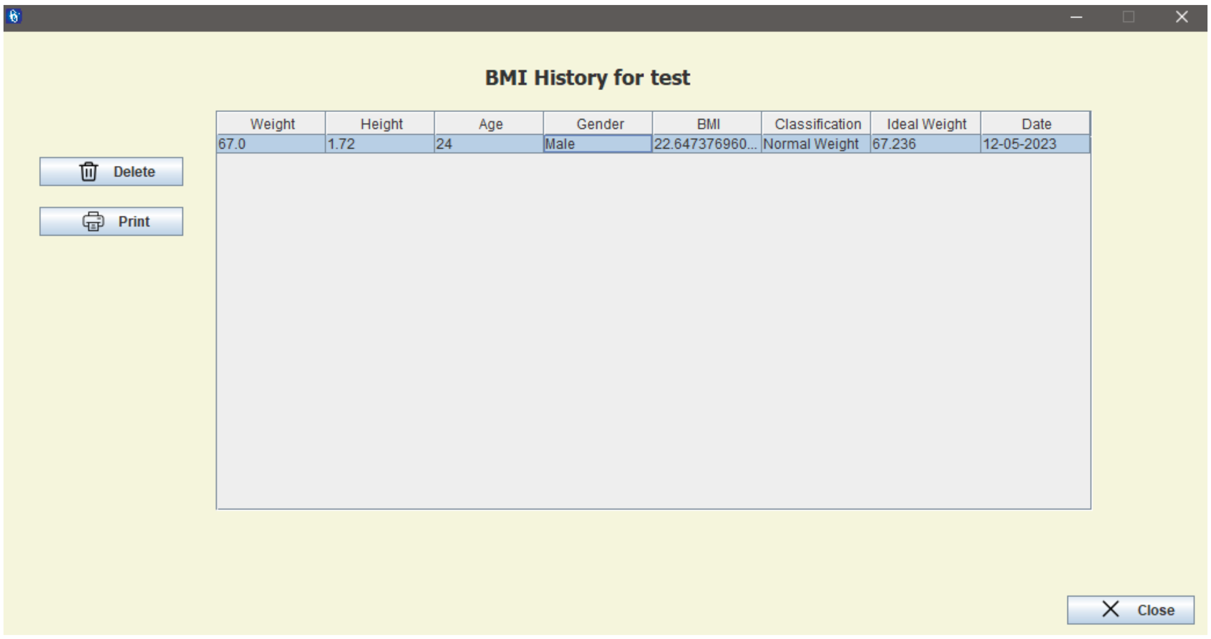
Copies
Number of copies: 1
Collate: ☐

A user can then choose to print out the contents of the table. The code behind the print button is below.



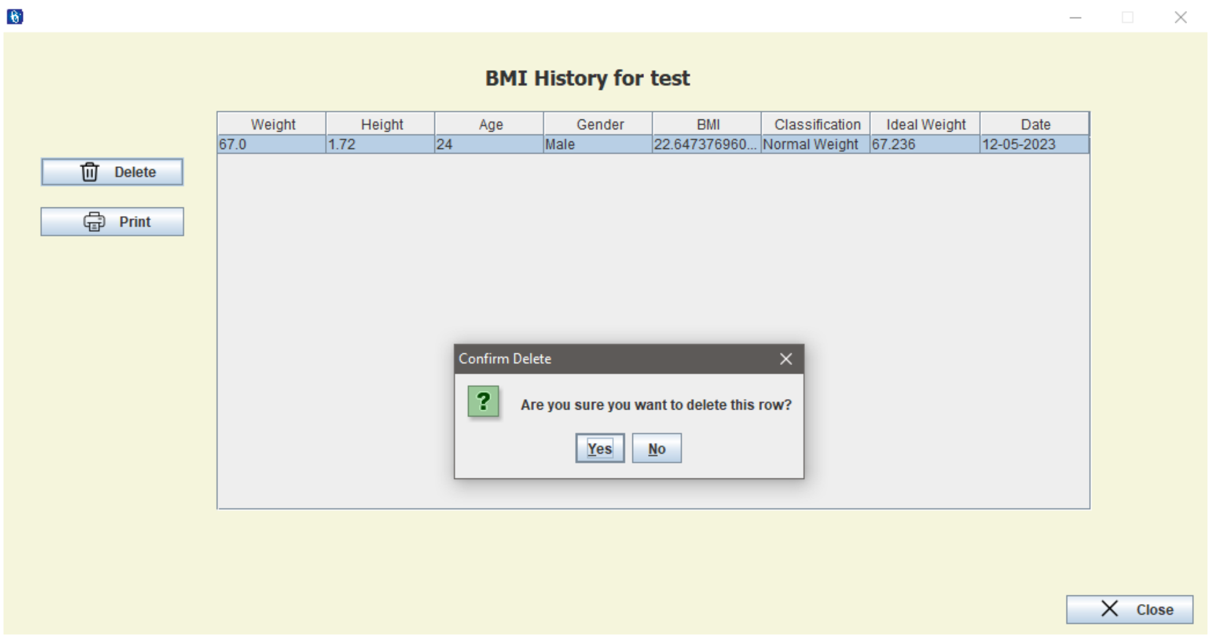
```
private void btnPrintActionPerformed(java.awt.event.ActionEvent evt) {
    MessageFormat header = new MessageFormat("Printing in progress");
    MessageFormat footer = new MessageFormat("Page {0, number, integer}");

    try {
        jTable1.print(JTable.PrintMode.NORMAL, header, footer);
    } catch (java.awt.print.PrinterException e) {
        System.err.format("No Printer found", e.getMessage());
    }
}
```

A user can click an entry and then delete if so choose. Below is a screenshot of this.


BMI History for test

Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date
67.0	1.72	24	Male	22.647376960...	Normal Weight	67.236	12-05-2023

Entry is clicked.

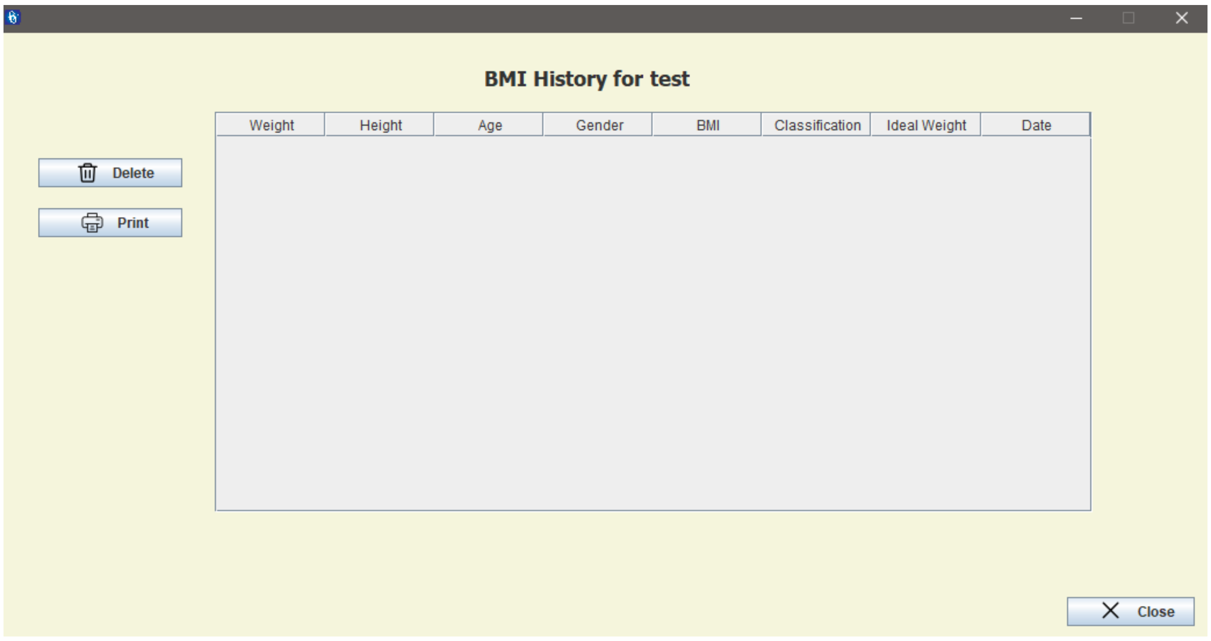

BMI History for test

Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date
67.0	1.72	24	Male	22.647376960...	Normal Weight	67.236	12-05-2023

Confirm Delete

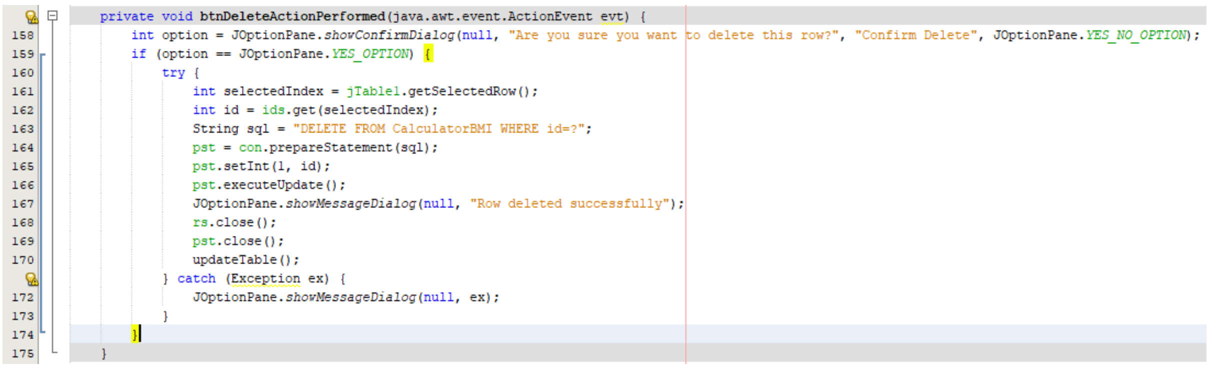
Are you sure you want to delete this row?

User presses delete. If a user presses no it closes the dialogue, and nothing is deleted. If a user presses yes the entry is deleted.



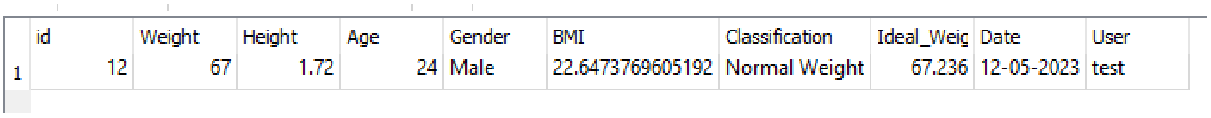
Entry was deleted.

The code for the delete button is below.



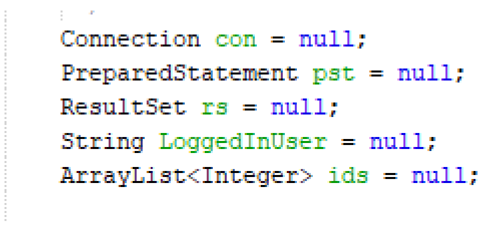
```
        private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
158         int option = JOptionPane.showConfirmDialog(null, "Are you sure you want to delete this row?", "Confirm Delete", JOptionPane.YES_NO_OPTION);
159         if (option == JOptionPane.YES_OPTION) {
160             try {
161                 int selectedIndex = jTable1.getSelectedRow();
162                 int id = ids.get(selectedIndex);
163                 String sql = "DELETE FROM CalculatorBMI WHERE id=?";
164                 pst = con.prepareStatement(sql);
165                 pst.setInt(1, id);
166                 pst.executeUpdate();
167                 JOptionPane.showMessageDialog(null, "Row deleted successfully");
168                 rs.close();
169                 pst.close();
170                 updateTable();
                } catch (Exception ex) {
172                 JOptionPane.showMessageDialog(null, ex);
173             }
174         }
175     }
```

The delete button first asks the user if they are sure if they want to delete this row. This is on line 158. On line 159 there is an if statement that if the user presses yes will activate a try catch block the try deletes the entry from the database it does this by deleting from the database where the ID is a certain value. This can be seen on line 163. You may have noticed there is no visible ID in the table. There is in fact an ID attributed to each entry the ID is just not visible to the user. Below is a screenshot of the database and you can see the ID.



	id	Weight	Height	Age	Gender	BMI	Classification	Ideal_Weig	Date	User
1	12	67	1.72	24	Male	22.6473769605192	Normal Weight	67.236	12-05-2023	test

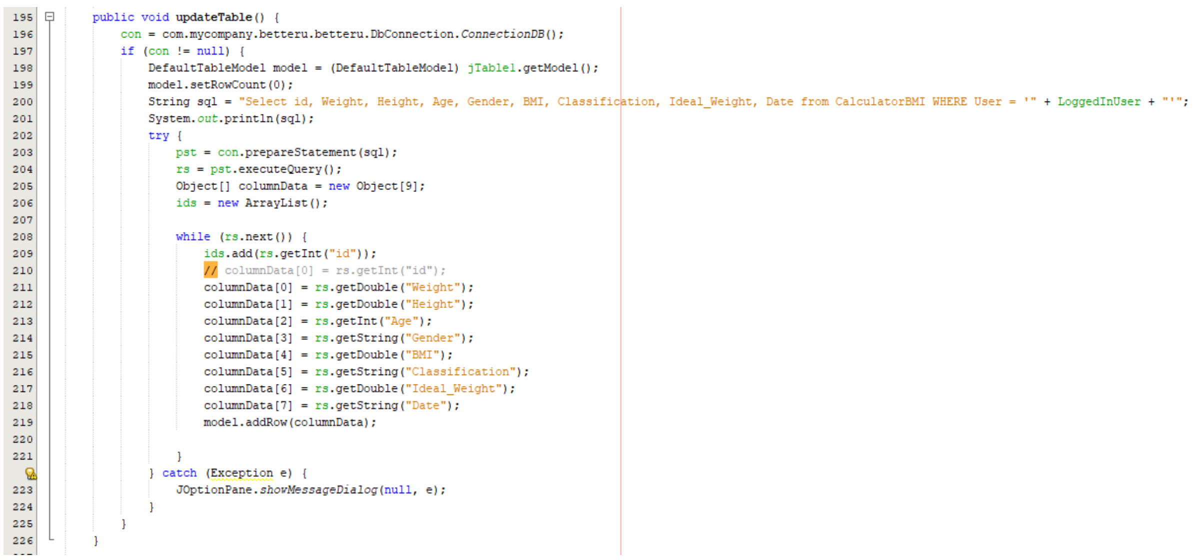
To get the ID for the entry of the table there is an array list declared amongst the local variables. Of the class for viewing the BMI results.



```
Connection con = null;
PreparedStatement pst = null;
ResultSet rs = null;
String LoggedInUser = null;
ArrayList<Integer> ids = null;
```

This Arraylist is then used in the delete button code. In the first two lines of the try catch block line 161-162 the system retrieves the index of the selected row from the table. It then gets the id value from this row. This allows the statement on line 163 to run. It is put together on line 164 and on line 165 the id is plugged into the query. On line 166 the statement is run. After this the user sees a message saying the row has been deleted and then the database connection is closed. The update table method is then executed which I will show next.

The update table method is an important method, and it is called in the constructor of class and anytime the table needs to be updated. I will show the code for the update table method below.



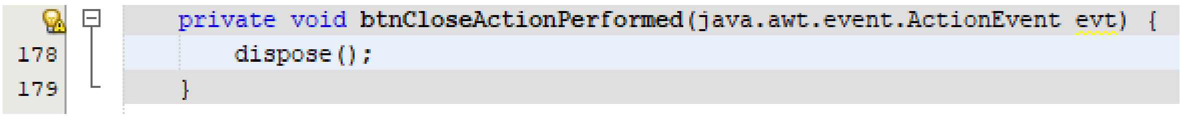
```
195     public void updateTable() {
196         con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
197         if (con != null) {
198             DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
199             model.setRowCount(0);
200             String sql = "Select id, Weight, Height, Age, Gender, BMI, Classification, Ideal_Weight, Date from CalculatorBMI WHERE User = '" + LoggedInUser + "'";
201             System.out.println(sql);
202             try {
203                 pst = con.prepareStatement(sql);
204                 rs = pst.executeQuery();
205                 Object[] columnData = new Object[9];
206                 ids = new ArrayList();
207 
208                 while (rs.next()) {
209                     ids.add(rs.getInt("id"));
210                     // columnData[0] = rs.getInt("id");
211                     columnData[0] = rs.getDouble("Weight");
212                     columnData[1] = rs.getDouble("Height");
213                     columnData[2] = rs.getInt("Age");
214                     columnData[3] = rs.getString("Gender");
215                     columnData[4] = rs.getDouble("BMI");
216                     columnData[5] = rs.getString("Classification");
217                     columnData[6] = rs.getDouble("Ideal_Weight");
218                     columnData[7] = rs.getString("Date");
219                     model.addRow(columnData);
220 
221                 }
                } catch (Exception e) {
223                 JOptionPane.showMessageDialog(null, e);
224             }
225         }
226     }
```

This code establishes the connection to the database in line 196. It then has an if statement to verify the connection to the database. If there is a connection it runs the code inside the if statement. The first thing it does is clear the table this is good practice as it will stop duplicates of entries being seen. This is done on line 198-199. The SQL query is then prepared to select all the information to be displayed in the table. It is important to note the LoggedInUser is used to for the where clause in the SQL where clause is “WHERE User =”. This ensures that the user can only see the results that they have added.

Next the system runs a try block this gets the statement ready. There is an object array declaration and an array list for the id. The while loop goes over the result set to get the values for each column.

It then adds the value for the id to the ids list. This is on line 209. It then populates the object array with the retrieved column values. It then adds the column data to the table using line 219.

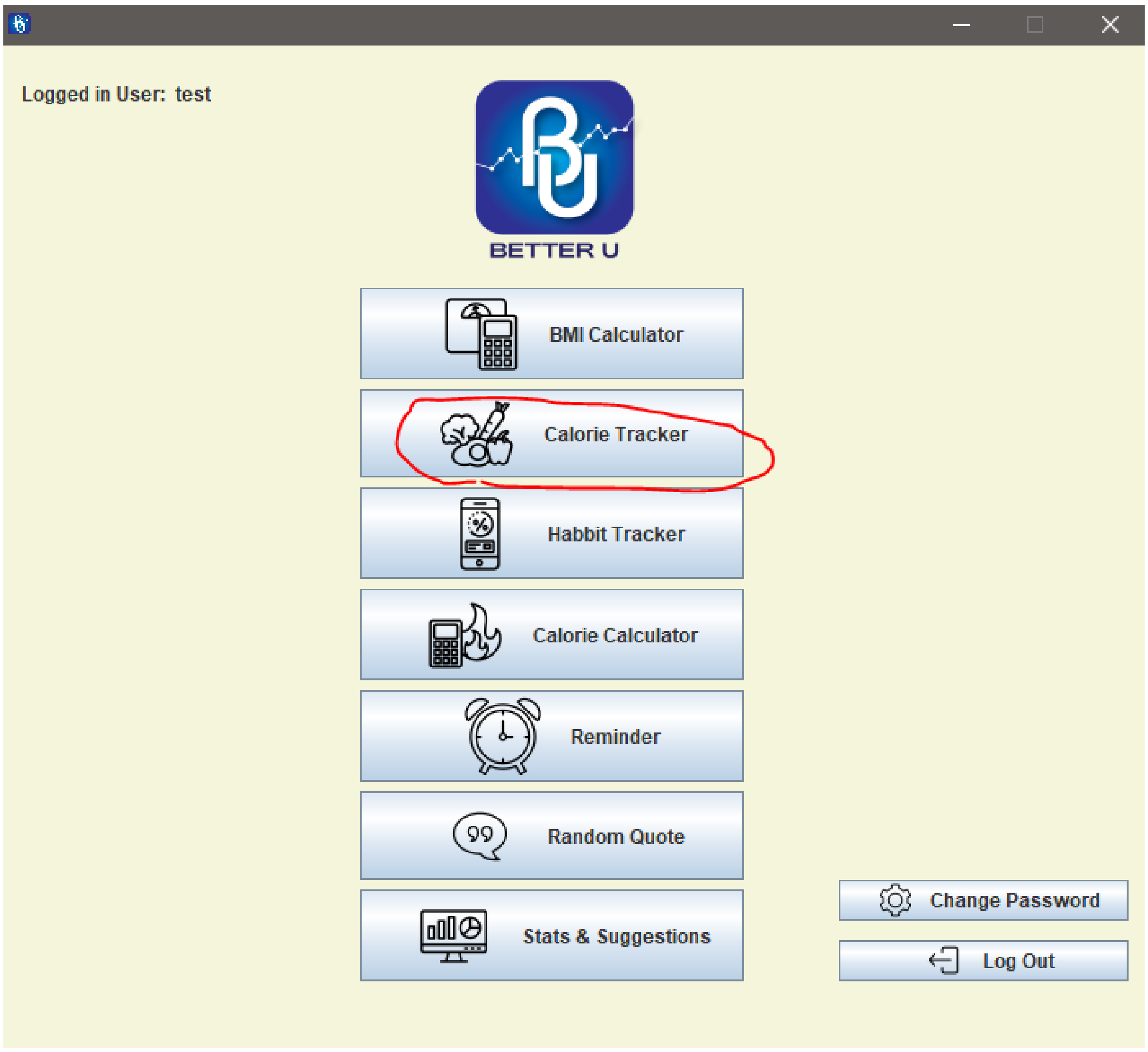
The last thing is the close button which disposes of the view window for the BMI calculator.



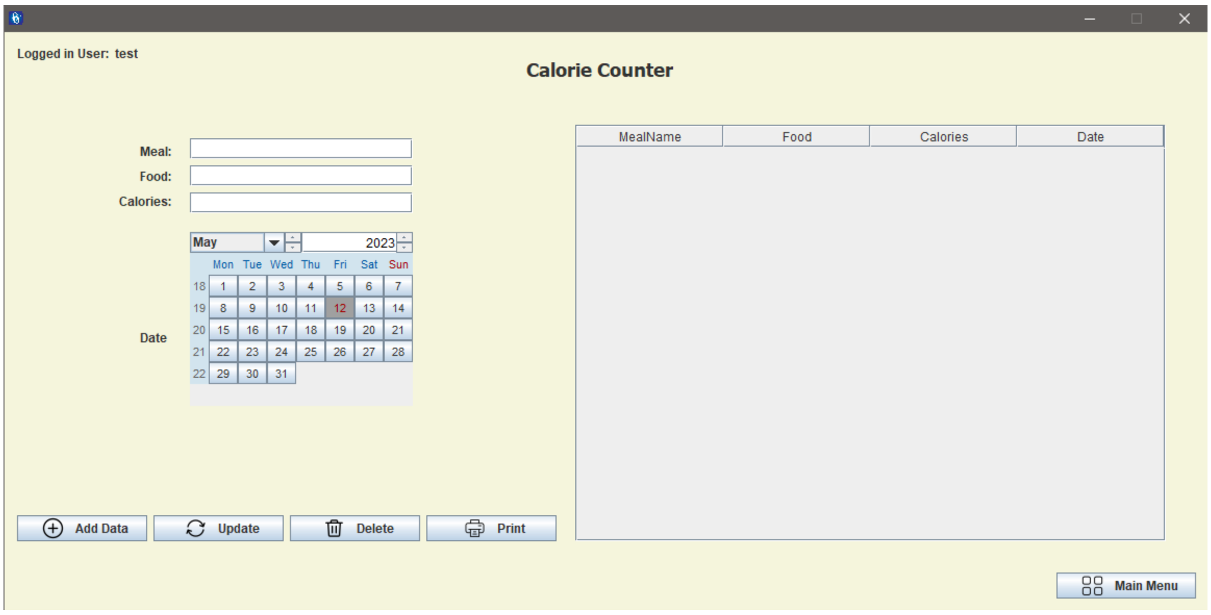
```
    private void btnCloseActionPerformed(java.awt.event.ActionEvent evt) {
        dispose();
    }
```

This concludes the view of the BMI calculator. When the user presses close, they are taken back to the calculator where the user can then click the button to return to the main menu. This is the same code as the close button. As these are new windows being opened the button will close them windows.

Now back at the main menu the next feature is the calorie tracker.



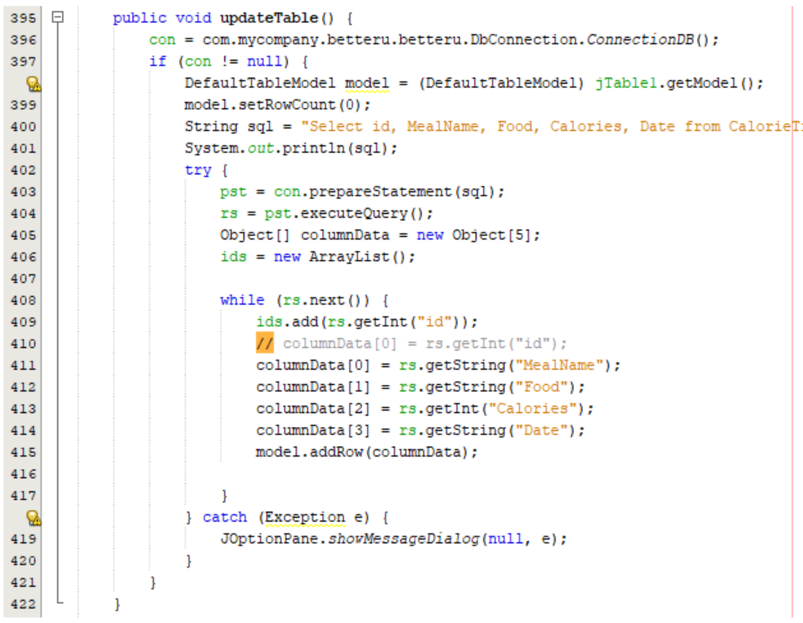
Upon opening the calorie tracker, we can see this window below.



This calorie tracker takes the form of a crud application here we can see the user can enter the name of the meal they are about to eat. They can then enter the food and the calories associated with that food after that the user can use the calendar to select the date they are entering. The user then has 4 buttons below this. These buttons are add, update, delete and print.

The add button adds the information the user has inputted into the Meal, food, calories, and date. The delete buttons works like the view on the BMI calculator. When the table is populated, it allows a user to be able to be able to click on an entry to the table and it will populate the meal, food, calories, and date input field. The user can press delete when an entry is selected, and it will delete the entry. The update works when an entry is selected it will populate the relevant fields, the user can then make an edit to the data and then press update to update the record. The print feature works the same way as the print in the BMI calculator.

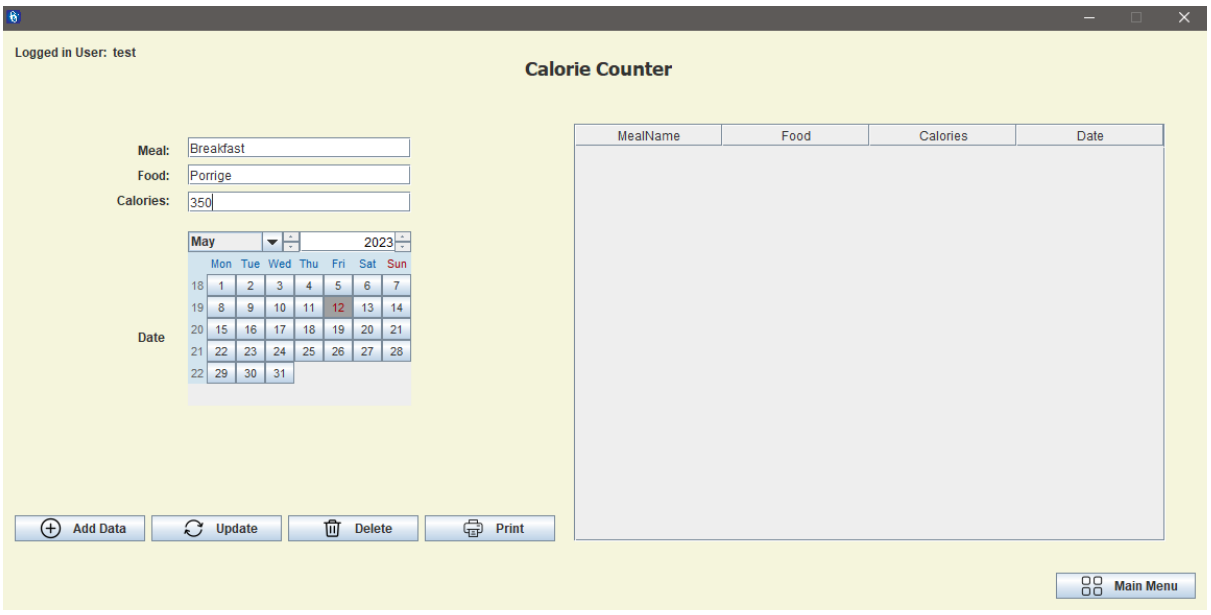
The table is populated using an update Table method. The code for this is seen below.



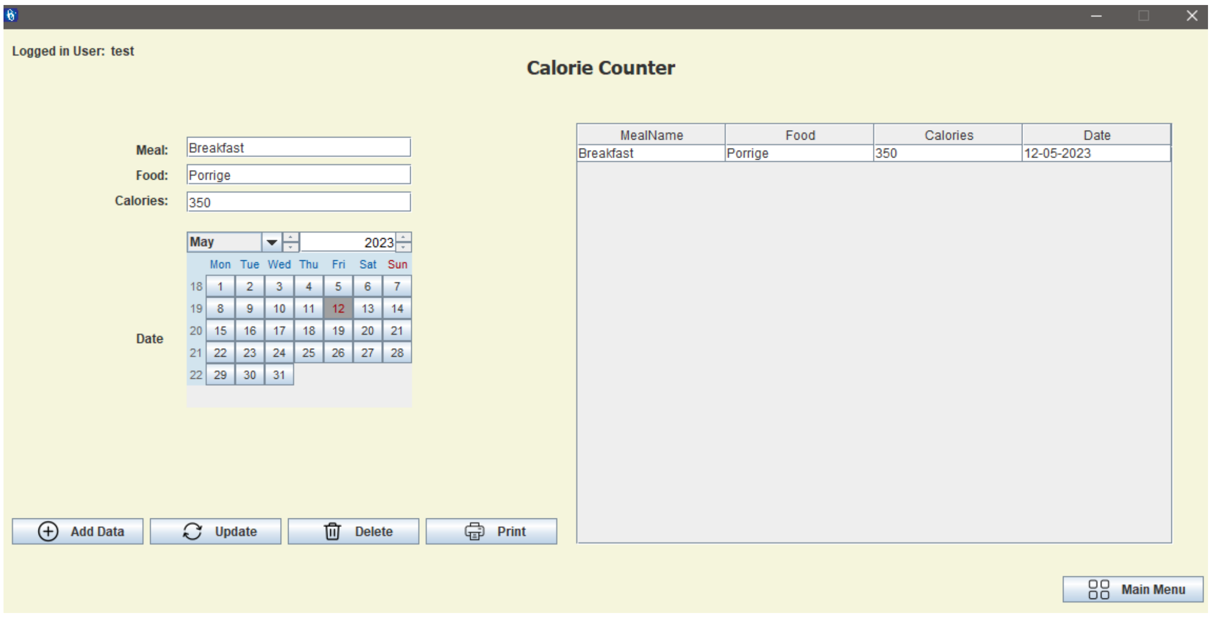
```java
public void updateTable() {
    con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
    if (con != null) {
        DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
        model.setRowCount(0);
        String sql = "Select id, MealName, Food, Calories, Date from CalorieTrack WHERE User = '" + LoggedInUser + "'";
        System.out.println(sql);
        try {
            pst = con.prepareStatement(sql);
            rs = pst.executeQuery();
            Object[] columnData = new Object[5];
            ids = new ArrayList();

            while (rs.next()) {
                ids.add(rs.getInt("id"));
                // columnData[0] = rs.getInt("id");
                columnData[0] = rs.getString("MealName");
                columnData[1] = rs.getString("Food");
                columnData[2] = rs.getInt("Calories");
                columnData[3] = rs.getString("Date");
                model.addRow(columnData);

            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e);
        }
    }
}
```

This code is very similar to the code for the update table in the BMI calculator so I will not go into dept explaining it as there is already an explanation there.

I will add an entry to the table to show what happens and how its done. After the input fields have been entered by the user the user can press the add data button to add the data to the database.



Above is before pressing add. Below is after.

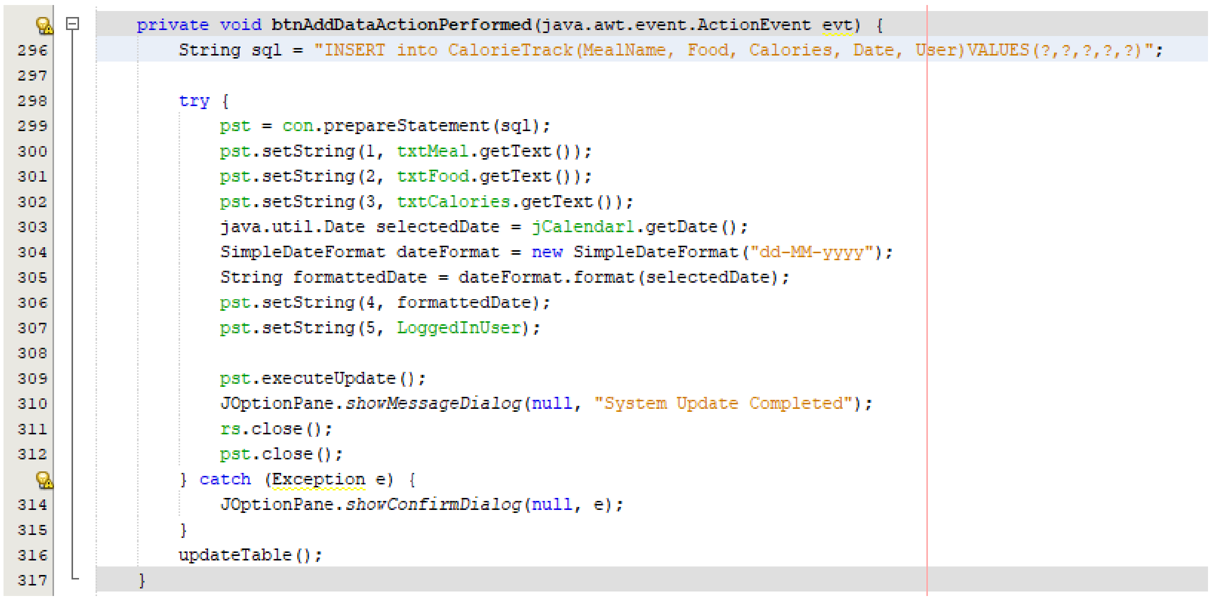

Logged in User: test

Calorie Counter

Meal: Breakfast
Food: Porrige
Calories: 350
Date

MealName	Food	Calories	Date
Breakfast	Porrige	350	12-05-2023

The add button has added the information to the database. The code for the add button is below.

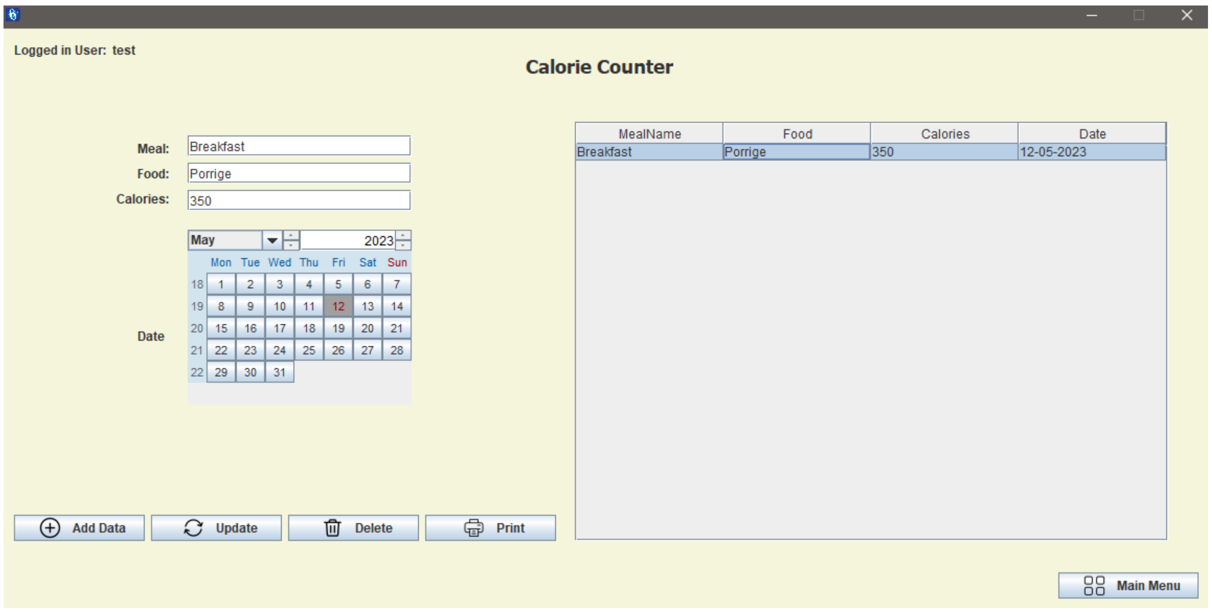

```
    private void btnAddDataActionPerformed(java.awt.event.ActionEvent evt) {
296     String sql = "INSERT into CalorieTrack(MealName, Food, Calories, Date, User)VALUES(?,?,?,?,?)";
297
298     try {
299         pst = con.prepareStatement(sql);
300         pst.setString(1, txtMeal.getText());
301         pst.setString(2, txtFood.getText());
302         pst.setString(3, txtCalories.getText());
303         java.util.Date selectedDate = jCalendar1.getDate();
304         SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
305         String formattedDate = dateFormat.format(selectedDate);
306         pst.setString(4, formattedDate);
307         pst.setString(5, LoggedInUser);
308
309         pst.executeUpdate();
310         JOptionPane.showMessageDialog(null, "System Update Completed");
311         rs.close();
312         pst.close();
        } catch (Exception e) {
314         JOptionPane.showConfirmDialog(null, e);
315     }
316     updateTable();
317 }
```

At the beginning of this on line 296 there is the SQL query getting ready. It is an insert statement which inserts into the CalorieTrack table the meal name the food the calories the date and the user who is logged in.

The try block here runs inserts the information into the database. On line 299-302 it prepares the SQL statement, and it uses prepared statements and gets the information from the inputs of the user. Line 303 gets the date from the calendar the user uses to input the date. Line 304-305 parses the date into the format dd-mm-yyyy. After all the parameters for the query have been set the statement is executed on line 309. Notice that the user is also being added to the database on line 307.

Once the data has been added a dialogue box opens saying the system has been updated. The database connection is then closed on lines 311-312. After this then updates the tale to show the new addition in line 316.

I will now show the update button. To use the update button, you must select the entry.

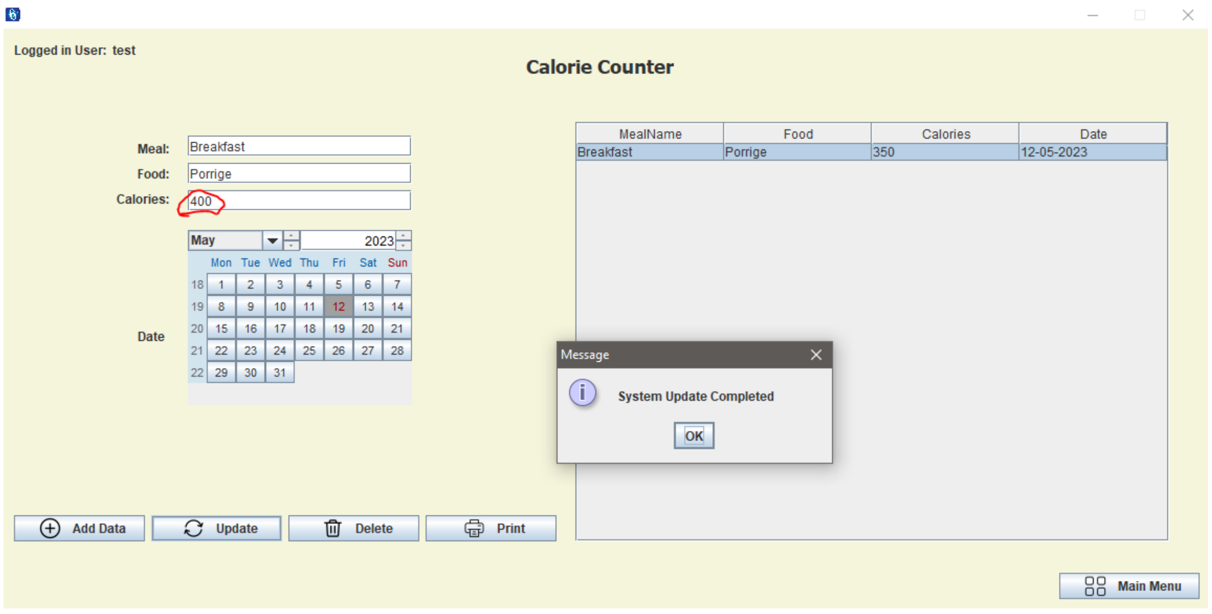

Logged in User: test

Calorie Counter

Meal: Breakfast
Food: Porrige
Calories: 350
Date: May 2023

MealName	Food	Calories	Date
Breakfast	Porrige	350	12-05-2023

Once the entry has been clicked the input fields are populated with the information from that entry. A user can then make a chance and press update. For demonstration purposes I will change the 350 to 400 and press update.


Logged in User: test

Calorie Counter

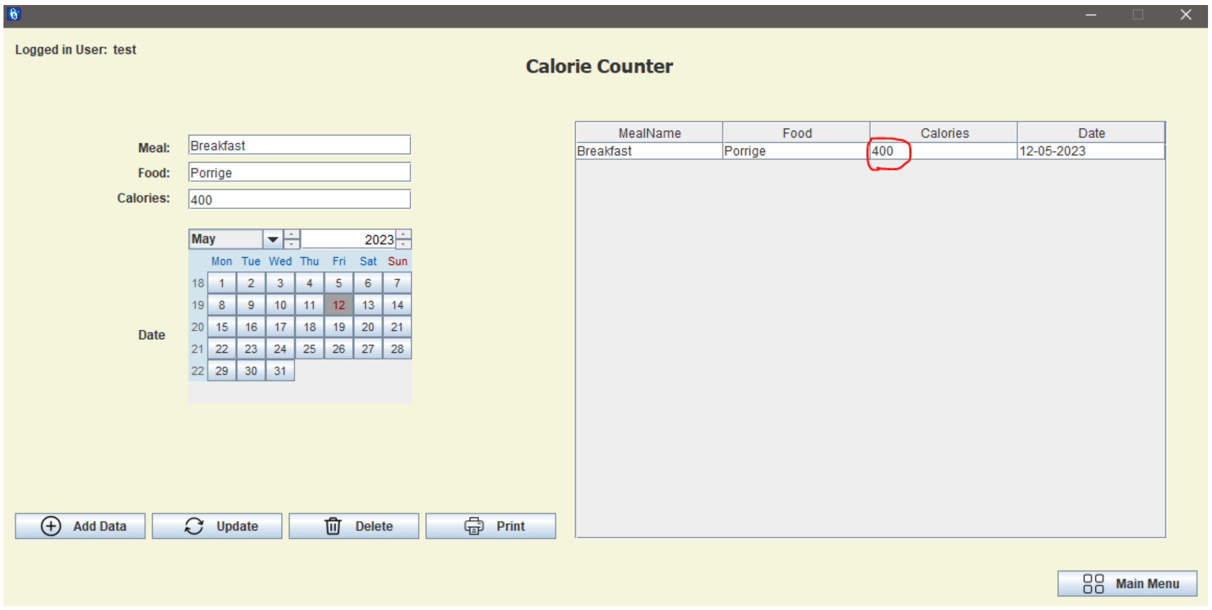
Meal: Breakfast
Food: Porrige
Calories: 400
Date: May 2023

MealName	Food	Calories	Date
Breakfast	Porrige	350	12-05-2023

Message

System Update Completed

This message pops up and upon clicking ok the table refreshes.

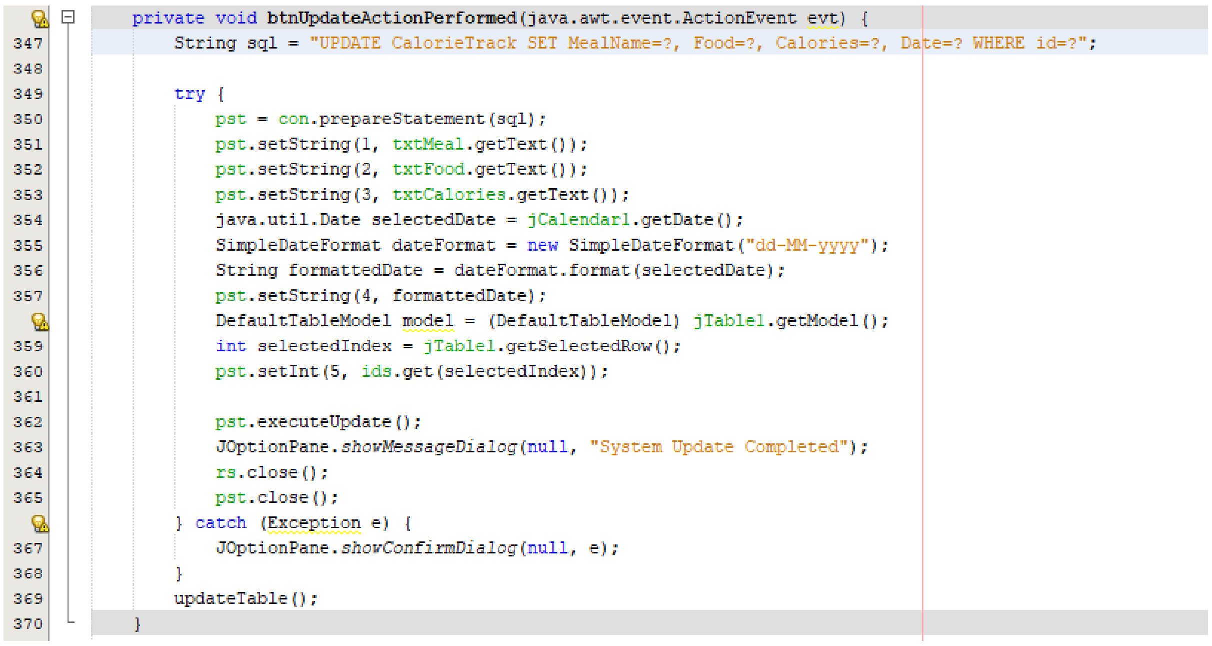

Logged in User: test

Calorie Counter

Meal: Breakfast
Food: Porrige
Calories: 400
Date

MealName	Food	Calories	Date
Breakfast	Porrige	400	12-05-2023

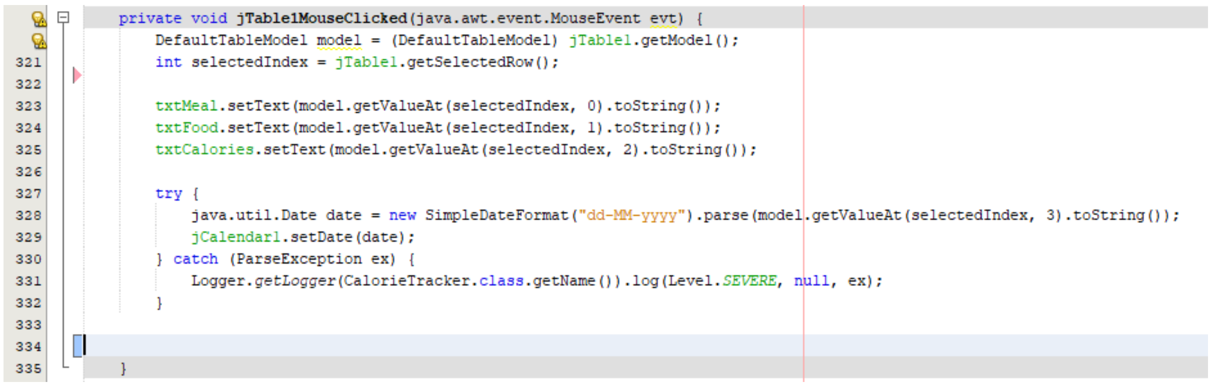
I will now show the code for the update button.


```
    private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {
347     String sql = "UPDATE CalorieTrack SET MealName=?, Food=?, Calories=?, Date=? WHERE id=?";
348
349     try {
350         pst = con.prepareStatement(sql);
351         pst.setString(1, txtMeal.getText());
352         pst.setString(2, txtFood.getText());
353         pst.setString(3, txtCalories.getText());
354         java.util.Date selectedDate = jCalendar1.getDate();
355         SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
356         String formattedDate = dateFormat.format(selectedDate);
357         pst.setString(4, formattedDate);
            DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
359         int selectedIndex = jTable1.getSelectedRow();
360         pst.setInt(5, ids.get(selectedIndex));
361
362         pst.executeUpdate();
363         JOptionPane.showMessageDialog(null, "System Update Completed");
364         rs.close();
365         pst.close();
        } catch (Exception e) {
367         JOptionPane.showConfirmDialog(null, e);
368     }
369     updateTable();
370 }
```

This code is very much the same as the add button so I won't explain it all I will just explain where it differs from the add button. As seen in line 347 the SQL statement start with UPDATE instead of INSERT. It does this as we do not want to add a new entry, we just simply want to update an entry. In the statement it says to update meal name, food, calories, and date and they are all equal to a question mark. These question marks have been seen in previous statements and they represent the values that will be plugged in. Notice it also uses the id field for the update. This is done because the id field is unique to each entry so that's why it says update WHERE id=?;

The ID is got from the lines 359-360 from the selected row.

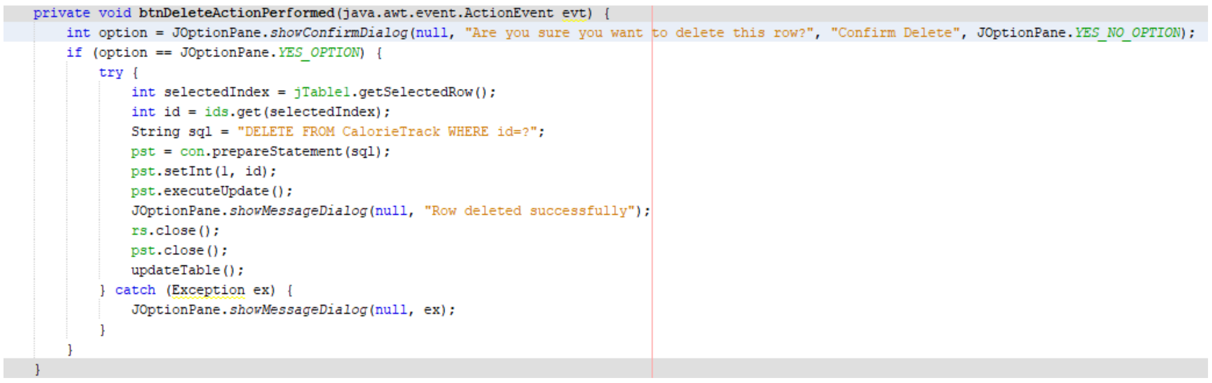
That's it for the update button I will now show the code for making the table selectable and able to populate the input fields upon selection.



```
        private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
            DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
321         int selectedIndex = jTable1.getSelectedRow();
322
323         txtMeal.setText(model.getValueAt(selectedIndex, 0).toString());
324         txtFood.setText(model.getValueAt(selectedIndex, 1).toString());
325         txtCalories.setText(model.getValueAt(selectedIndex, 2).toString());
326
327         try {
328             java.util.Date date = new SimpleDateFormat("dd-MM-yyyy").parse(model.getValueAt(selectedIndex, 3).toString());
329             jCalendar1.setDate(date);
330         } catch (ParseException ex) {
331             Logger.getLogger(CalorieTracker.class.getName()).log(Level.SEVERE, null, ex);
332         }
333
334
335     }
```

The first two lines let the table be able be selected. The lines 323-332 add the values from the table to the input fields. The first three the meal name the food and the calories are done quick the date however has to be parsed back into the format the java date works with.

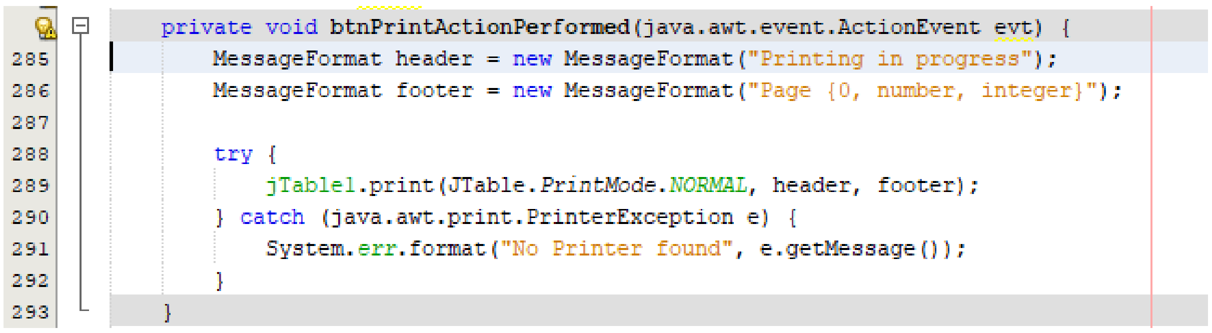
I will now show the code for the delete and the print buttons. Below is delete.



```
    private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        int option = JOptionPane.showConfirmDialog(null, "Are you sure you want to delete this row?", "Confirm Delete", JOptionPane.YES_NO_OPTION);
        if (option == JOptionPane.YES_OPTION) {
            try {
                int selectedIndex = jTable1.getSelectedRow();
                int id = ids.get(selectedIndex);
                String sql = "DELETE FROM CalorieTrack WHERE id=?";
                pst = con.prepareStatement(sql);
                pst.setInt(1, id);
                pst.executeUpdate();
                JOptionPane.showMessageDialog(null, "Row deleted successfully");
                rs.close();
                pst.close();
                updateTable();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, ex);
            }
        }
    }
```

The delete button is very much the same as the delete in the view BMI calculator so I will not explain or demonstrate again.

Below is the Print button code. This too is very much the same as the view BMI calculator so I will not explain or demonstrate again.

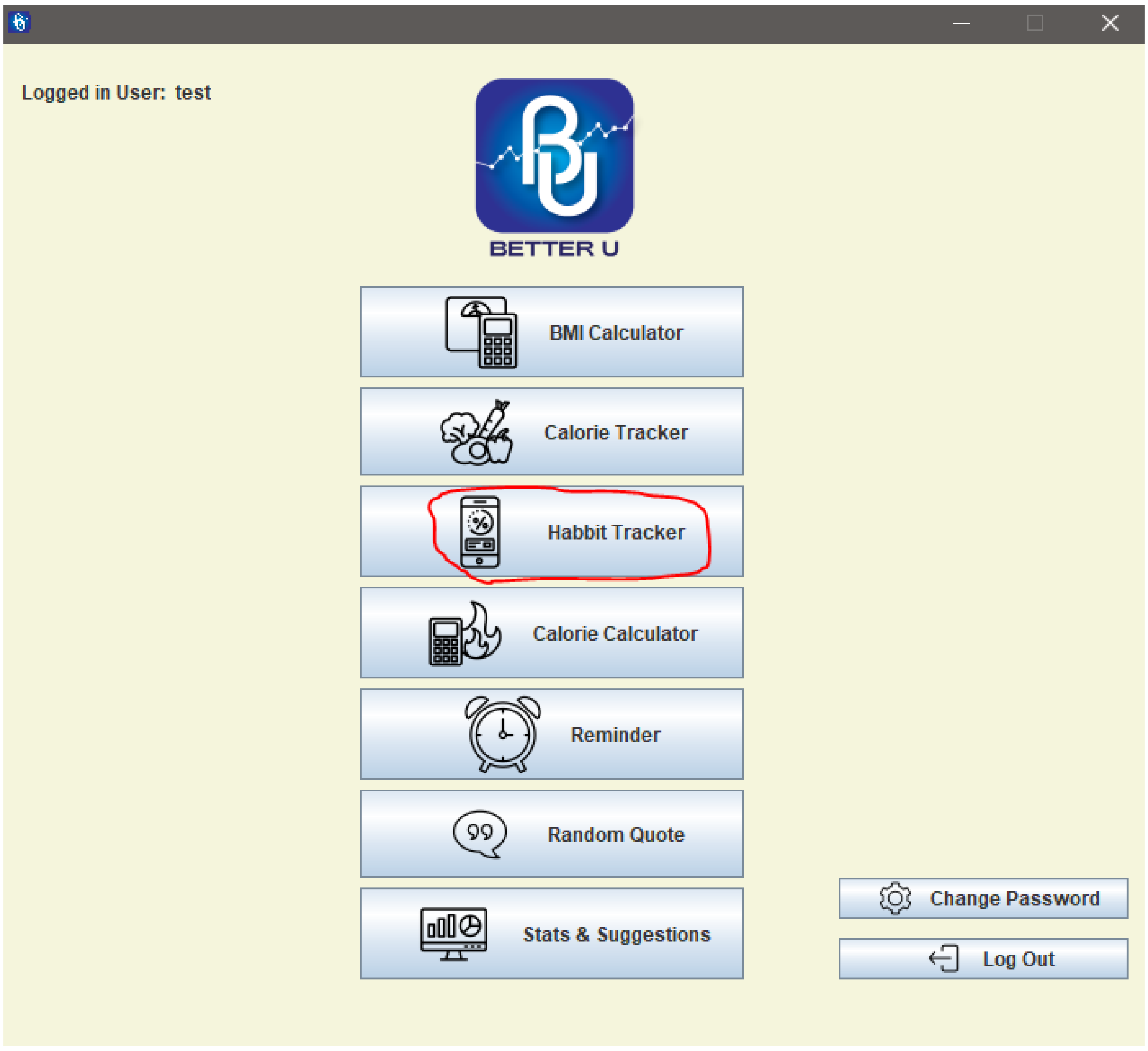


```
    private void btnPrintActionPerformed(java.awt.event.ActionEvent evt) {
        MessageFormat header = new MessageFormat("Printing in progress");
        MessageFormat footer = new MessageFormat("Page {0, number, integer}");

        try {
            jTable1.print(JTable.PrintMode.NORMAL, header, footer);
        } catch (java.awt.print.PrinterException e) {
            System.err.format("No Printer found", e.getMessage());
        }
    }
```

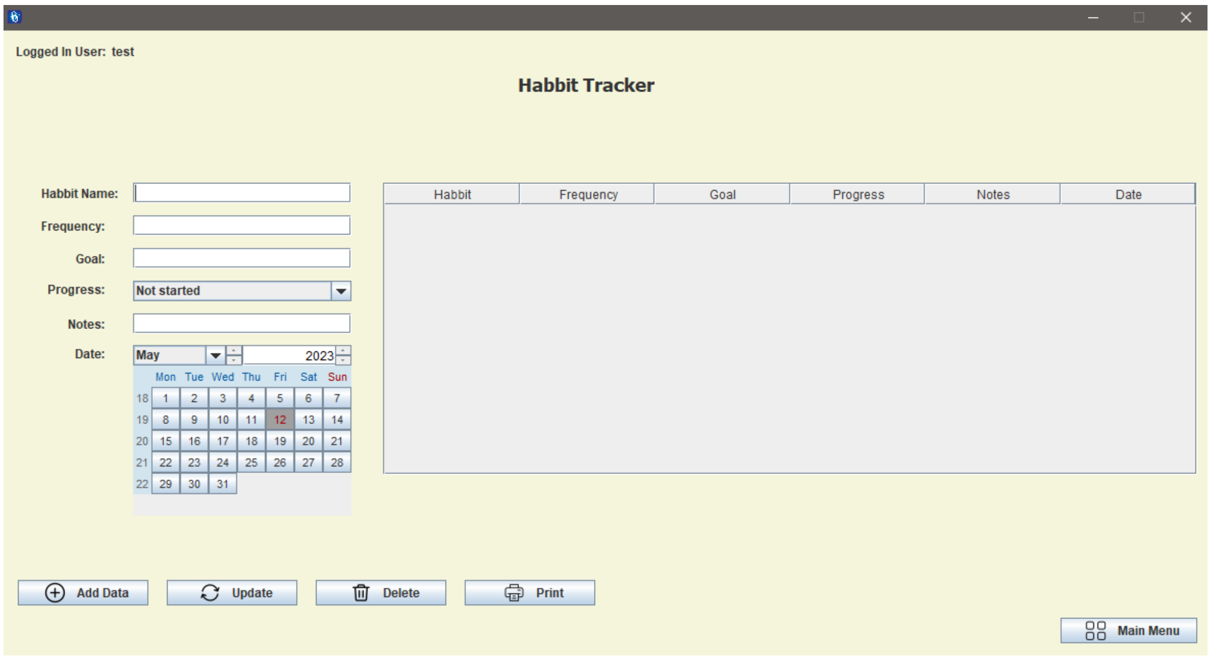
This concludes the Calorie counter.

The user can then press the main menu button to go back to the main menu.

Next its time for the habit tracker

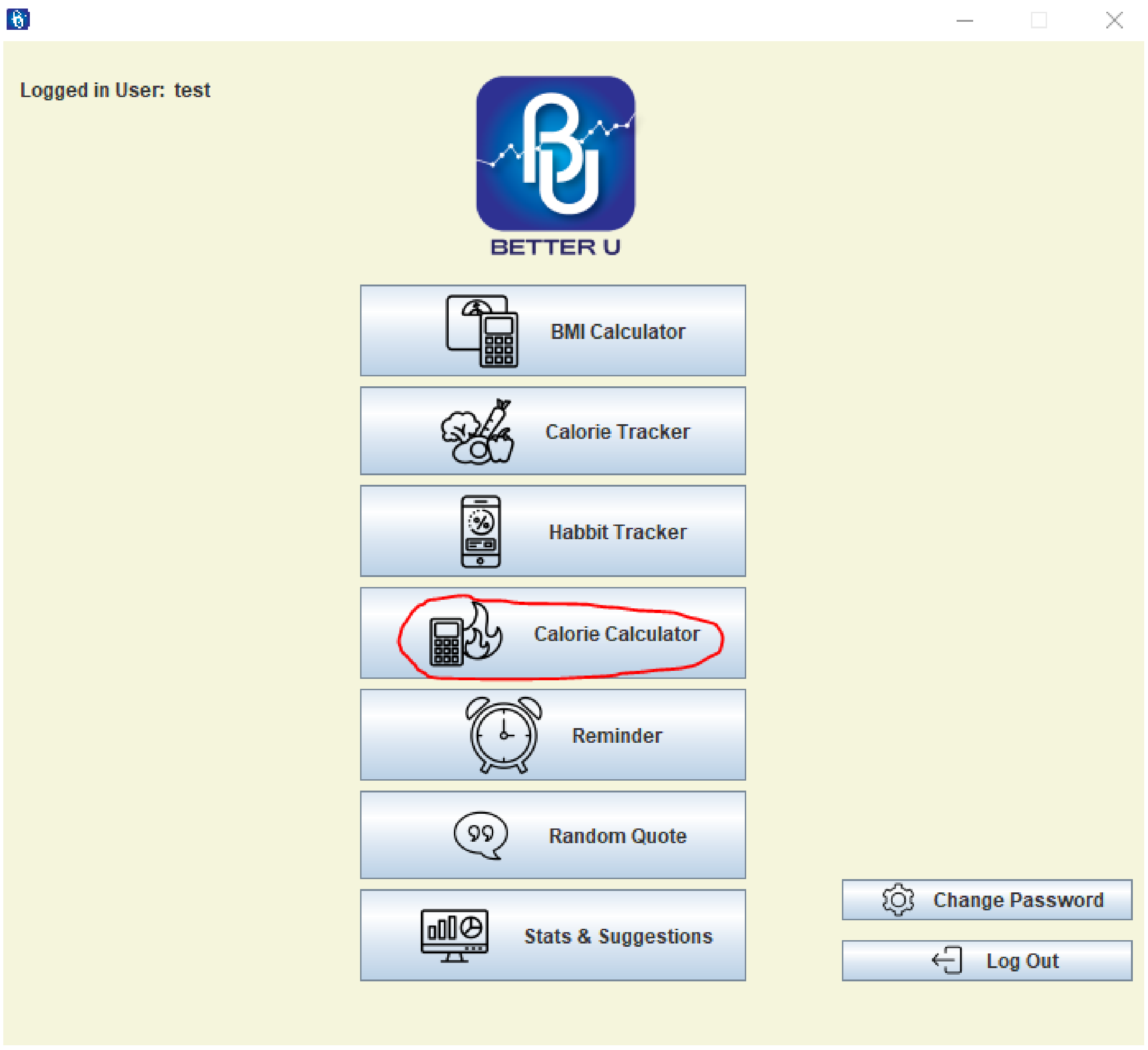


Upon Opening up the habit tracker this is what the user sees

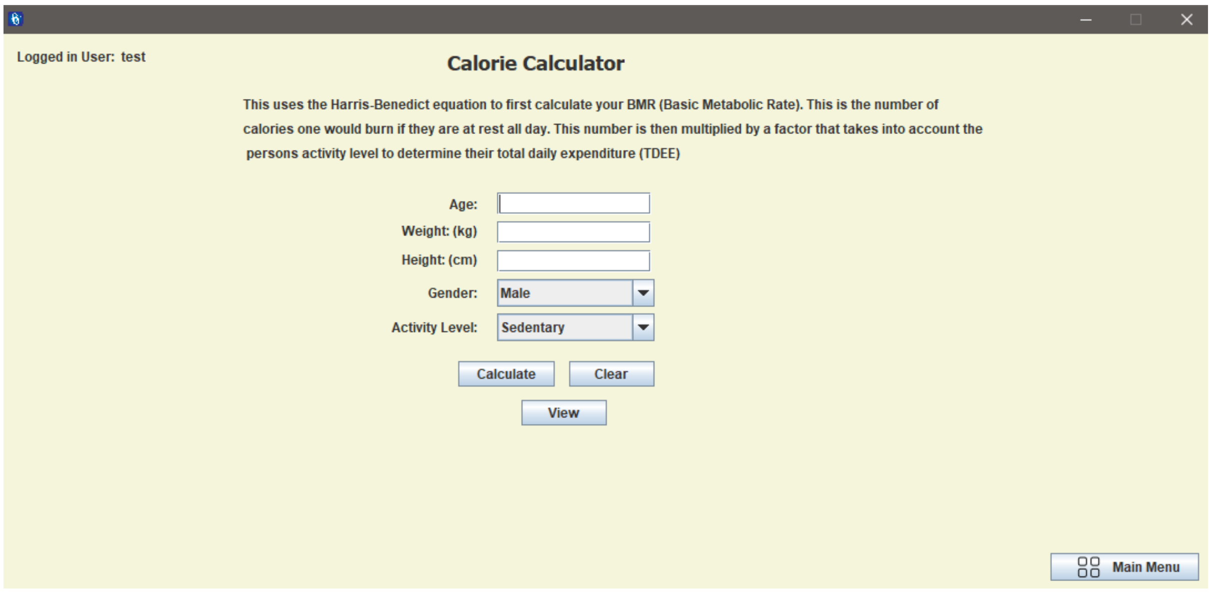


It is also a crud style application. This is the best fit for a habit tracker feature. It is very similar to the Calorie tracker so I will not explain the code for this feature.

I will now move onto the next feature. The next feature is the calorie calculator.



Upon clicking on the calorie calculator button, the calorie calculator opens. Below is a screenshot.


Logged in User: test

Calorie Calculator

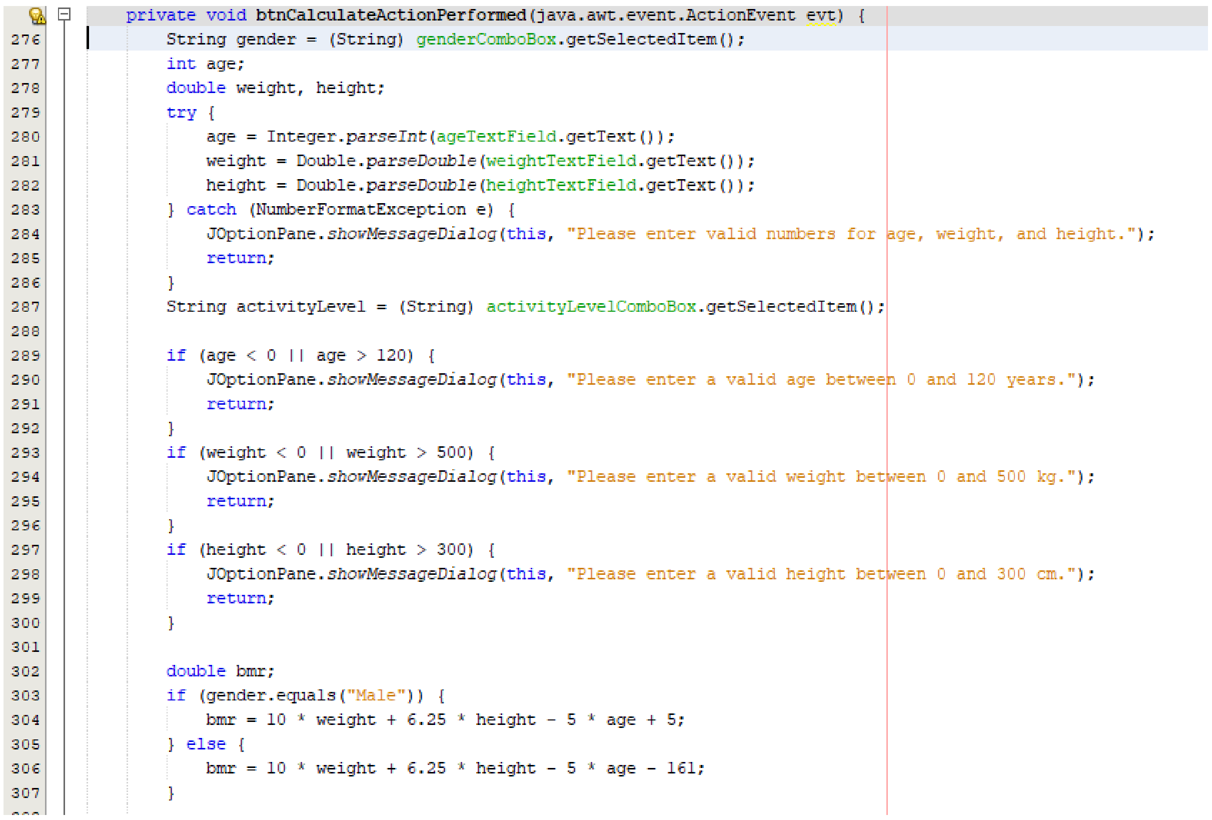
This uses the Harris-Benedict equation to first calculate your BMR (Basic Metabolic Rate). This is the number of calories one would burn if they are at rest all day. This number is then multiplied by a factor that takes into account the persons activity level to determine their total daily expenditure (TDEE)

Age:
Weight: (kg)
Height: (cm)
Gender: Male
Activity Level: Sedentary

Upon opening the calorie calculator, the user is presented with some text giving a brief explanation of how the feature work. It states that this calculator uses the Harris benedict equation to first calculate the users BMR which is the users' basic metabolic rate. This is the number of calories a user would consume if they were at res all day. The number is then multiplied by a factor that considers the persons activity level. This then results in the user's total daily expenditure.

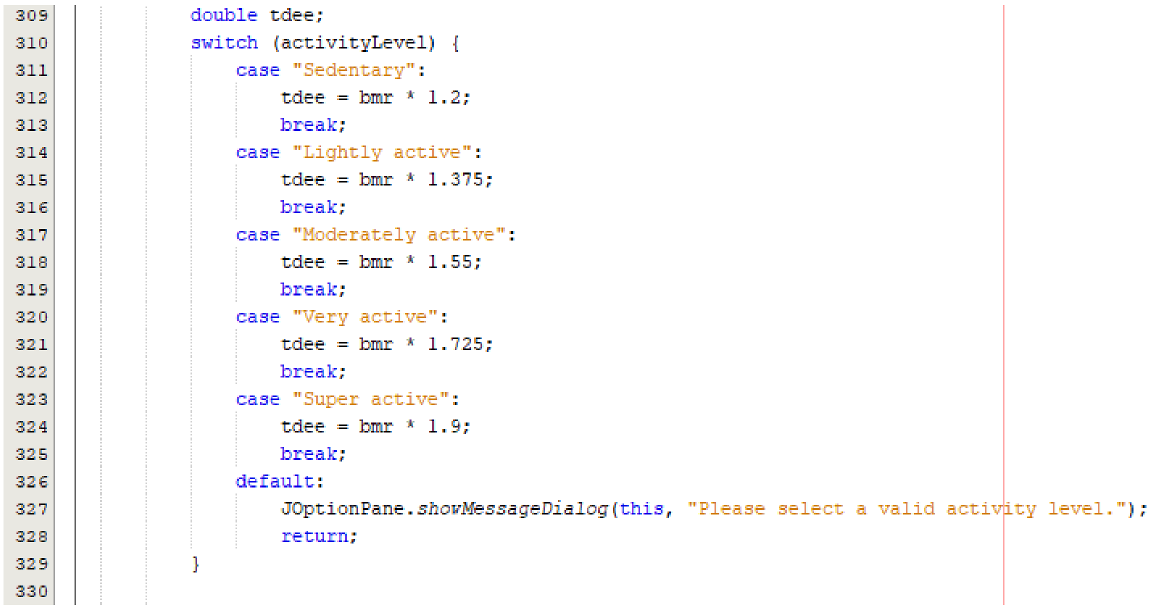
This calorie calculator calculates the number of calories the user needs to consume in a day.

The user is presented with 3 fields and 2 combo boxes upon opening the feature. The clear and view I have described in previous features so I will just show the calculate button which does the calculation and saving to the database. There is 3 screenshots for this button as it contains a large amount of code I will describe the code in each screenshot below the screenshot



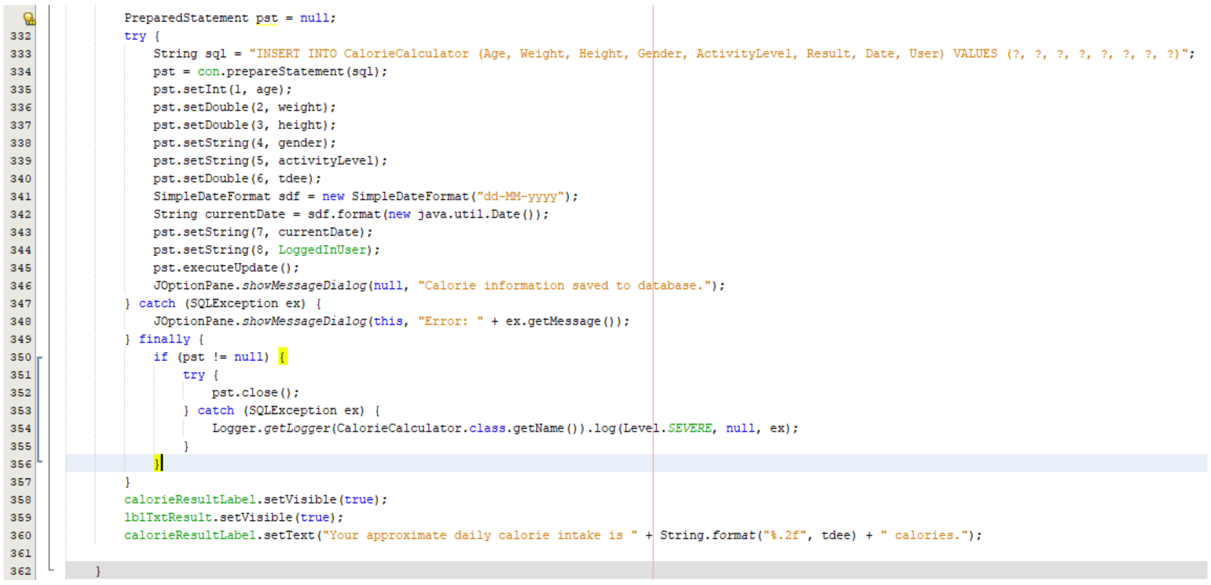
```
    private void btnCalculateActionPerformed(java.awt.event.ActionEvent evt) {
276     String gender = (String) genderComboBox.getSelectedItem();
277     int age;
278     double weight, height;
279     try {
280         age = Integer.parseInt(ageTextField.getText());
281         weight = Double.parseDouble(weightTextField.getText());
282         height = Double.parseDouble(heightTextField.getText());
283     } catch (NumberFormatException e) {
284         JOptionPane.showMessageDialog(this, "Please enter valid numbers for age, weight, and height.");
285         return;
286     }
287     String activityLevel = (String) activityLevelComboBox.getSelectedItem();
288 
289     if (age < 0 || age > 120) {
290         JOptionPane.showMessageDialog(this, "Please enter a valid age between 0 and 120 years.");
291         return;
292     }
293     if (weight < 0 || weight > 500) {
294         JOptionPane.showMessageDialog(this, "Please enter a valid weight between 0 and 500 kg.");
295         return;
296     }
297     if (height < 0 || height > 300) {
298         JOptionPane.showMessageDialog(this, "Please enter a valid height between 0 and 300 cm.");
299         return;
300     }
301 
302     double bmr;
303     if (gender.equals("Male")) {
304         bmr = 10 * weight + 6.25 * height - 5 * age + 5;
305     } else {
306         bmr = 10 * weight + 6.25 * height - 5 * age - 161;
307     }
```

In this screenshot we can see that there is age, weight and gender variables set up. Gender is being got from the gender combo box. There is then a try block which takes in the age, weight, and height of the user. These are in a try catch block as the catch catches a number format exception this is done for validation purposes. The activity level then then got from the activity level combo box on line 287. Lines 289-300 is validating the users input ensuring that they do not type in ridiculous values. Next the BMR for the user is being calculated in lines 302-307 this is done by determining if the user selected male or female into the combo box for gender. It then plugs the weight and the height of the user into the formula to calculate the basic metabolic rate.



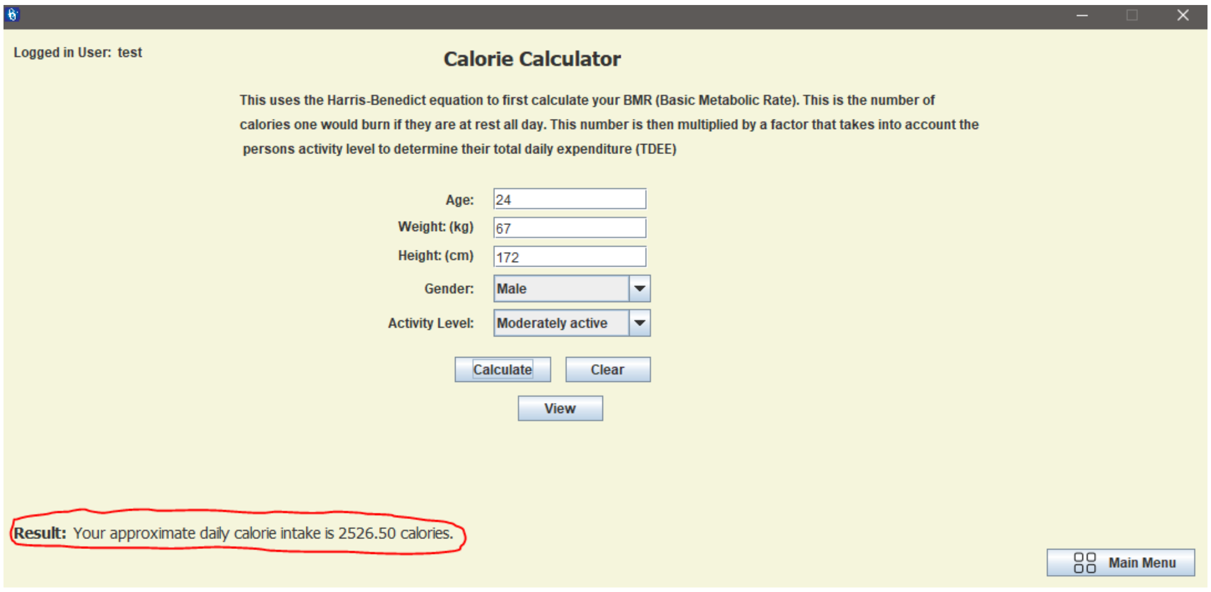
```
309     double tdee;
310     switch (activityLevel) {
311         case "Sedentary":
312             tdee = bmr * 1.2;
313             break;
314         case "Lightly active":
315             tdee = bmr * 1.375;
316             break;
317         case "Moderately active":
318             tdee = bmr * 1.55;
319             break;
320         case "Very active":
321             tdee = bmr * 1.725;
322             break;
323         case "Super active":
324             tdee = bmr * 1.9;
325             break;
326         default:
327             JOptionPane.showMessageDialog(this, "Please select a valid activity level.");
328             return;
329     }
330 
```

In the above screenshot the total daily expenditure is being calculated. This is done using a switch statement. If the user selects either sedentary, lightly active, moderately active, very active or super active the BMR is multiplied by a different figure depending on which option was chosen. If a user does not select a valid activity level a dialogue box appears telling them to select a valid activity level.



```
        PreparedStatement pst = null;
332     try {
333         String sql = "INSERT INTO CalorieCalculator (Age, Weight, Height, Gender, ActivityLevel, Result, Date, User) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
334         pst = con.prepareStatement(sql);
335         pst.setInt(1, age);
336         pst.setDouble(2, weight);
337         pst.setDouble(3, height);
338         pst.setString(4, gender);
339         pst.setString(5, activityLevel);
340         pst.setDouble(6, tdee);
341         SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
342         String currentDate = sdf.format(new java.util.Date());
343         pst.setString(7, currentDate);
344         pst.setString(8, LoggedInUser);
345         pst.executeUpdate();
346         JOptionPane.showMessageDialog(null, "Calorie information saved to database.");
347     } catch (SQLException ex) {
348         JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
349     } finally {
350         if (pst != null) {
351             try {
352                 pst.close();
353             } catch (SQLException ex) {
354                 Logger.getLogger(CalorieCalculator.class.getName()).log(Level.SEVERE, null, ex);
355             }
356         }
357     }
358     calorieResultLabel.setVisible(true);
359     lblTxtResult.setVisible(true);
360     calorieResultLabel.setText("Your approximate daily calorie intake is " + String.format("%.2f", tdee) + " calories.");
361
362 }
```

From line 331 to 357 this code saves the input and result to the database. I will not explain this as it has been explained previously for other files. It follows the same method. In lines 358-359 the labels for the result are made visible and at line 360 the result is printed. I will show the feature working below.



Logged in User: test

Calorie Calculator

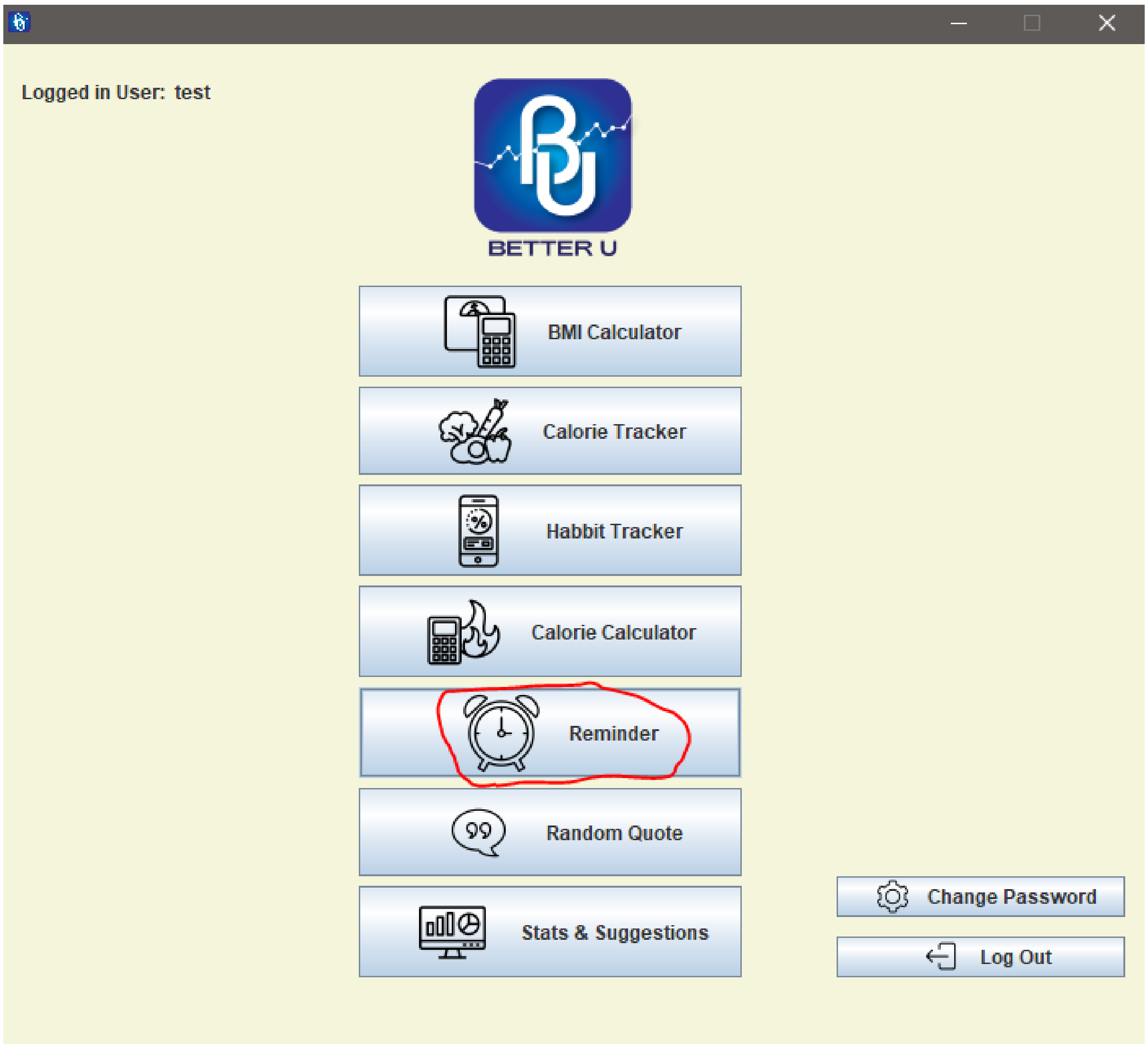
This uses the Harris-Benedict equation to first calculate your BMR (Basic Metabolic Rate). This is the number of calories one would burn if they are at rest all day. This number is then multiplied by a factor that takes into account the persons activity level to determine their total daily expenditure (TDEE)

Age: 24
Weight: (kg) 67
Height: (cm) 172
Gender: Male
Activity Level: Moderately active

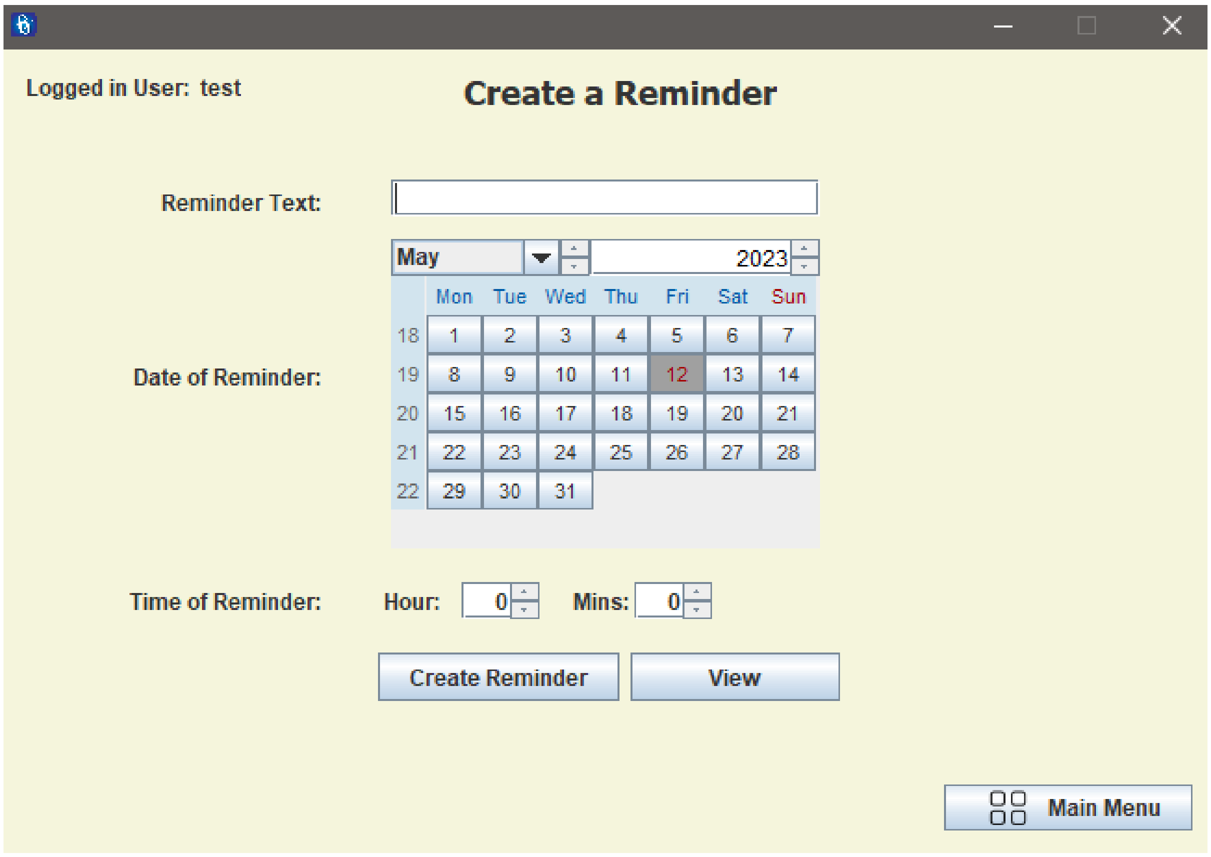
Result: Your approximate daily calorie intake is 2526.50 calories.

After plugging in the information and pressing submit there is a dialogue that says that the calorie information has been saved to the database. The result then appears at the bottom left of the window. This concludes the calorie calculator.

I will now move onto the next feature this is the reminder feature.



Upon clicking the reminder button, the user is taken to the reminder feature.


Logged in User: test

Create a Reminder

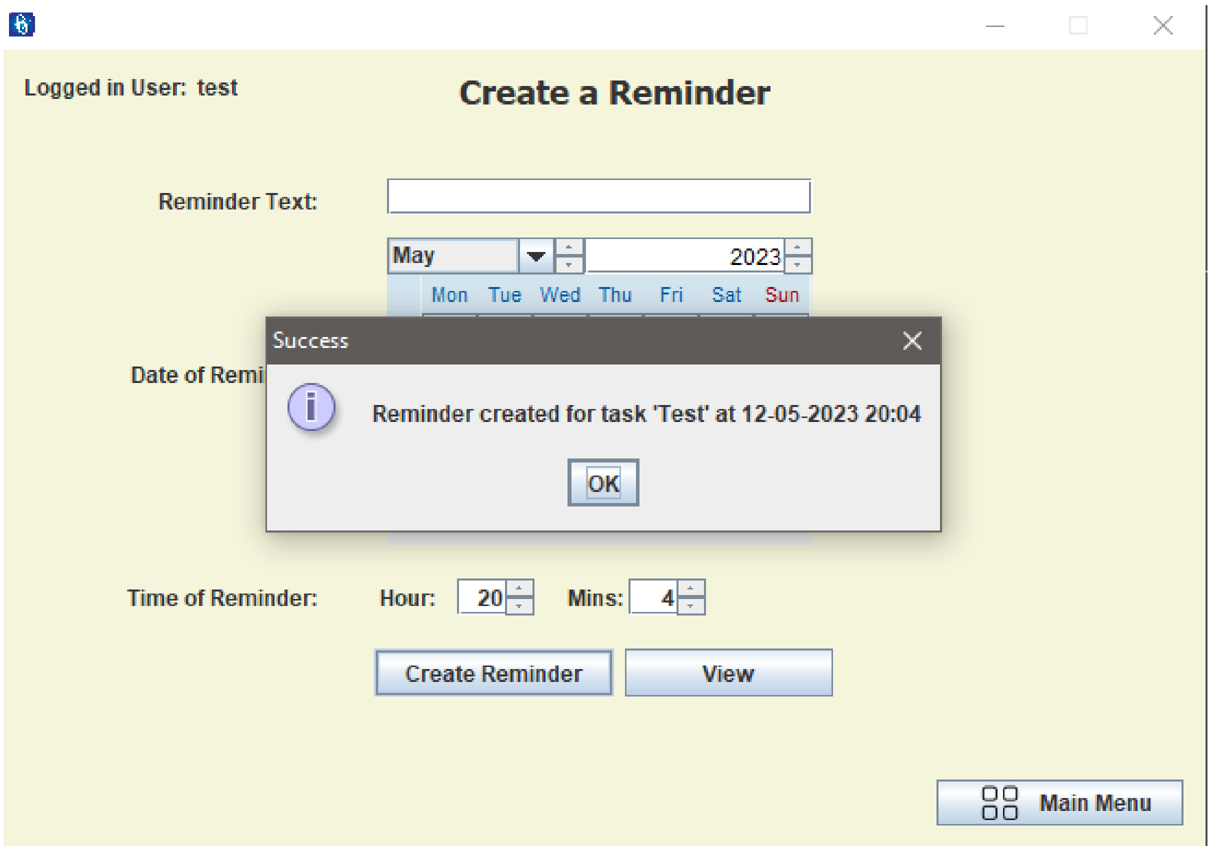
Reminder Text:

Date of Reminder: May 2023

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

Time of Reminder: Hour: 0 Mins: 0

To set a reminder the user adds some reminder text and the date and time of the reminder. Once this is entered the user can press create reminder.


Logged in User: test

Create a Reminder

Reminder Text:

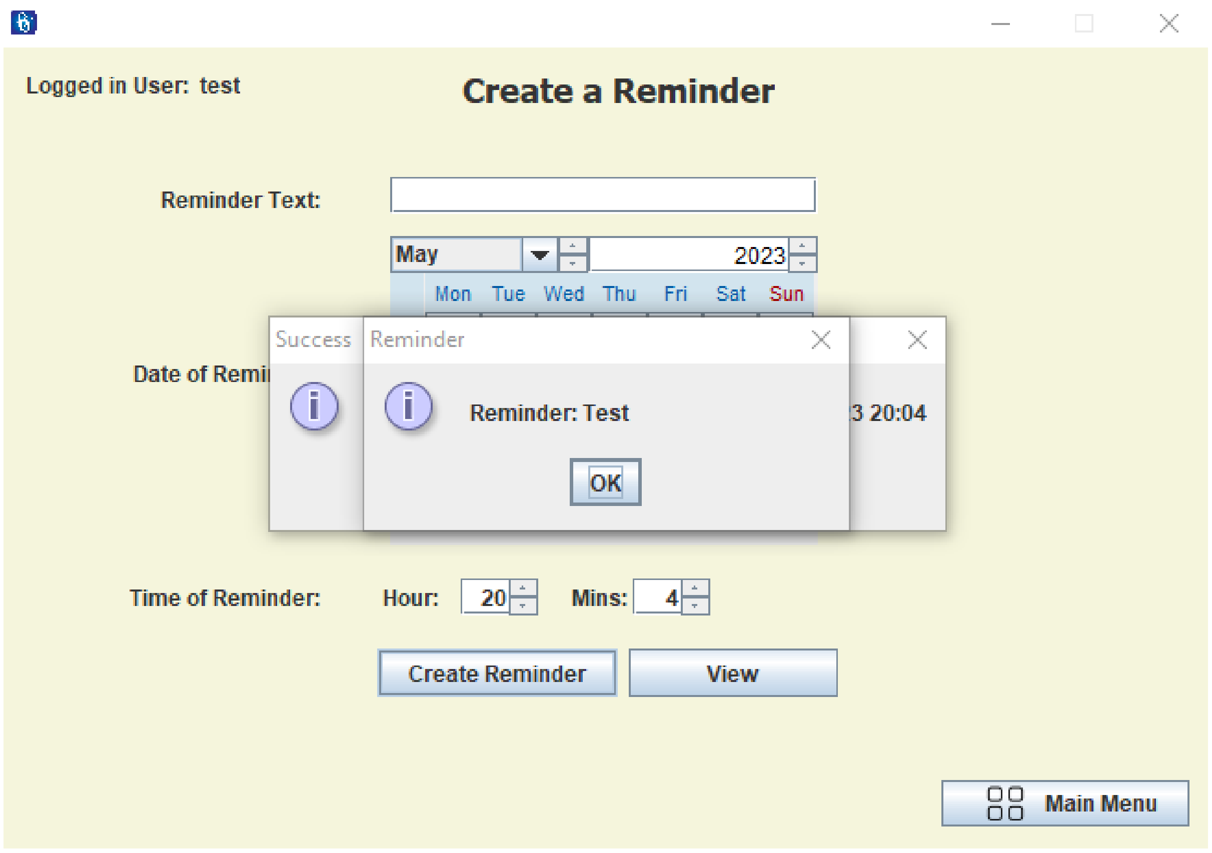
Date of Reminder: May 2023

Success

Reminder created for task 'Test' at 12-05-2023 20:04

Time of Reminder: Hour: 20 Mins: 4

As seen in the above screenshot the reminder has been created. I will show below what happens when the reminder goes off.


Logged in User: test

Create a Reminder

Reminder Text:

May 2023

Mon Tue Wed Thu Fri Sat Sun

Date of Remi

Success

Reminder

Reminder: Test

OK

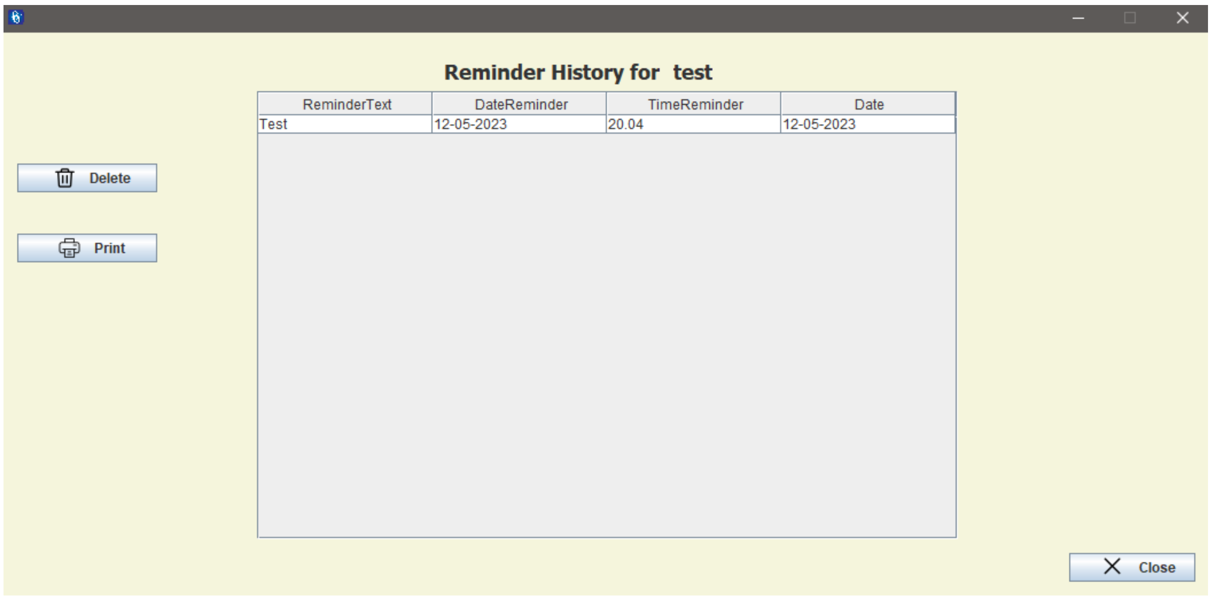
3 20:04

Time of Reminder: Hour: 20 Mins: 4

Create Reminder View

Main Menu

The reminder gets saved to the database upon creation.


Reminder History for test

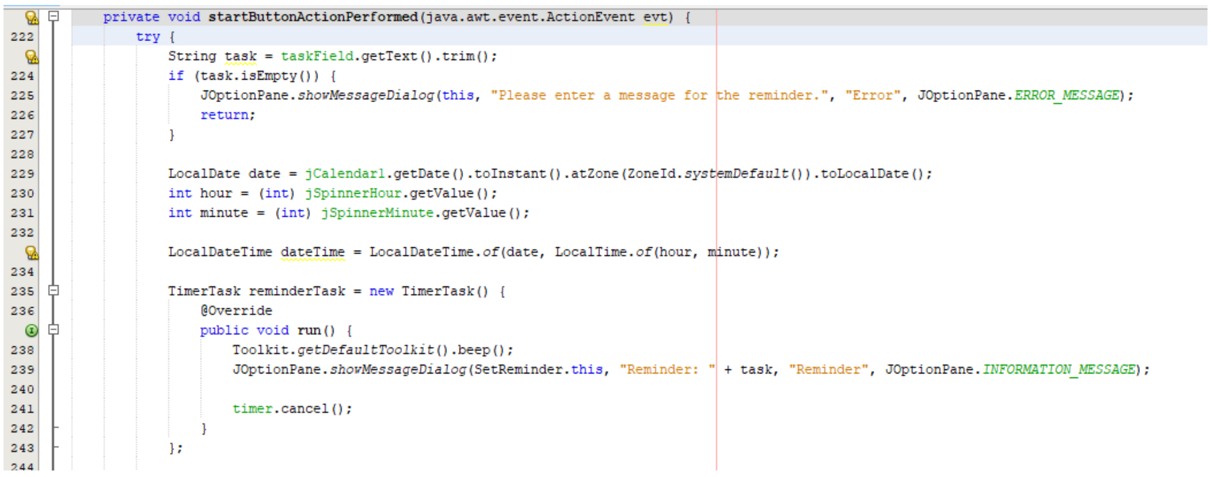
ReminderText	DateReminder	TimeReminder	Date
Test	12-05-2023	20.04	12-05-2023

Delete

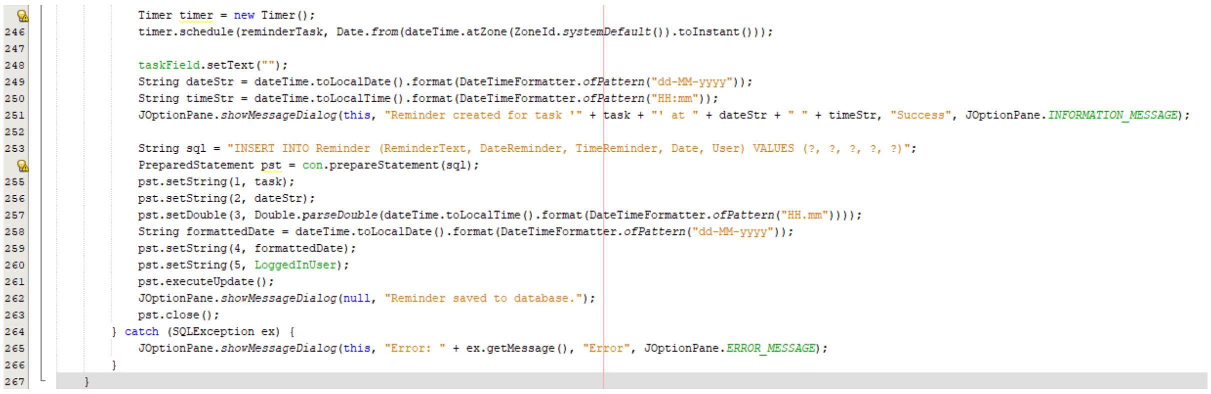
Print

Close

Here is the code for the reminder below



```
        private void startButtonActionPerformed(java.awt.event.ActionEvent evt) {
222         try {
                String task = taskField.getText().trim();
224             if (task.isEmpty()) {
225                 JOptionPane.showMessageDialog(this, "Please enter a message for the reminder.", "Error", JOptionPane.ERROR_MESSAGE);
226                 return;
227             }
228
229             LocalDate date = jCalendar1.getDate().toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
230             int hour = (int) jSpinnerHour.getValue();
231             int minute = (int) jSpinnerMinute.getValue();
232
                LocalDateTime dateTime = LocalDateTime.of(date, LocalTime.of(hour, minute));
234
235             TimerTask reminderTask = new TimerTask() {
236                 @Override
                    public void run() {
238                     Toolkit.getDefaultToolkit().beep();
239                     JOptionPane.showMessageDialog(SetReminder.this, "Reminder: " + task, "Reminder", JOptionPane.INFORMATION_MESSAGE);
240
241                     timer.cancel();
242                 }
243             };
244
```



```
                Timer timer = new Timer();
246             timer.schedule(reminderTask, Date.from(dateTime.atZone(ZoneId.systemDefault()).toInstant()));
247
248             taskField.setText("");
249             String dateStr = dateTime.toLocalDate().format(DateTimeFormatter.ofPattern("dd-MM-yyyy"));
250             String timeStr = dateTime.toLocalTime().format(DateTimeFormatter.ofPattern("HH:mm"));
251             JOptionPane.showMessageDialog(this, "Reminder created for task '" + task + "' at " + dateStr + " " + timeStr, "Success", JOptionPane.INFORMATION_MESSAGE);
252
253             String sql = "INSERT INTO Reminder (ReminderText, DateReminder, TimeReminder, Date, User) VALUES (?, ?, ?, ?, ?)";
                PreparedStatement pst = con.prepareStatement(sql);
255             pst.setString(1, task);
256             pst.setString(2, dateStr);
257             pst.setDouble(3, Double.parseDouble(dateTime.toLocalTime().format(DateTimeFormatter.ofPattern("HH.mm"))));
258             String formattedDate = dateTime.toLocalDate().format(DateTimeFormatter.ofPattern("dd-MM-yyyy"));
259             pst.setString(4, formattedDate);
260             pst.setString(5, LoggedInUser);
261             pst.executeUpdate();
262             JOptionPane.showMessageDialog(null, "Reminder saved to database.");
263             pst.close();
264         } catch (SQLException ex) {
265             JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
266         }
267     }
```

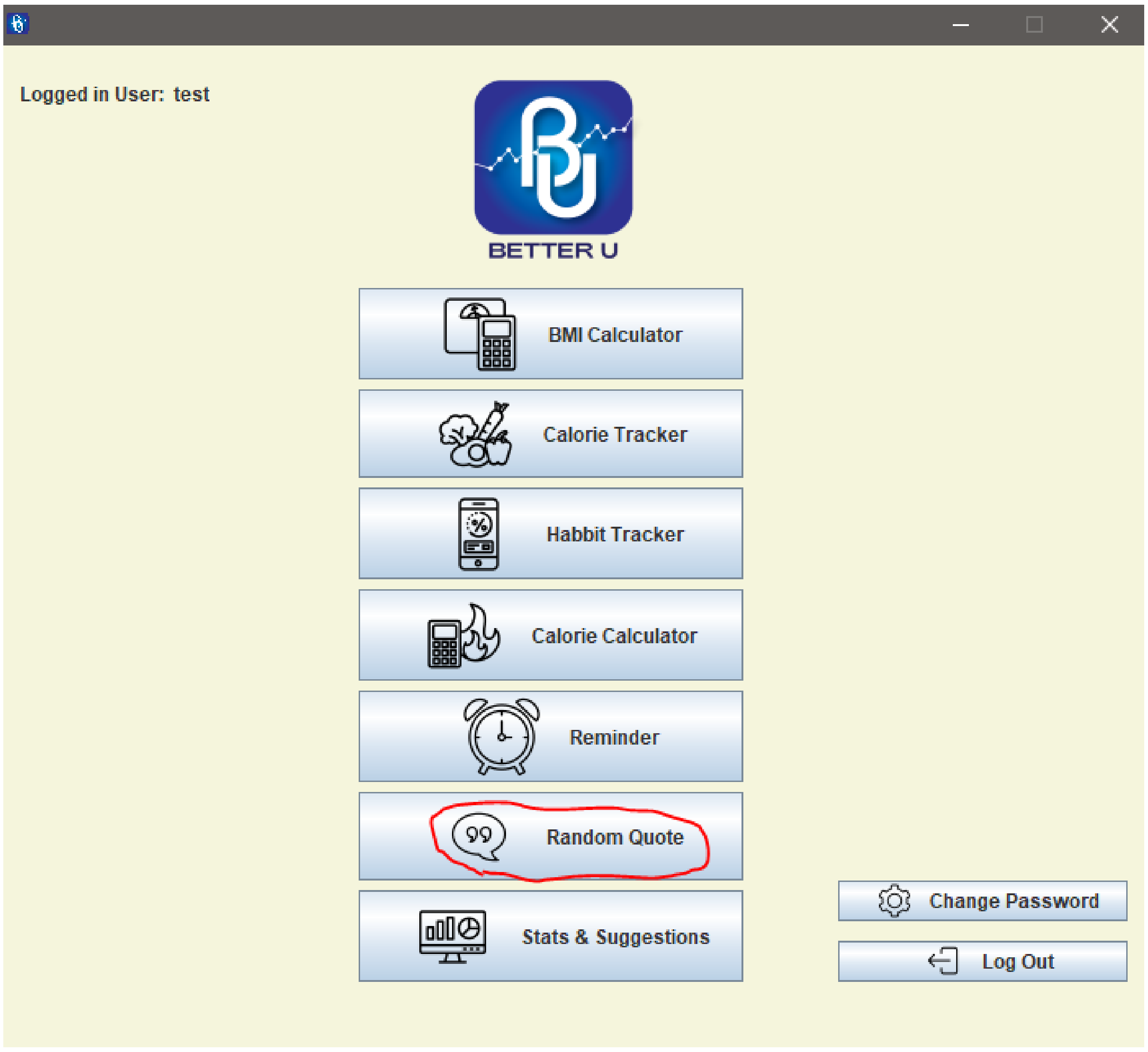
From line 222-227 the feature is checking to make sure there is a message for the reminder. Line 229-231 is getting the time and date for the reminder from the calendar and the two spinners. After this on line 233 a Local date time object is created to represent the time and date of the reminder. Lines 235 -243 is an object that is used for when the reminder goes off. Here it shows a dialogue for the user to see the reminder text and plays a beep noise.

Lines 245-246 uses the timer class to be able to get the reminderTask to be set off at a scheduled time.

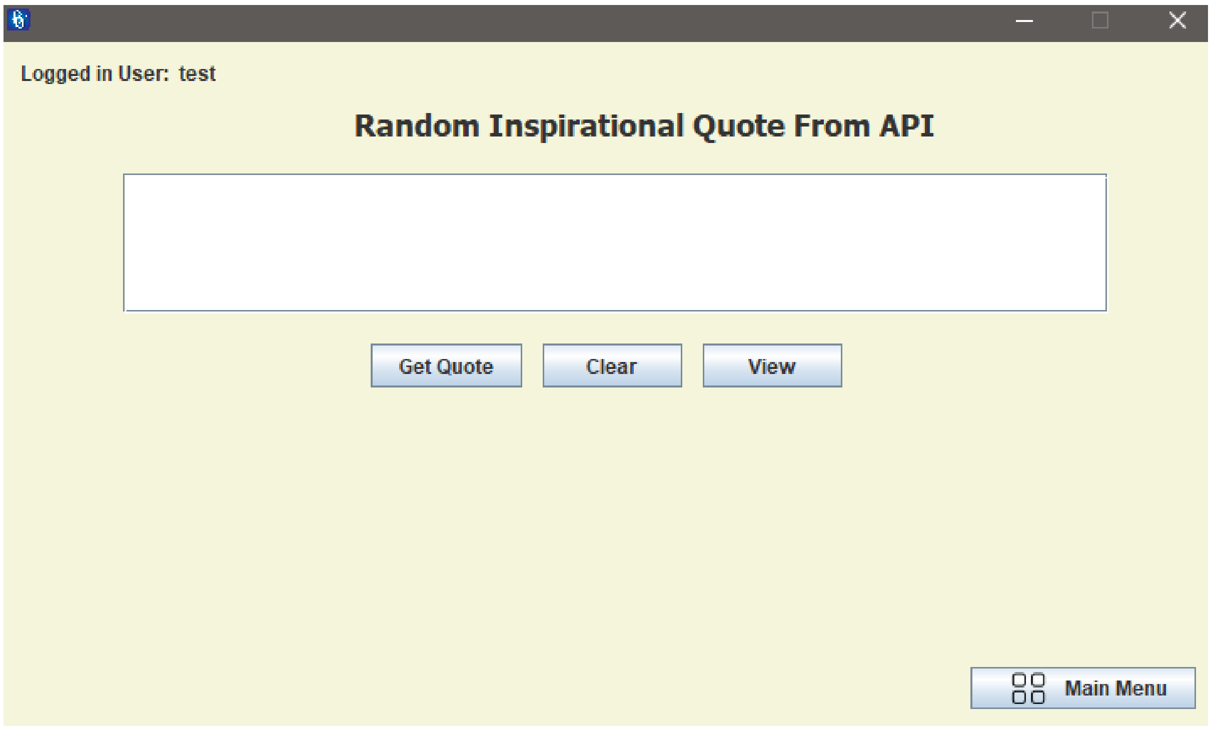
Lines 249-251 formats the time and date of the reminder to strings that represent the time and date of the reminder and then shows a dialogue telling the user that the reminder has been created. After this the information for the timer is stored to the database. I won't go over this again as I have gone over this before and the code is nearly the same. It follows the same procedure.

The user can use the view button to show previous reminder or reminders set for the future. I won't go over this feature as it is the same as in the BMI calculator just for the reminders.

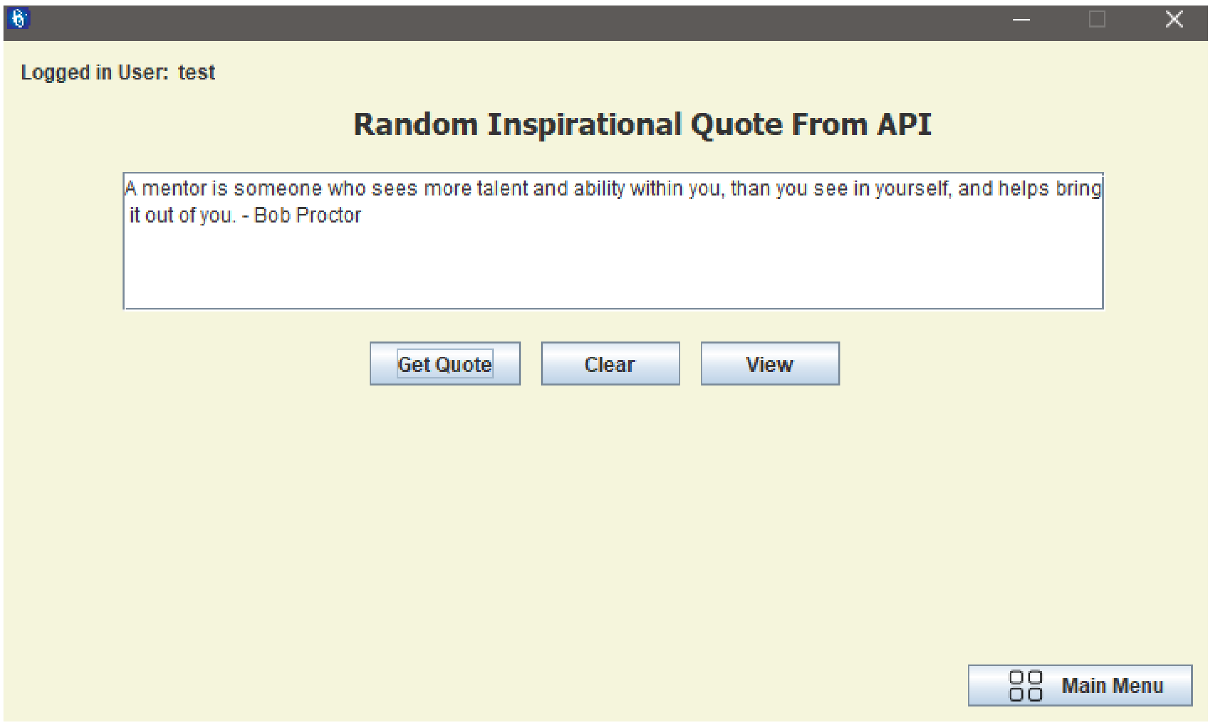
I will now move onto the next feature.



The next feature is the Random Quote feature. A user can open this upon clicking the random quote button.

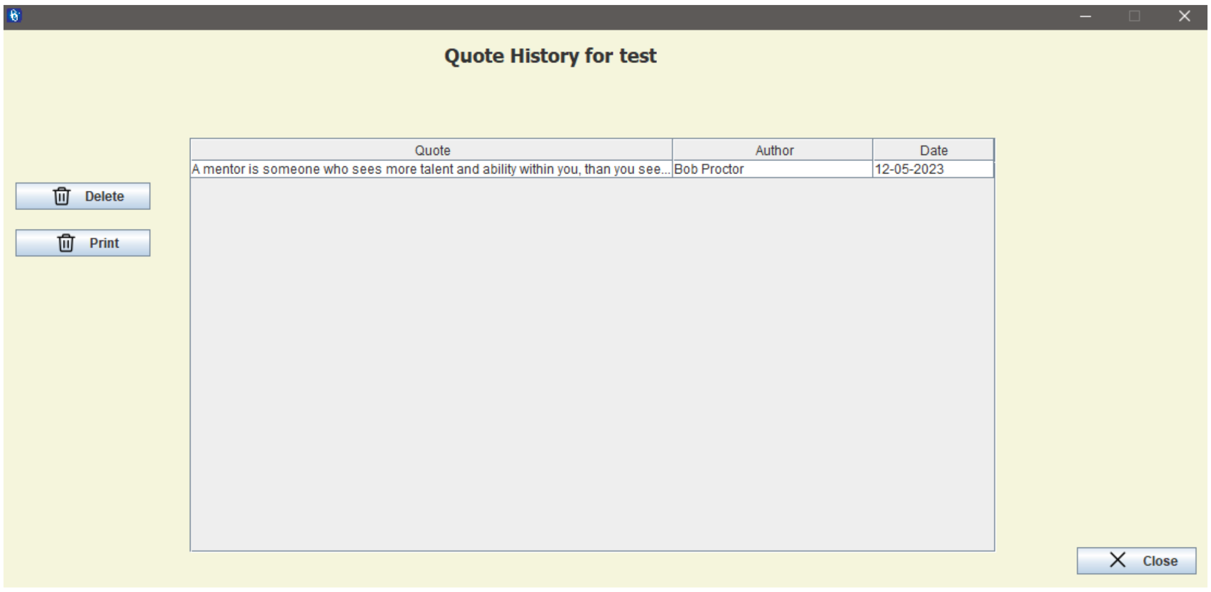


This feature generates a random quote for the user from an API it receives the quote in JSON format and parses it into a more readable format. To use this the user just clicks get quote. Below is what happens after user presses get quote a user can then clear the quote if they wish or view the quotes they have generated



A mentor is someone who sees more talent and ability within you, than you see in yourself, and helps bring it out of you. - Bob Proctor

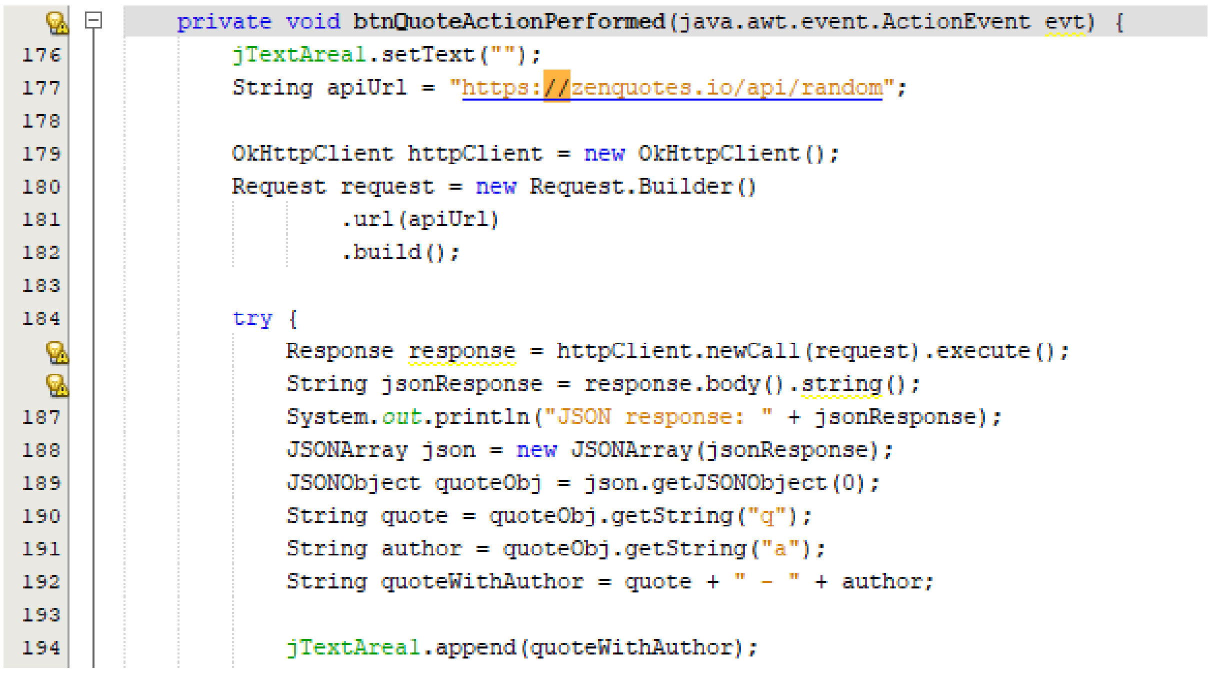
Below is a screenshot of the view window.



Quote History for test

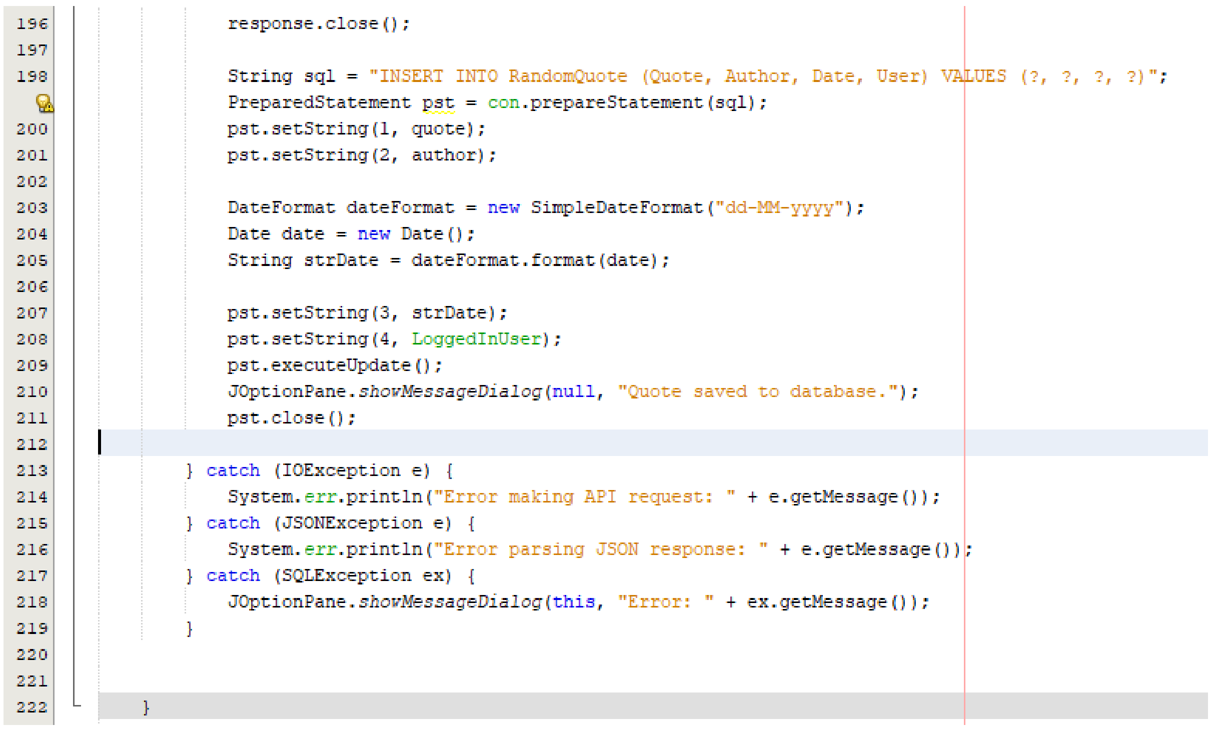
Quote	Author	Date
A mentor is someone who sees more talent and ability within you, than you see...	Bob Proctor	12-05-2023

As seen above the quote is saved to the database and displayed on the table. As I have already explained the view and the clear in the previous features I will just explain the code behind the get quote button.



```
        private void btnQuoteActionPerformed(java.awt.event.ActionEvent evt) {
176         jTextArea1.setText("");
177         String apiUrl = "https://zenquotes.io/api/random";
178
179         OkHttpClient httpClient = new OkHttpClient();
180         Request request = new Request.Builder()
181                 .url(apiUrl)
182                 .build();
183
184         try {
                Response response = httpClient.newCall(request).execute();
                String jsonResponse = response.body().string();
187             System.out.println("JSON response: " + jsonResponse);
188             JSONArray json = new JSONArray(jsonResponse);
189             JSONObject quoteObj = json.getJSONObject(0);
190             String quote = quoteObj.getString("q");
191             String author = quoteObj.getString("a");
192             String quoteWithAuthor = quote + " - " + author;
193
194             jTextArea1.append(quoteWithAuthor);
```

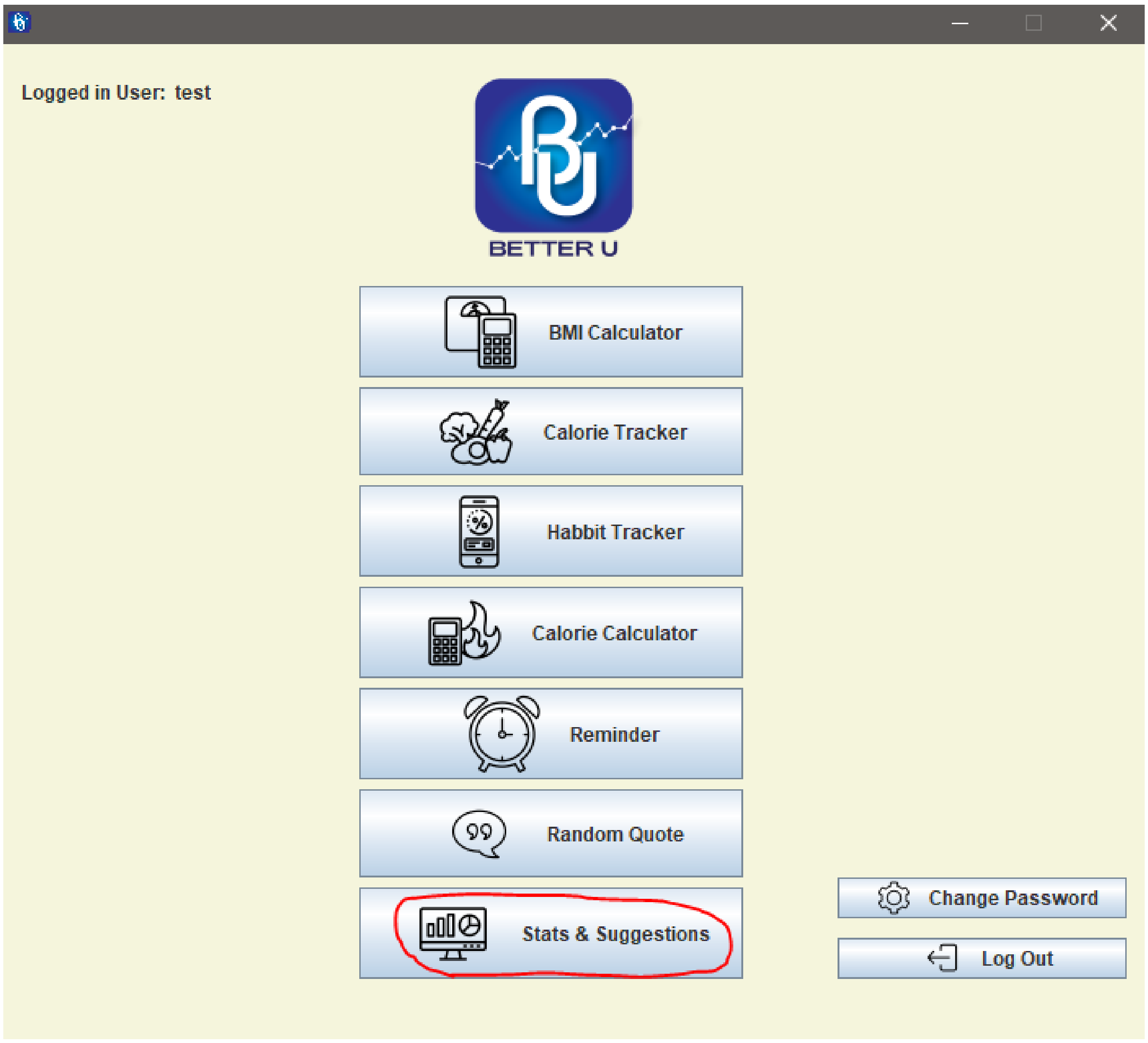
The line 177 is a variable that stores the URL from which the quote is got from. Line 179-182 creates a OkHttpClient instance and then builds a request with the url. In the try block the request is executed on 185 and the response is obtained. Line 186 gets the quote in the form of JSON. The JSON response is converted to a JSONArray this allows for accessing the individual parts of the response. After this the first quote object in the array is got in line 189. After this the quote is broken into the quote and the author. For the reader in the text box on the quote window the quote and author are stuck together for reading purposes.



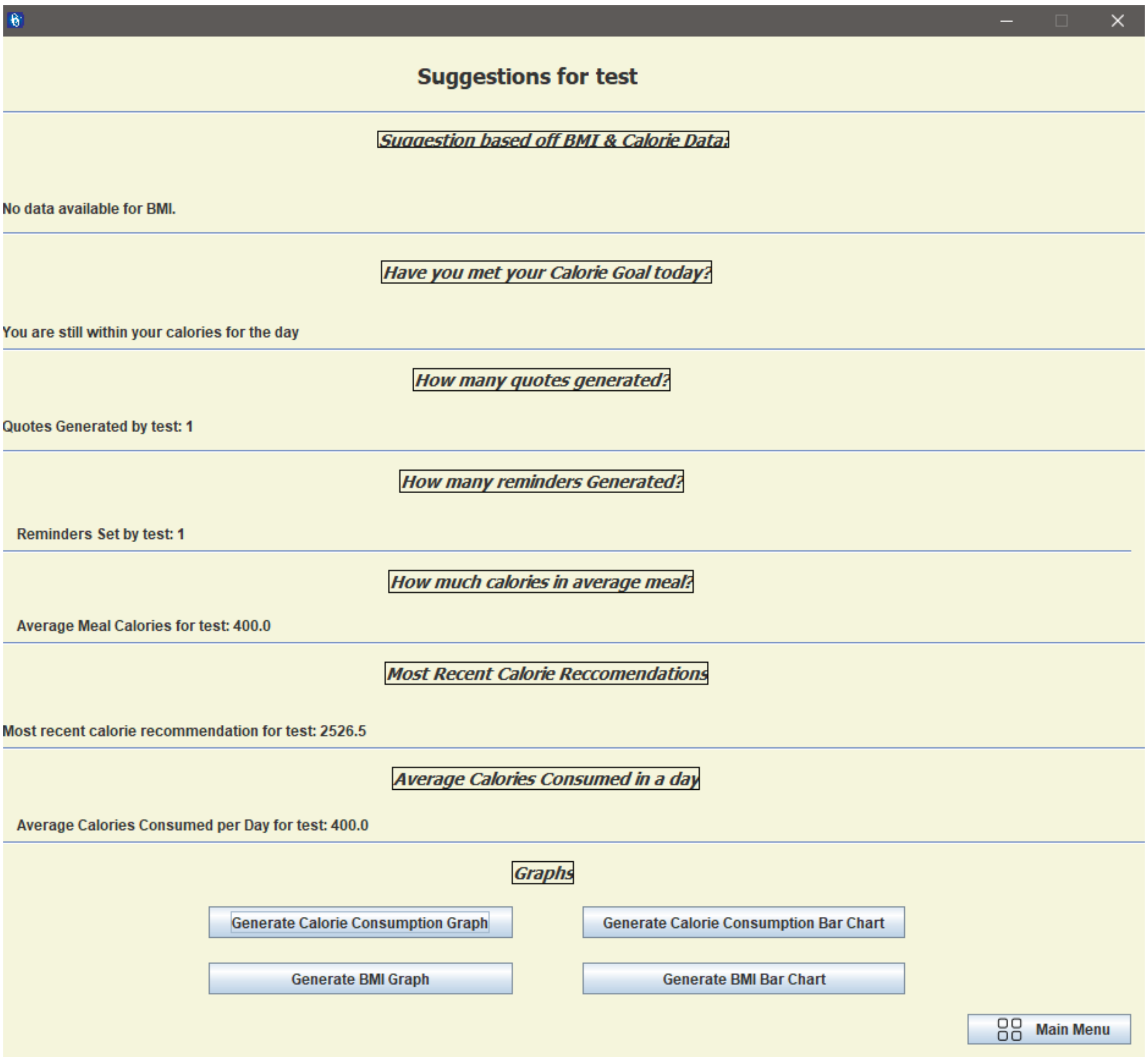
```
196             response.close();
197
198             String sql = "INSERT INTO RandomQuote (Quote, Author, Date, User) VALUES (?, ?, ?, ?)";
                PreparedStatement pst = con.prepareStatement(sql);
200             pst.setString(1, quote);
201             pst.setString(2, author);
202
203             DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
204             Date date = new Date();
205             String strDate = dateFormat.format(date);
206
207             pst.setString(3, strDate);
208             pst.setString(4, LoggedInUser);
209             pst.executeUpdate();
210             JOptionPane.showMessageDialog(null, "Quote saved to database.");
211             pst.close();
212
213         } catch (IOException e) {
214             System.err.println("Error making API request: " + e.getMessage());
215         } catch (JSONException e) {
216             System.err.println("Error parsing JSON response: " + e.getMessage());
217         } catch (SQLException ex) {
218             JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
219         }
220
221
222     }
```

On line 196 the response is closed for proper resource management. The rest of the code inserts the quote to the database. As I have already explained this process in previous features, I will not be explaining it again. On lines 213-219 the catch for the try blocks is here. The first catch catches the IO exception if there was an error making the request to API. The second is for if there was an error parsing the JSON response and the last is for if there was an error adding the quote to the database.

I will now move onto the last feature which is the stats and suggestions.


Logged in User: test
BETTER U
BMI Calculator
Calorie Tracker
Habbit Tracker
Calorie Calculator
Reminder
Random Quote
Stats & Suggestions
Change Password
Log Out

To open this feature a user presses the button for stats and suggestions. Upon opening this is what a user sees below.


Suggestions for test

Suggestion based off BMI & Calorie Data:

No data available for BMI.

Have you met your Calorie Goal today?

You are still within your calories for the day

How many quotes generated?

Quotes Generated by test: 1

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 400.0

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 2526.5

Average Calories Consumed in a day

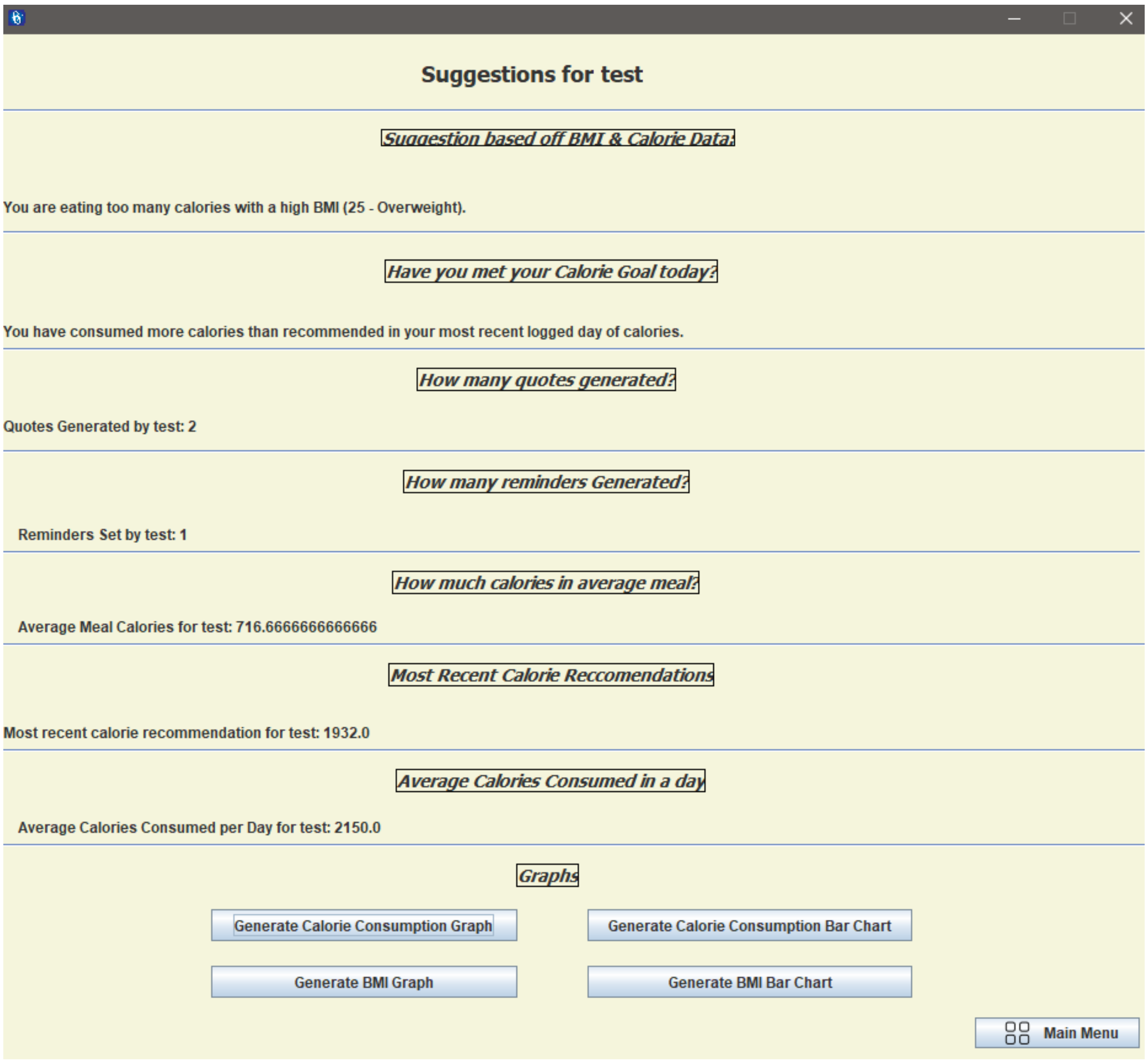
Average Calories Consumed per Day for test: 400.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

The user can see suggestions based off BMI and calorie data. They can see if they have met their calorie goal, they can see how many quotes have been generated and they can see how many reminders have been generated. They can see how much calories in an average meal and they can see the most recent calorie reccomendation the user can also see the average calories consumed in a day. At the end the user then has the option to be able to generate a graph or bar chart on the calore consumption of the user or else a graph on the BMI data and bar chart.

I will populate the system with more entries to show all these stats and suggestions better.


Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 716.6666666666666

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

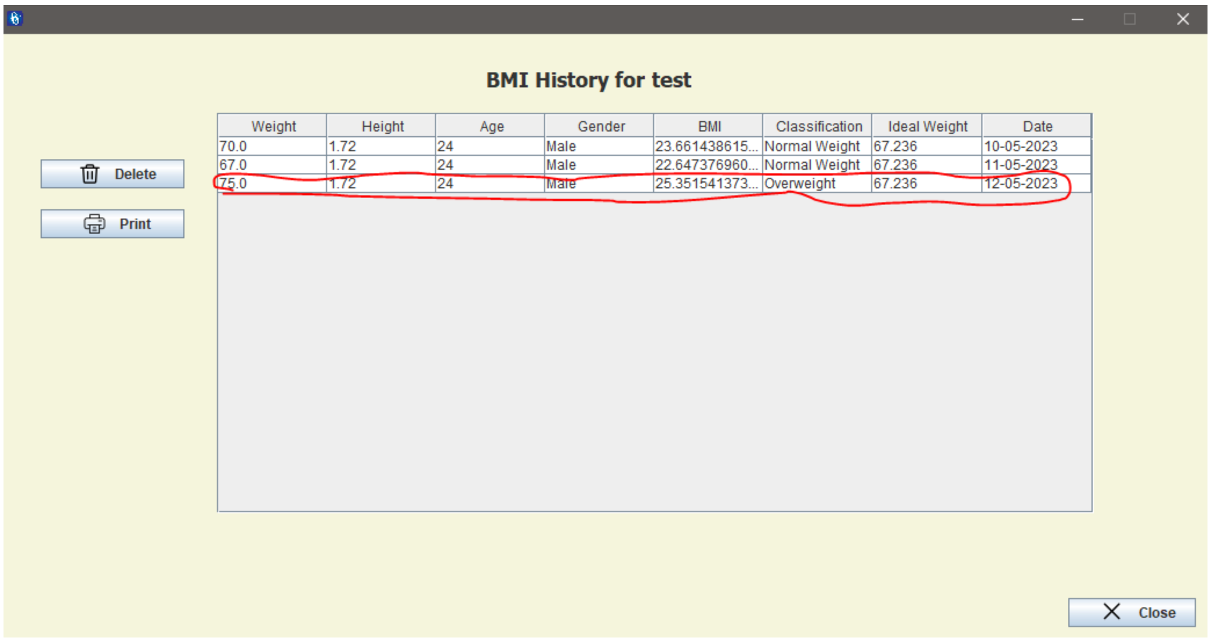
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2150.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

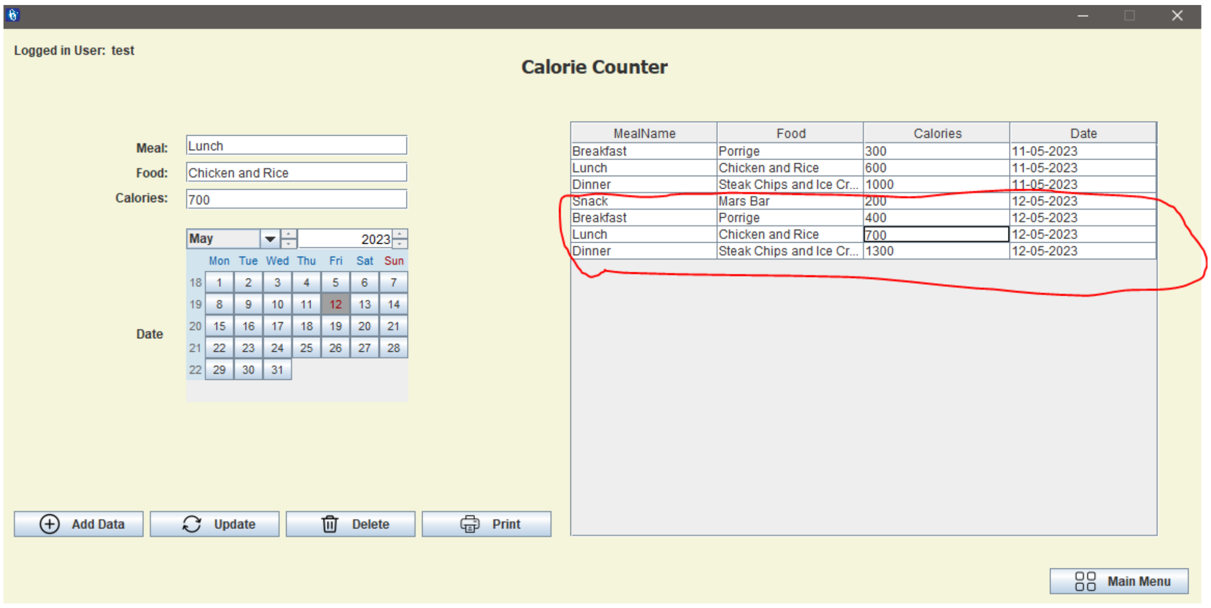
I have populated the system with more data to show the feature better. The first part the suggestion based off the BMI and calorie data it is telling the user that they are eating too many calories with a high BMI. This suggestion uses the most recent BMI recommendation and the most recent day of calories. If they meet a certain criteria this message is triggered.



BMI History for test

Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date
70.0	1.72	24	Male	23.661438615...	Normal Weight	67.236	10-05-2023
67.0	1.72	24	Male	22.647376960...	Normal Weight	67.236	11-05-2023
75.0	1.72	24	Male	25.351541373...	Overweight	67.236	12-05-2023

Above is most recent BMI. Below is the most recent calories for the day.



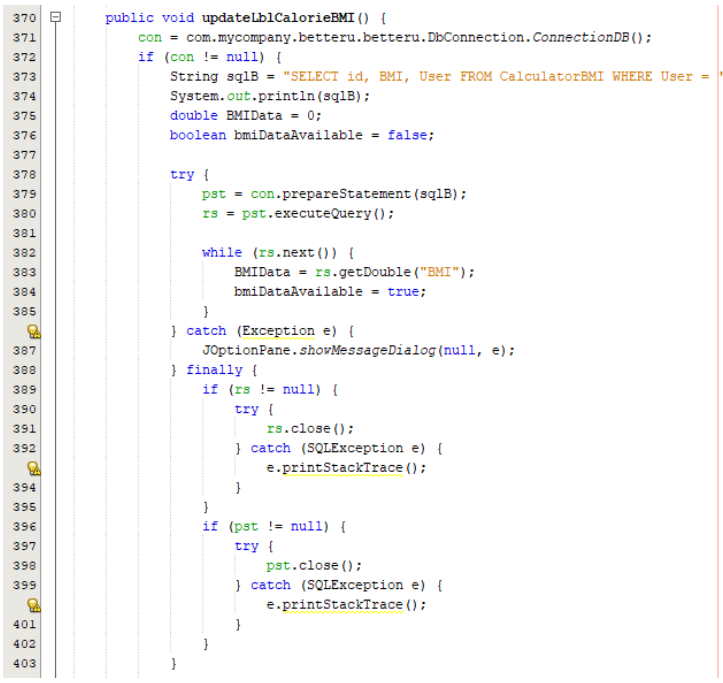
Logged in User: test

Calorie Counter

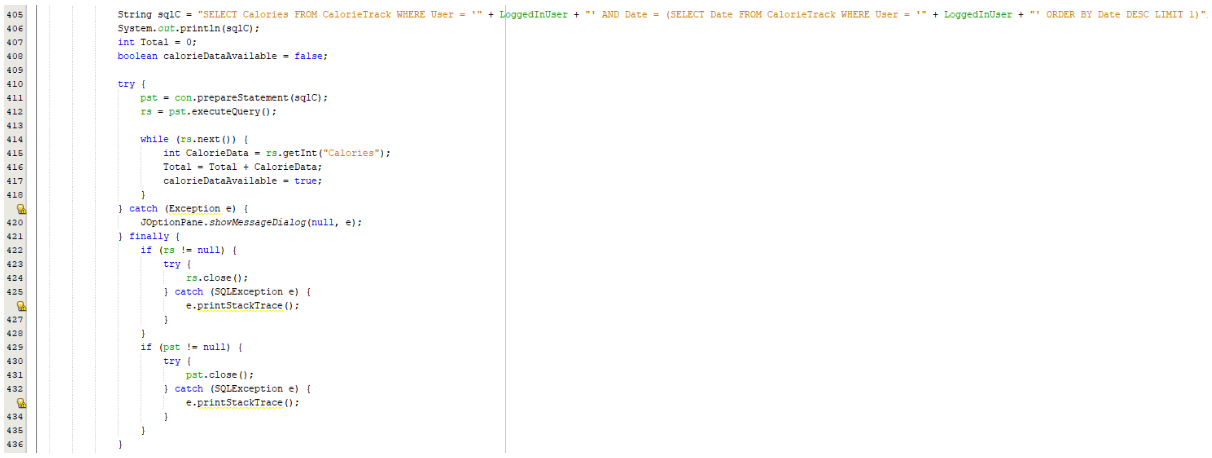
Meal:	Lunch
Food:	Chicken and Rice
Calories:	700
Date	May 2023

MealName	Food	Calories	Date
Breakfast	Porrige	300	11-05-2023
Lunch	Chicken and Rice	600	11-05-2023
Dinner	Steak Chips and Ice Cr...	1000	11-05-2023
Snack	Mars Bar	200	12-05-2023
Breakfast	Porrige	400	12-05-2023
Lunch	Chicken and Rice	700	12-05-2023
Dinner	Steak Chips and Ice Cr...	1300	12-05-2023

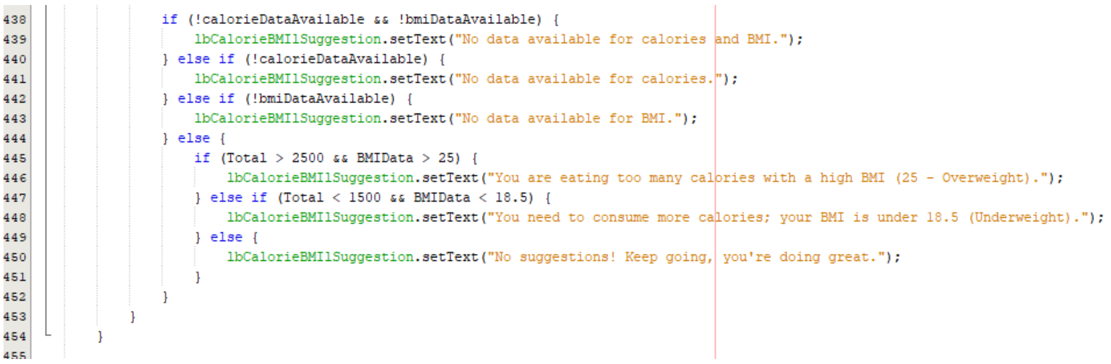
Ass seen from the above screenshots the most recent BMI was overweight and the most recent calories is 2600. If we look at how this suggestion is created we will see why this message was made.


```
370      public void updateLblCalorieBMI() {
371          con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
372          if (con != null) {
373              String sqlB = "SELECT id, BMI, User FROM CalculatorBMI WHERE User = '" + LoggedInUser + "' ORDER BY id DESC LIMIT 1";
374              System.out.println(sqlB);
375              double BMIData = 0;
376              boolean bmiDataAvailable = false;
377
378              try {
379                  pst = con.prepareStatement(sqlB);
380                  rs = pst.executeQuery();
381
382                  while (rs.next()) {
383                      BMIData = rs.getDouble("BMI");
384                      bmiDataAvailable = true;
385                  }
                 } catch (Exception e) {
387                  JOptionPane.showMessageDialog(null, e);
388              } finally {
389                  if (rs != null) {
390                      try {
391                          rs.close();
392                      } catch (SQLException e) {
                             e.printStackTrace();
394                      }
395                  }
396                  if (pst != null) {
397                      try {
398                          pst.close();
399                      } catch (SQLException e) {
                             e.printStackTrace();
401                      }
402                  }
403              }
```

This first screenshot gets the most recent BMI reading from the user. It does this by running the query on line 373. It gets the most recent by ordering by id and using a descending limit of 1.

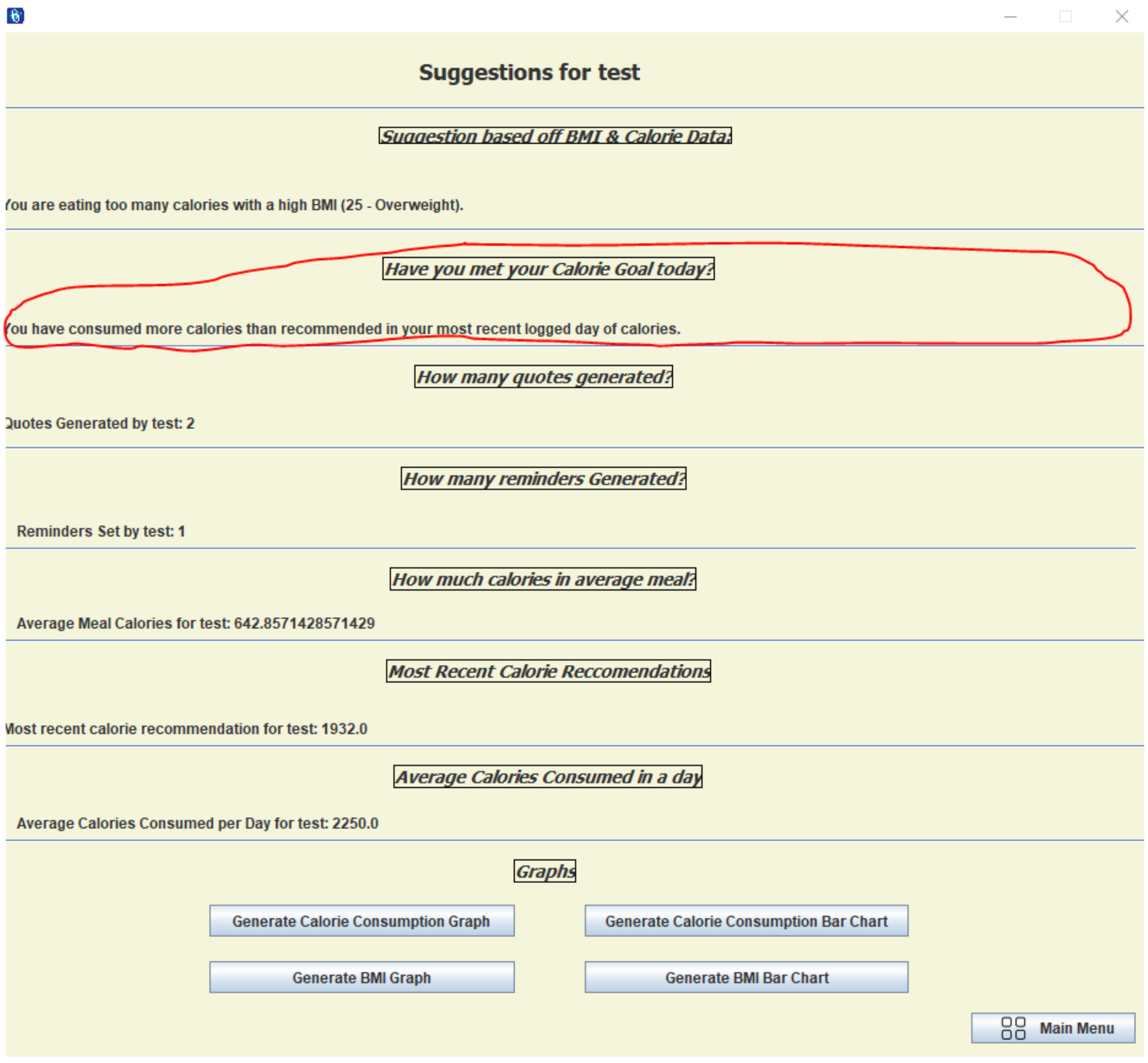

```
405              String sqlC = "SELECT Calories FROM CalorieTrack WHERE User = '" + LoggedInUser + "' AND Date = (SELECT Date FROM CalorieTrack WHERE User = '" + LoggedInUser + "' ORDER BY Date DESC LIMIT 1)".
406              System.out.println(sqlC);
407              int Total = 0;
408              boolean calorieDataAvailable = false;
409
410              try {
411                  pst = con.prepareStatement(sqlC);
412                  rs = pst.executeQuery();
413
414                  while (rs.next()) {
415                      int CalorieData = rs.getInt("Calories");
416                      Total = Total + CalorieData;
417                      calorieDataAvailable = true;
418                  }
                 } catch (Exception e) {
420                  JOptionPane.showMessageDialog(null, e);
421              } finally {
422                  if (rs != null) {
423                      try {
424                          rs.close();
425                      } catch (SQLException e) {
                             e.printStackTrace();
427                      }
428                  }
429                  if (pst != null) {
430                      try {
431                          pst.close();
432                      } catch (SQLException e) {
                             e.printStackTrace();
434                      }
435                  }
436              }
```

This uses a nested SQL query to grab the most recent calorie data from a user. It gets the most recent date of calories entered by the user and groups them together with the calorie amounts and adds them up. It does this on line 405. What's in the brackets in the SQL statement is essentially like a variable for the date that will grab the most recent dates calories.


```
438              if (!calorieDataAvailable && !bmiDataAvailable) {
439                  lbCalorieBMI1Suggestion.setText("No data available for calories and BMI.");
440              } else if (!calorieDataAvailable) {
441                  lbCalorieBMI1Suggestion.setText("No data available for calories.");
442              } else if (!bmiDataAvailable) {
443                  lbCalorieBMI1Suggestion.setText("No data available for BMI.");
444              } else {
445                  if (Total > 2500 && BMIData > 25) {
446                      lbCalorieBMI1Suggestion.setText("You are eating too many calories with a high BMI (25 - Overweight).");
447                  } else if (Total < 1500 && BMIData < 18.5) {
448                      lbCalorieBMI1Suggestion.setText("You need to consume more calories; your BMI is under 18.5 (Underweight).");
449                  } else {
450                      lbCalorieBMI1Suggestion.setText("No suggestions! Keep going, you're doing great.");
451                  }
452              }
453          }
454      }
455
```

At the end is a bunch of if statements to give a label to the two results for the user to read. For example for this demonstration if statement on line 445 was triggered due to the total calories being 2600 which is greater then 2500 and the BMI is over 25. Depending on the results one of the if statements will be triggered. If there isn't enough data for either the BMI or the calories the lines 438-444 will be triggered depending on the reason.

I will now move onto the second suggestion. This is the have you met your calorie goal today. Screenshot below.



Suggestions for test

Suggestion based off BMI & Calorie Data:

ou are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

ou have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

uotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

ost recent calorie recommendation for test: 1932.0

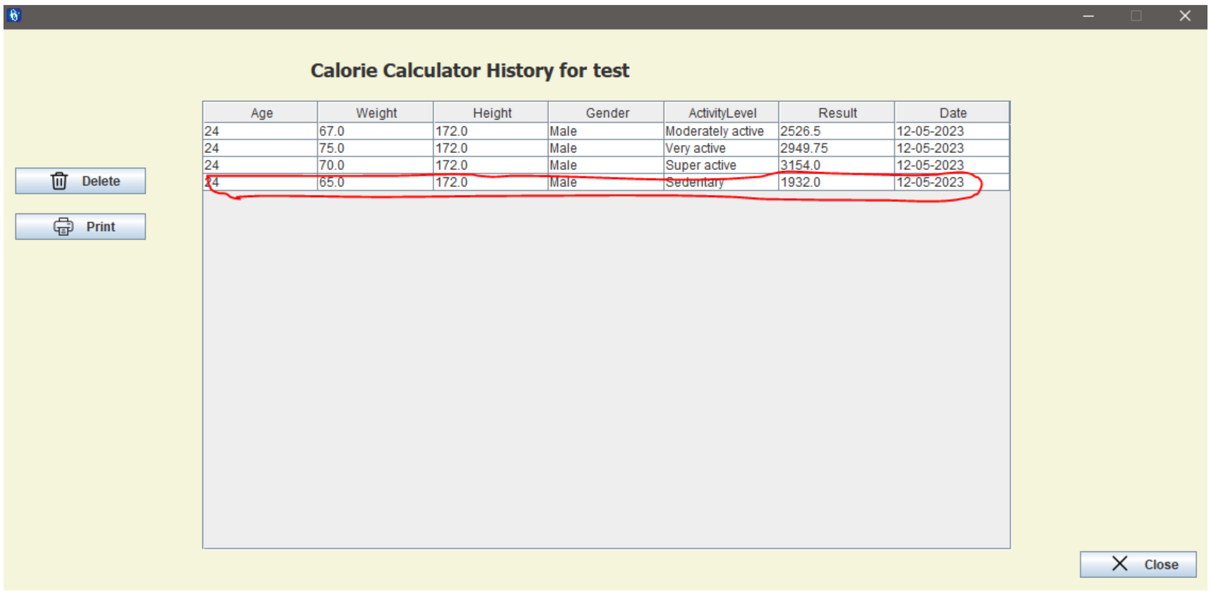
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

Graphs
Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

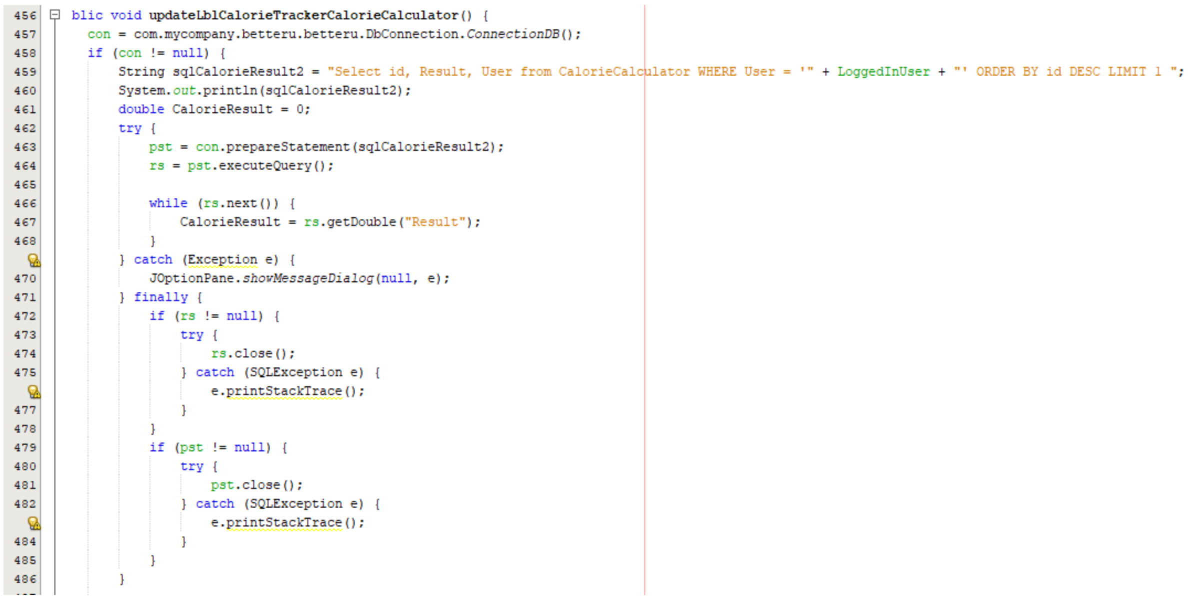
This suggestion checks if you have consumed more calories today than was reccomened for you by the calorie calcultor. In this case we have.

The most recent calorie recommendation is below.



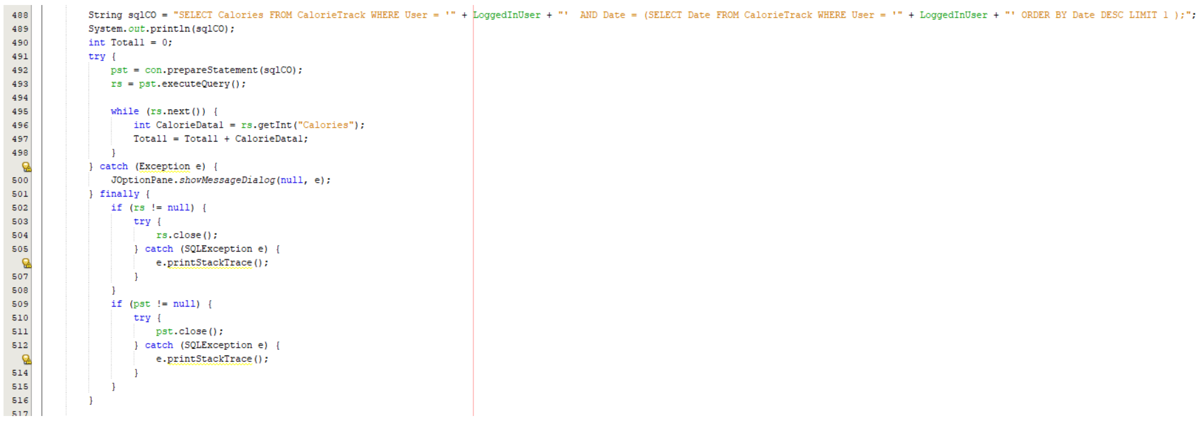
Age	Weight	Height	Gender	ActivityLevel	Result	Date
24	67.0	172.0	Male	Moderately active	2526.5	12-05-2023
24	75.0	172.0	Male	Very active	2949.75	12-05-2023
24	70.0	172.0	Male	Super active	3154.0	12-05-2023
24	65.0	172.0	Male	Sedentary	1932.0	12-05-2023

As seen in the screenshot the most recent calorie recommendation is is 1932 calories. Today we have exceeded this due to consuming 2600 calories. I will show the code for this suggestion.



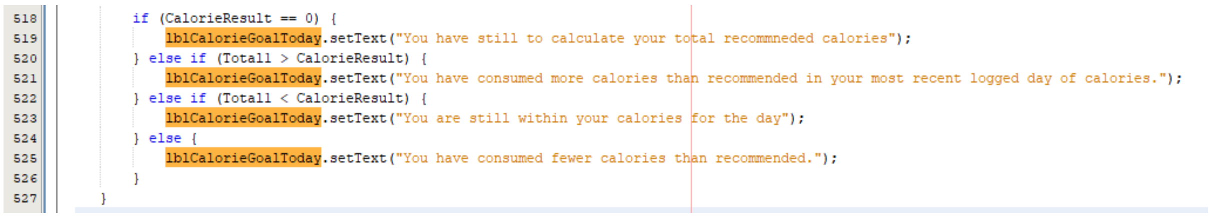
```
456      blic void updateLblCalorieTrackerCalorieCalculator() {
457          con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
458          if (con != null) {
459              String sqlCalorieResult2 = "Select id, Result, User from CalorieCalculator WHERE User = '" + LoggedInUser + "' ORDER BY id DESC LIMIT 1 ";
460              System.out.println(sqlCalorieResult2);
461              double CalorieResult = 0;
462              try {
463                  pst = con.prepareStatement(sqlCalorieResult2);
464                  rs = pst.executeQuery();
465  
466                  while (rs.next()) {
467                      CalorieResult = rs.getDouble("Result");
468                  }
                 } catch (Exception e) {
470                  JOptionPane.showMessageDialog(null, e);
471              } finally {
472                  if (rs != null) {
473                      try {
474                          rs.close();
475                      } catch (SQLException e) {
                             e.printStackTrace();
477                      }
478                  }
479                  if (pst != null) {
480                      try {
481                          pst.close();
482                      } catch (SQLException e) {
                             e.printStackTrace();
484                      }
485                  }
486              }
```

The first SQL statement on line 459 gets the most recent result from the calorie calcuator for the user.



```
488              String sqlCO = "SELECT Calories FROM CalorieTrack WHERE User = '" + LoggedInUser + "'  AND Date = (SELECT Date FROM CalorieTrack WHERE User = '" + LoggedInUser + "' ORDER BY Date DESC LIMIT 1 );";
489              System.out.println(sqlCO);
490              int Totall = 0;
491              try {
492                  pst = con.prepareStatement(sqlCO);
493                  rs = pst.executeQuery();
494  
495                  while (rs.next()) {
496                      int CalorieDatal = rs.getInt("Calories");
497                      Totall = Totall + CalorieDatal;
498                  }
                 } catch (Exception e) {
500                  JOptionPane.showMessageDialog(null, e);
501              } finally {
502                  if (rs != null) {
503                      try {
504                          rs.close();
505                      } catch (SQLException e) {
                             e.printStackTrace();
507                      }
508                  }
509                  if (pst != null) {
510                      try {
511                          pst.close();
512                      } catch (SQLException e) {
                             e.printStackTrace();
514                      }
515                  }
516              }
```

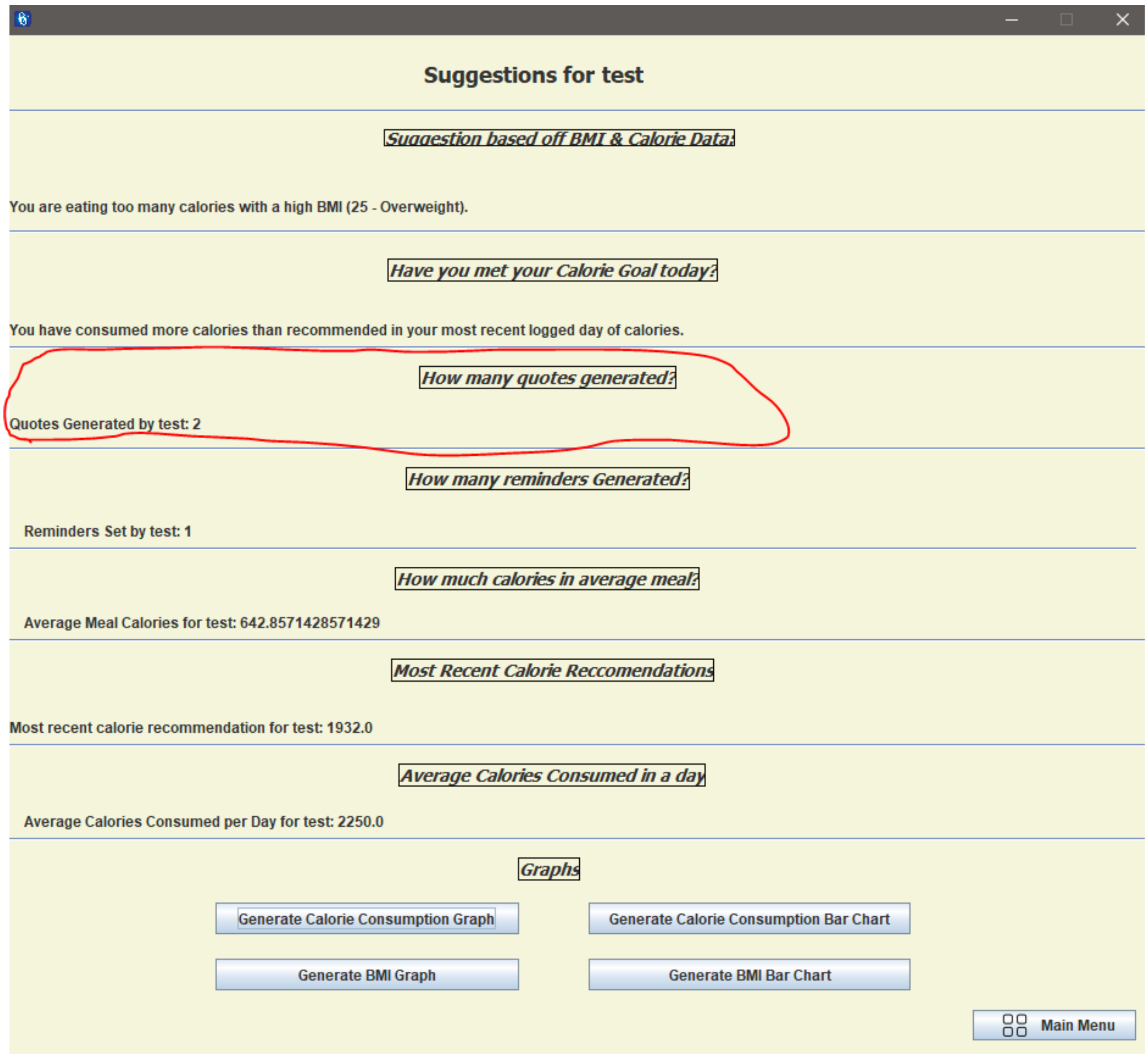
The second SQL query gets the most recent days results from the from the calorie tracker. It gets the results and uses the nested query to be able to get date for the most recent and the outer query gets all the calorie values for that date. This gives the value of all the calories logged eaten in that day.



```
        if (CalorieResult == 0) {
            lblCalorieGoalToday.setText("You have still to calculate your total recommneded calories");
        } else if (Totall > CalorieResult) {
            lblCalorieGoalToday.setText("You have consumed more calories than recommended in your most recent logged day of calories.");
        } else if (Totall < CalorieResult) {
            lblCalorieGoalToday.setText("You are still within your calories for the day");
        } else {
            lblCalorieGoalToday.setText("You have consumed fewer calories than recommended.");
        }
    }
```

There is then some if statements to print a label for the result of the two querys when comapred against each other. The first is making sure the calorie result isnt 0 otherwise it tells the user to calculate their total reccomended calories. If total calories consumed today is greater than the calorie result which in this case it is the user gets the message that they have consumed too many calories today. If it is less then the user gets the message that they have consumed fewer calories than reccomended and if the result of the reccomendation is bigger than the total the user is told they are still within their calories for the day.

It is time to move onto the third suggestion.


Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

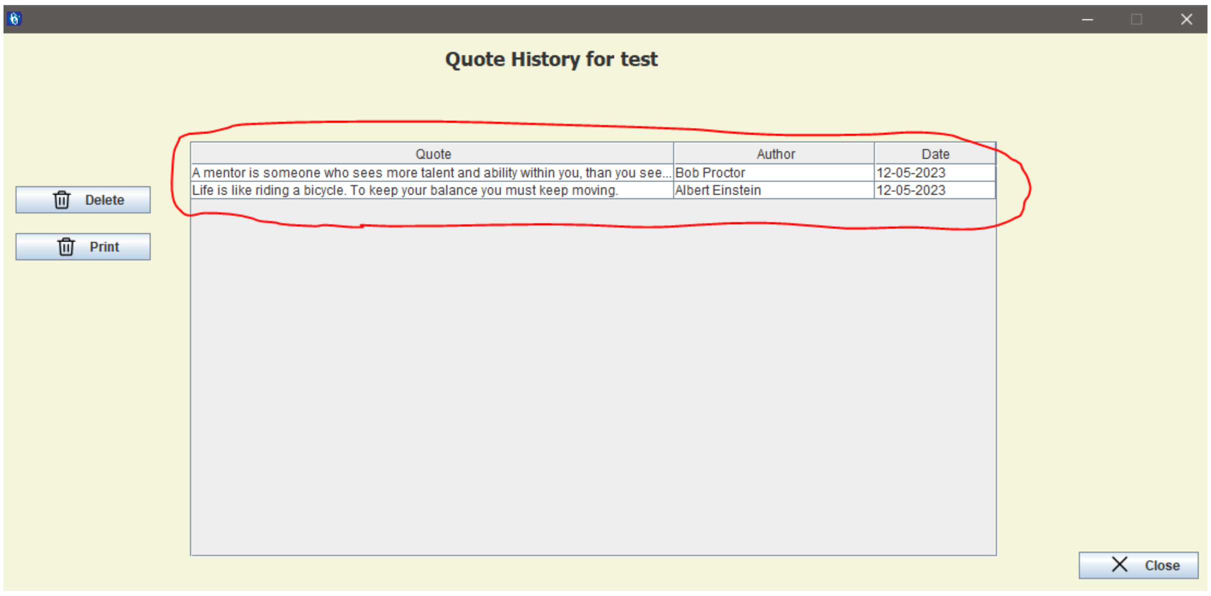
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

Graphs

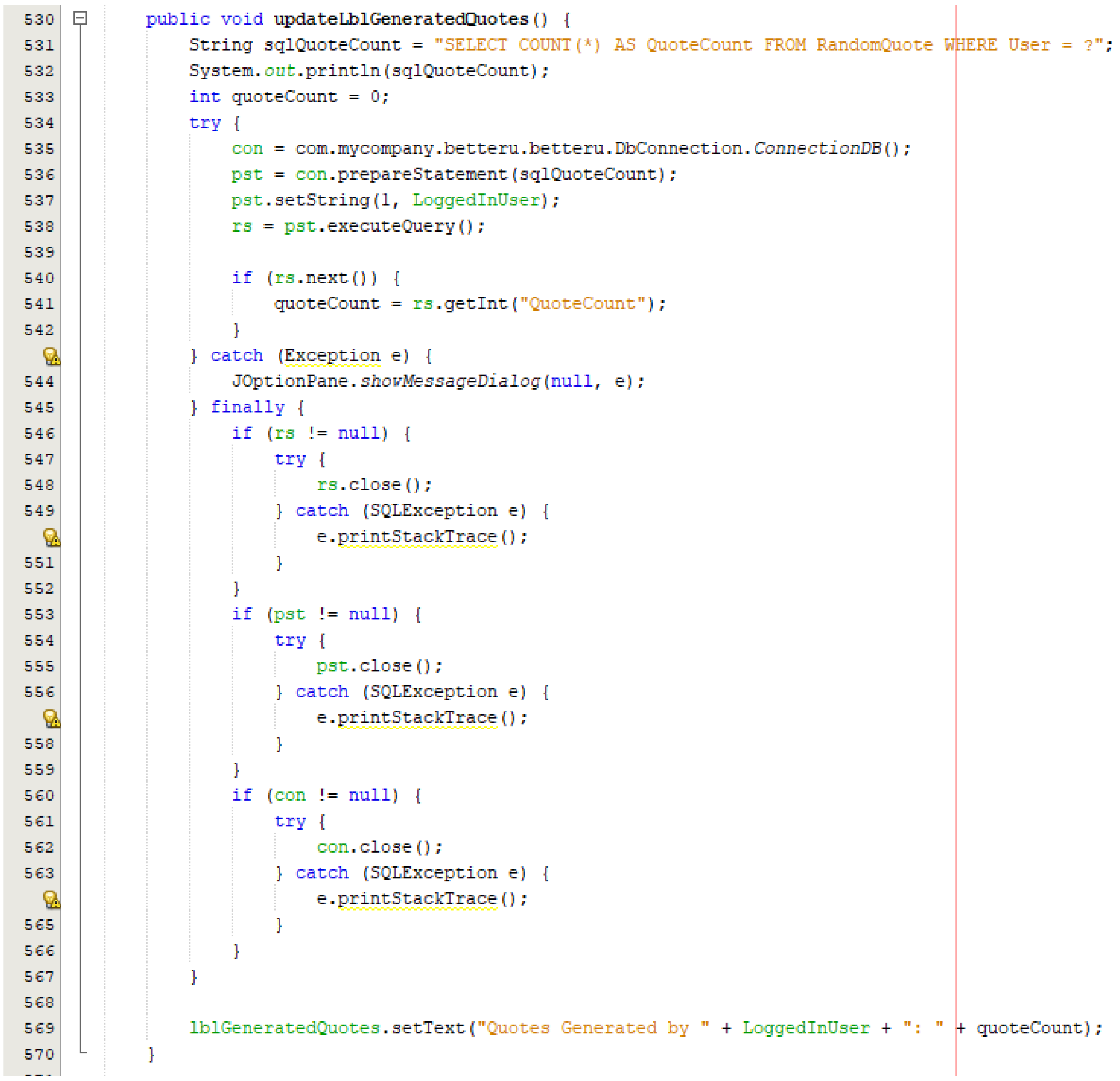
Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

The third suggestion is how many quotes are generated by the user logged in. In this case test is the user logged in.


Quote History for test

Quote	Author	Date
A mentor is someone who sees more talent and ability within you, than you see...	Bob Proctor	12-05-2023
Life is like riding a bicycle. To keep your balance you must keep moving.	Albert Einstein	12-05-2023

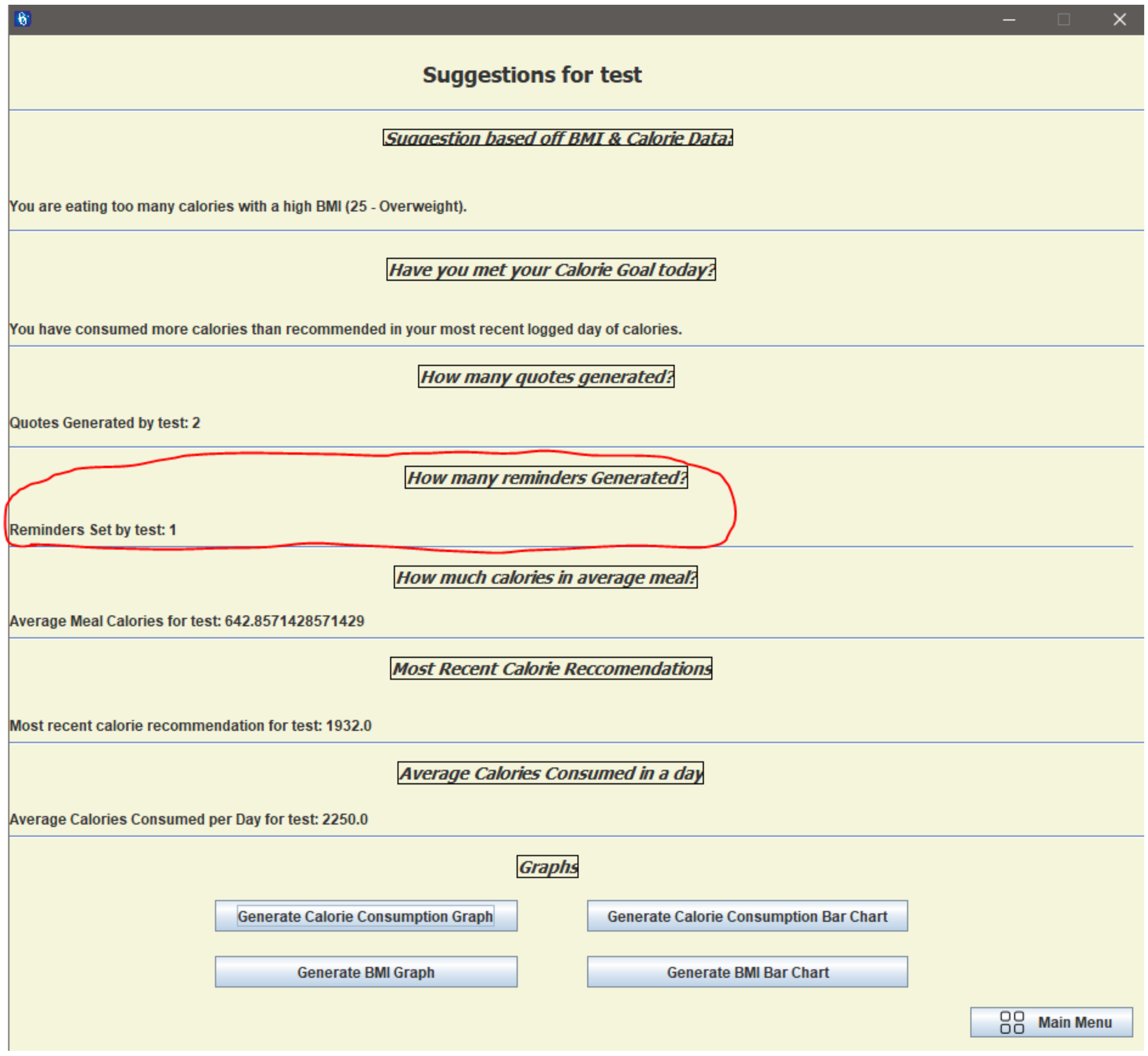
As seen in the above screenshot the user test has generated 2 quotes. Below I will show you thr code behind being able to get the result on the stats and suggestions page.



```
530      public void updateLblGeneratedQuotes() {
531          String sqlQuoteCount = "SELECT COUNT(*) AS QuoteCount FROM RandomQuote WHERE User = ?";
532          System.out.println(sqlQuoteCount);
533          int quoteCount = 0;
534          try {
535              con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
536              pst = con.prepareStatement(sqlQuoteCount);
537              pst.setString(1, LoggedInUser);
538              rs = pst.executeQuery();
539
540              if (rs.next()) {
541                  quoteCount = rs.getInt("QuoteCount");
542              }
             } catch (Exception e) {
544              JOptionPane.showMessageDialog(null, e);
545          } finally {
546              if (rs != null) {
547                  try {
548                      rs.close();
549                  } catch (SQLException e) {
                         e.printStackTrace();
551                  }
552              }
553              if (pst != null) {
554                  try {
555                      pst.close();
556                  } catch (SQLException e) {
                         e.printStackTrace();
558                  }
559              }
560              if (con != null) {
561                  try {
562                      con.close();
563                  } catch (SQLException e) {
                         e.printStackTrace();
565                  }
566              }
567          }
568
569          lblGeneratedQuotes.setText("Quotes Generated by " + LoggedInUser + ": " + quoteCount);
570      }
```

This is quite a simple query. The query on line 531 gets a count of the quotes in the database where the user is equal to LoggedInUser in this case it is test.

I will now move onto the next stat and suggestion. Which is how many reminders generated.



Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

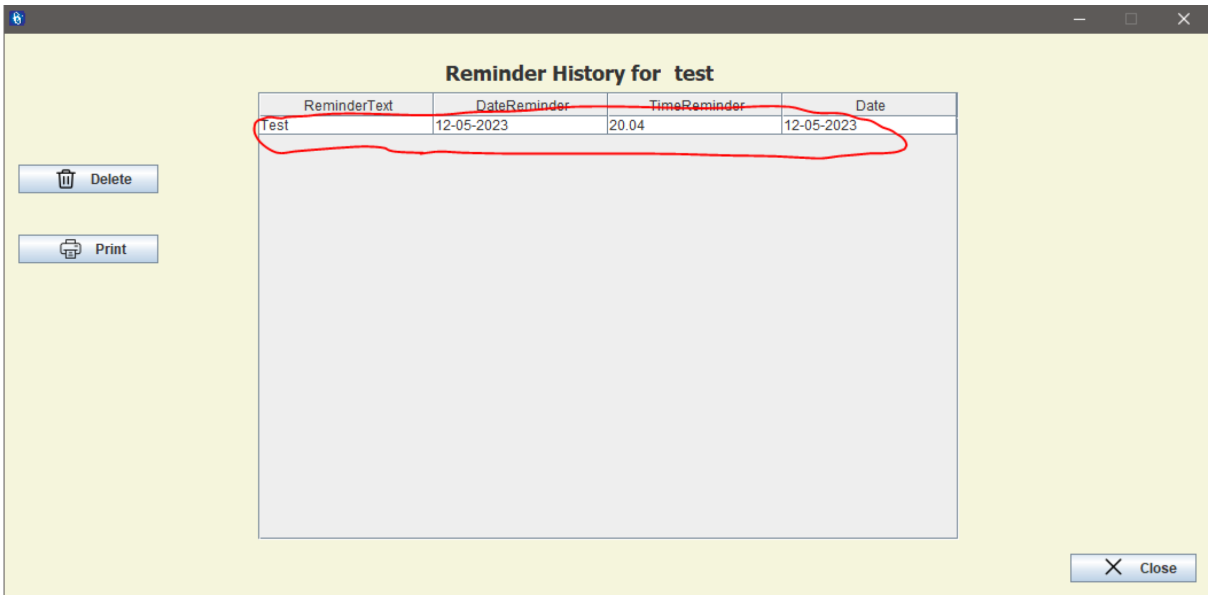
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

I will show the table of reminders for test below.

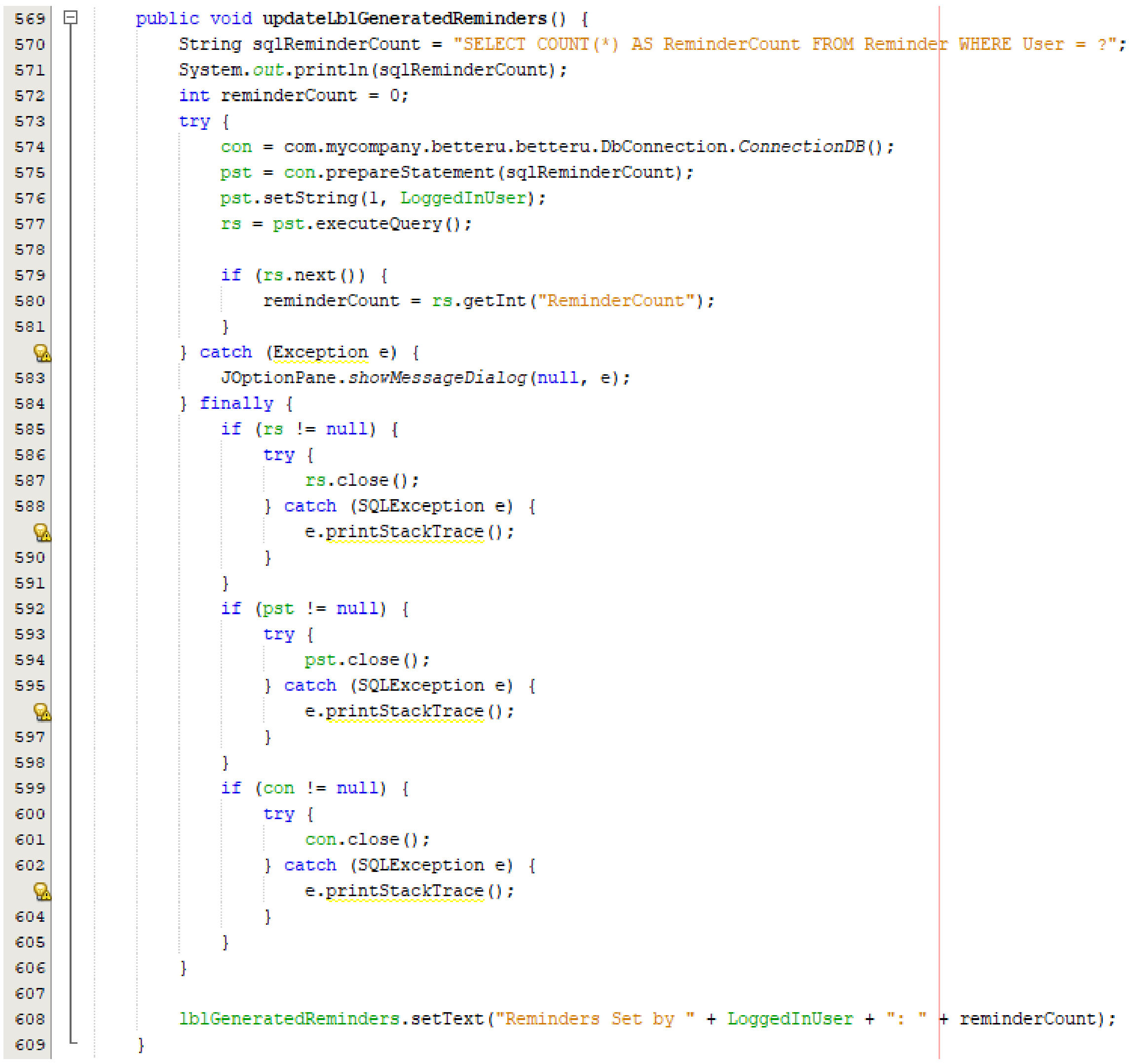


Reminder History for test

ReminderText	DateReminder	TimeReminder	Date
Test	12-05-2023	20.04	12-05-2023

As seen in the table test only has one reminder.

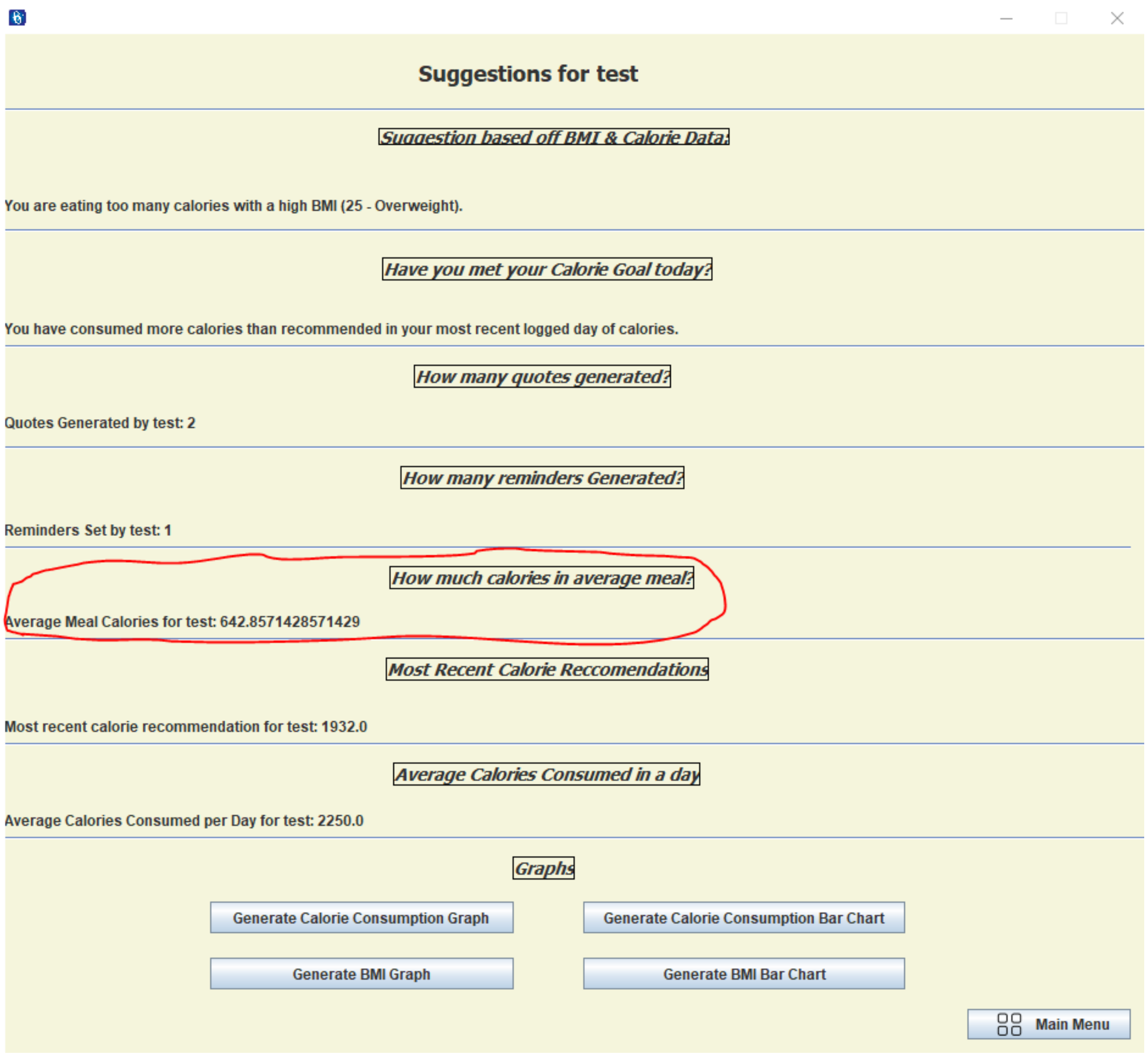
The code for producing the result on the stats and suggestion page is below.



```
569     public void updateLblGeneratedReminders() {
570         String sqlReminderCount = "SELECT COUNT(*) AS ReminderCount FROM Reminder WHERE User = ?";
571         System.out.println(sqlReminderCount);
572         int reminderCount = 0;
573         try {
574             con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
575             pst = con.prepareStatement(sqlReminderCount);
576             pst.setString(1, LoggedInUser);
577             rs = pst.executeQuery();
578 
579             if (rs.next()) {
580                 reminderCount = rs.getInt("ReminderCount");
581             }
            } catch (Exception e) {
583             JOptionPane.showMessageDialog(null, e);
584         } finally {
585             if (rs != null) {
586                 try {
587                     rs.close();
588                 } catch (SQLException e) {
                        e.printStackTrace();
590                 }
591             }
592             if (pst != null) {
593                 try {
594                     pst.close();
595                 } catch (SQLException e) {
                        e.printStackTrace();
597                 }
598             }
599             if (con != null) {
600                 try {
601                     con.close();
602                 } catch (SQLException e) {
                        e.printStackTrace();
604                 }
605             }
606         }
607 
608         lblGeneratedReminders.setText("Reminders Set by " + LoggedInUser + ": " + reminderCount);
609     }
```

This code is very similar to the how many quotes code. It works by using the sql query on line 570. This gets the count of quotes thay have the user as LoggedInUser which in this case is test.

I will now move onto the next suggestion. This is how many calories in a average meal.


Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

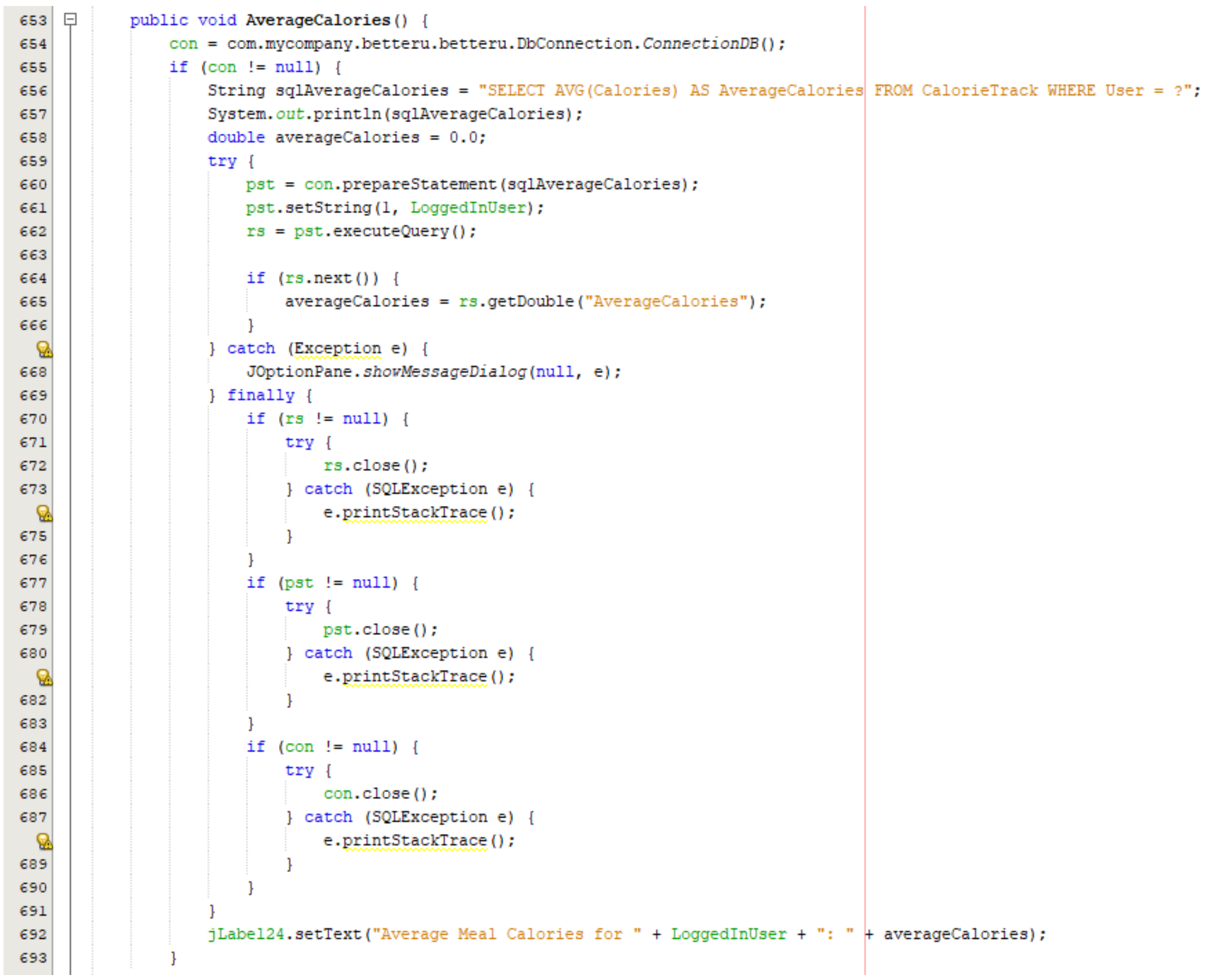
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

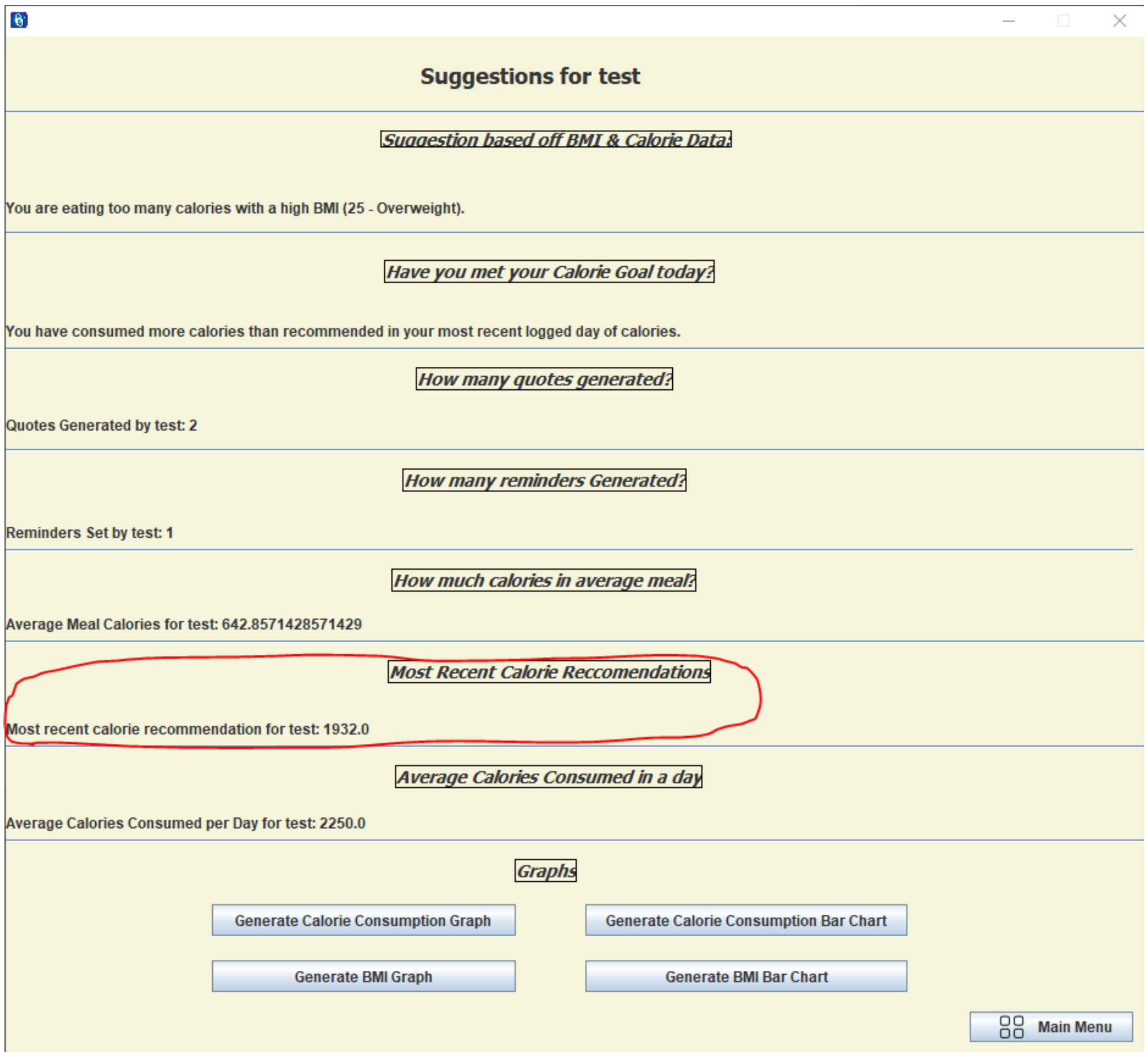
As seen above this calcualtes the average calories that are in each meal. I will show the code for this below.



```
653     public void AverageCalories() {
654         con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
655         if (con != null) {
656             String sqlAverageCalories = "SELECT AVG(Calories) AS AverageCalories FROM CalorieTrack WHERE User = ?";
657             System.out.println(sqlAverageCalories);
658             double averageCalories = 0.0;
659             try {
660                 pst = con.prepareStatement(sqlAverageCalories);
661                 pst.setString(1, LoggedInUser);
662                 rs = pst.executeQuery();
663 
664                 if (rs.next()) {
665                     averageCalories = rs.getDouble("AverageCalories");
666                 }
                } catch (Exception e) {
668                 JOptionPane.showMessageDialog(null, e);
669             } finally {
670                 if (rs != null) {
671                     try {
672                         rs.close();
673                     } catch (SQLException e) {
                            e.printStackTrace();
675                     }
676                 }
677                 if (pst != null) {
678                     try {
679                         pst.close();
680                     } catch (SQLException e) {
                            e.printStackTrace();
682                     }
683                 }
684                 if (con != null) {
685                     try {
686                         con.close();
687                     } catch (SQLException e) {
                            e.printStackTrace();
689                     }
690                 }
691             }
692             jLabel24.setText("Average Meal Calories for " + LoggedInUser + ": " + averageCalories);
693         }
```

This is quite a simple statement. On line 656 the statement is to select the averge calories from the calorie tracker table where the user is LoggedInUser. In this case the LoggedInUser is test.

I will now move onto the next suggestion this is the most recent calorie reccomendation.



Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

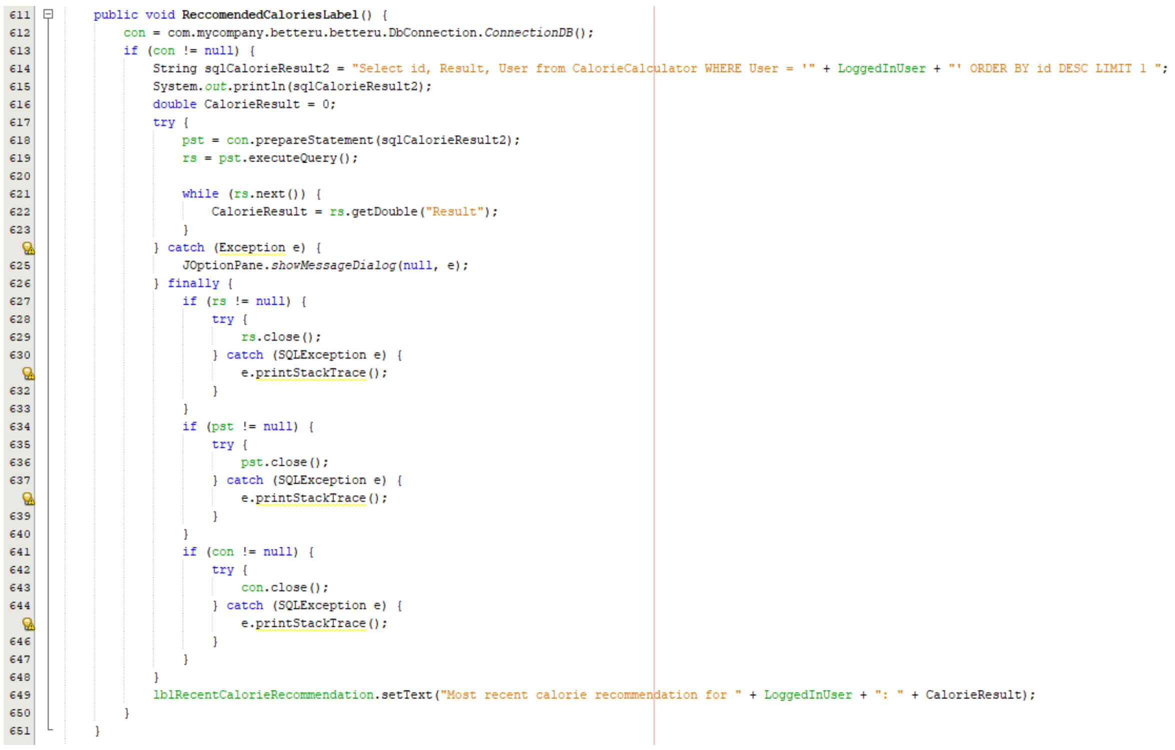
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

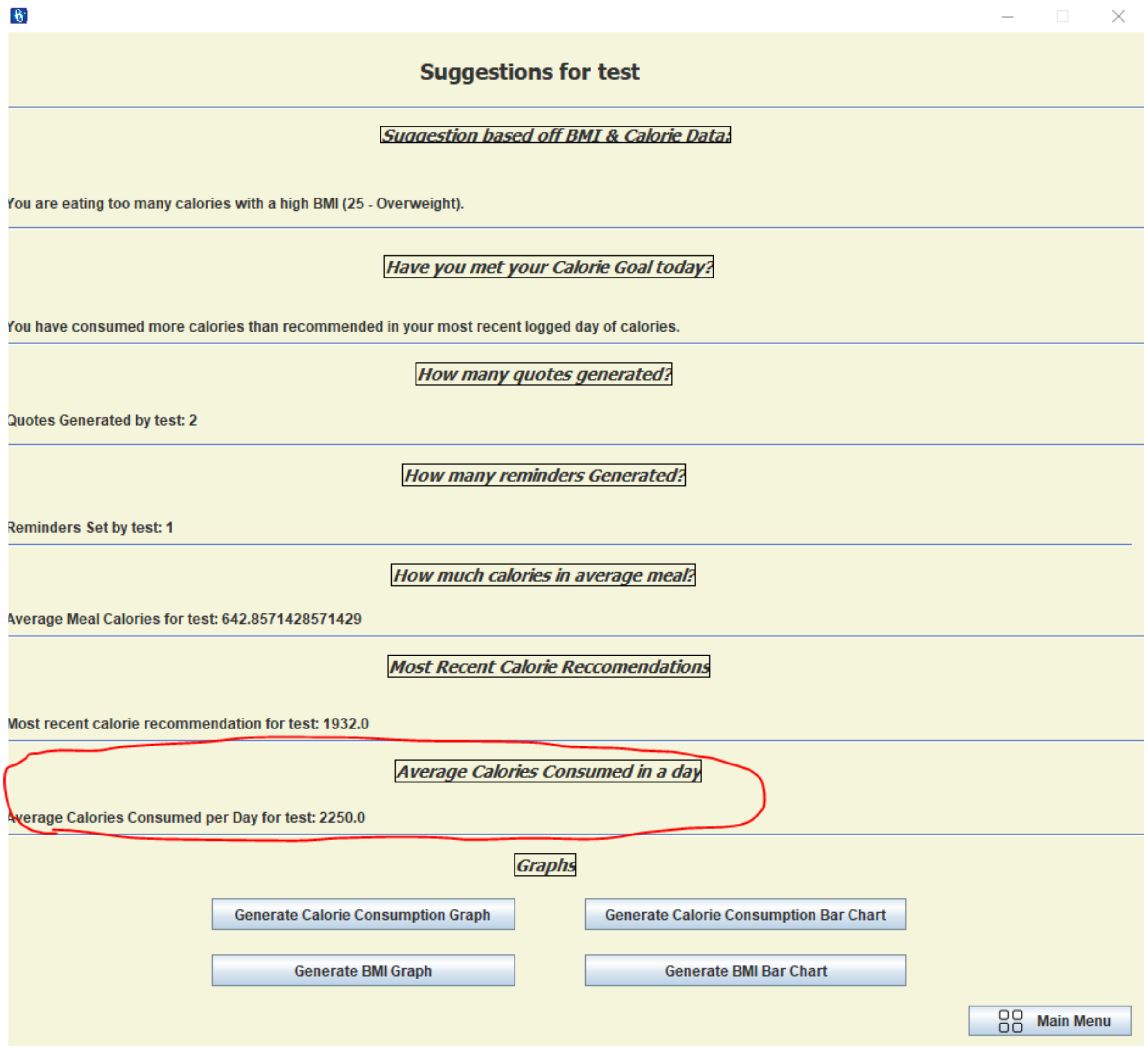
This stat gets the most recent suggestion for calories from the calorie calucator feature. Below is the code for this.



```
611     public void ReccomendedCaloriesLabel() {
612         con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
613         if (con != null) {
614             String sqlCalorieResult2 = "Select id, Result, User from CalorieCalculator WHERE User = '" + LoggedInUser + "' ORDER BY id DESC LIMIT 1 ";
615             System.out.println(sqlCalorieResult2);
616             double CalorieResult = 0;
617             try {
618                 pst = con.prepareStatement(sqlCalorieResult2);
619                 rs = pst.executeQuery();
620 
621                 while (rs.next()) {
622                     CalorieResult = rs.getDouble("Result");
623                 }
                } catch (Exception e) {
625                 JOptionPane.showMessageDialog(null, e);
626             } finally {
627                 if (rs != null) {
628                     try {
629                         rs.close();
630                     } catch (SQLException e) {
                            e.printStackTrace();
632                     }
633                 }
634                 if (pst != null) {
635                     try {
636                         pst.close();
637                     } catch (SQLException e) {
                            e.printStackTrace();
639                     }
640                 }
641                 if (con != null) {
642                     try {
643                         con.close();
644                     } catch (SQLException e) {
                            e.printStackTrace();
646                     }
647                 }
648             }
649             lblRecentCalorieRecommendation.setText("Most recent calorie recommendation for " + LoggedInUser + ": " + CalorieResult);
650         }
651     }
```

As seen on line 614 this uses a query where it gets the result the from the calorie clcaulator table by using a select statement and using the order by id DESC LIMIT 1 to ensure it is the most recent result.

I will now move onto the last stat and suggestion this is the Average calories consumed in a day.


Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

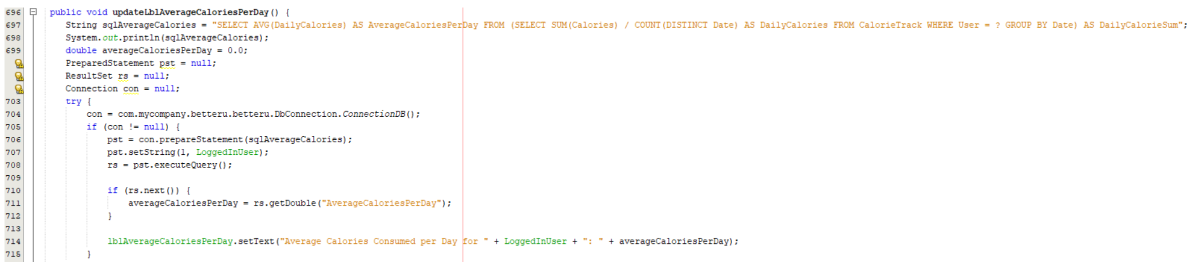
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

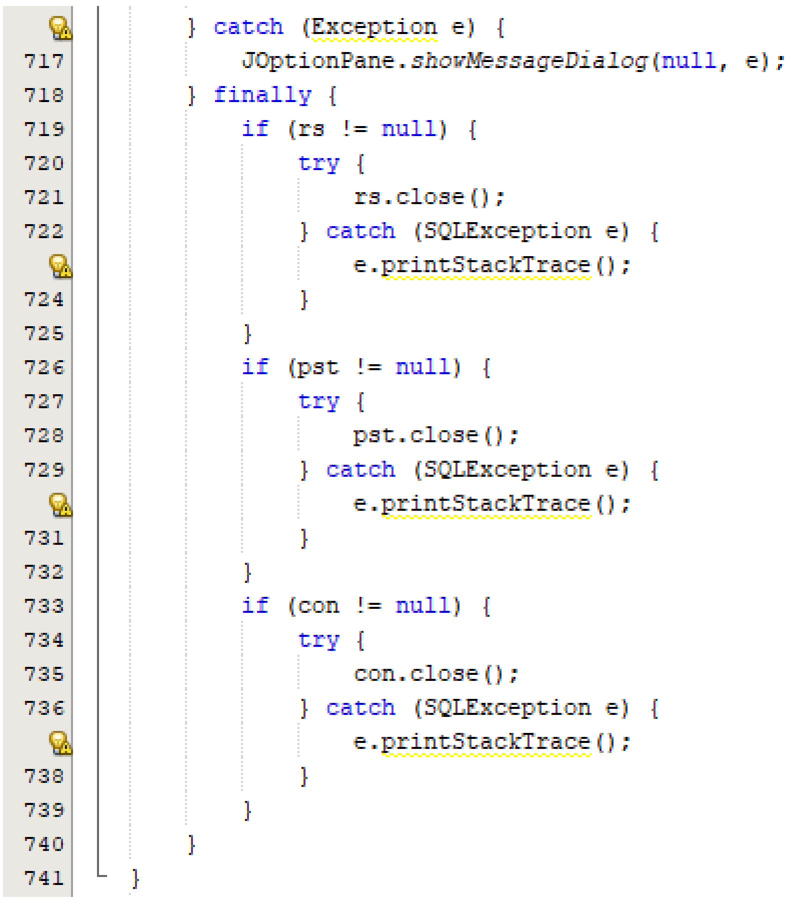
Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

Below is the code for this stat.


```java
public void updateLblAverageCaloriesPerDay() {
    String sqlAverageCalories = "SELECT AVG(DailyCalories) AS AverageCaloriesPerDay FROM (SELECT SUM(Calories) / COUNT(DISTINCT Date) AS DailyCalories FROM CalorieTrack WHERE User = ? GROUP BY Date) AS DailyCalorieSum";
    System.out.println(sqlAverageCalories);
    double averageCaloriesPerDay = 0.0;
    PreparedStatement pst = null;
    ResultSet rs = null;
    Connection con = null;
    try {
        con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
        if (con != null) {
            pst = con.prepareStatement(sqlAverageCalories);
            pst.setString(1, LoggedInUser);
            rs = pst.executeQuery();

            if (rs.next()) {
                averageCaloriesPerDay = rs.getDouble("AverageCaloriesPerDay");
            }

            lblAverageCaloriesPerDay.setText("Average Calories Consumed per Day for " + LoggedInUser + ": " + averageCaloriesPerDay);
        }
```

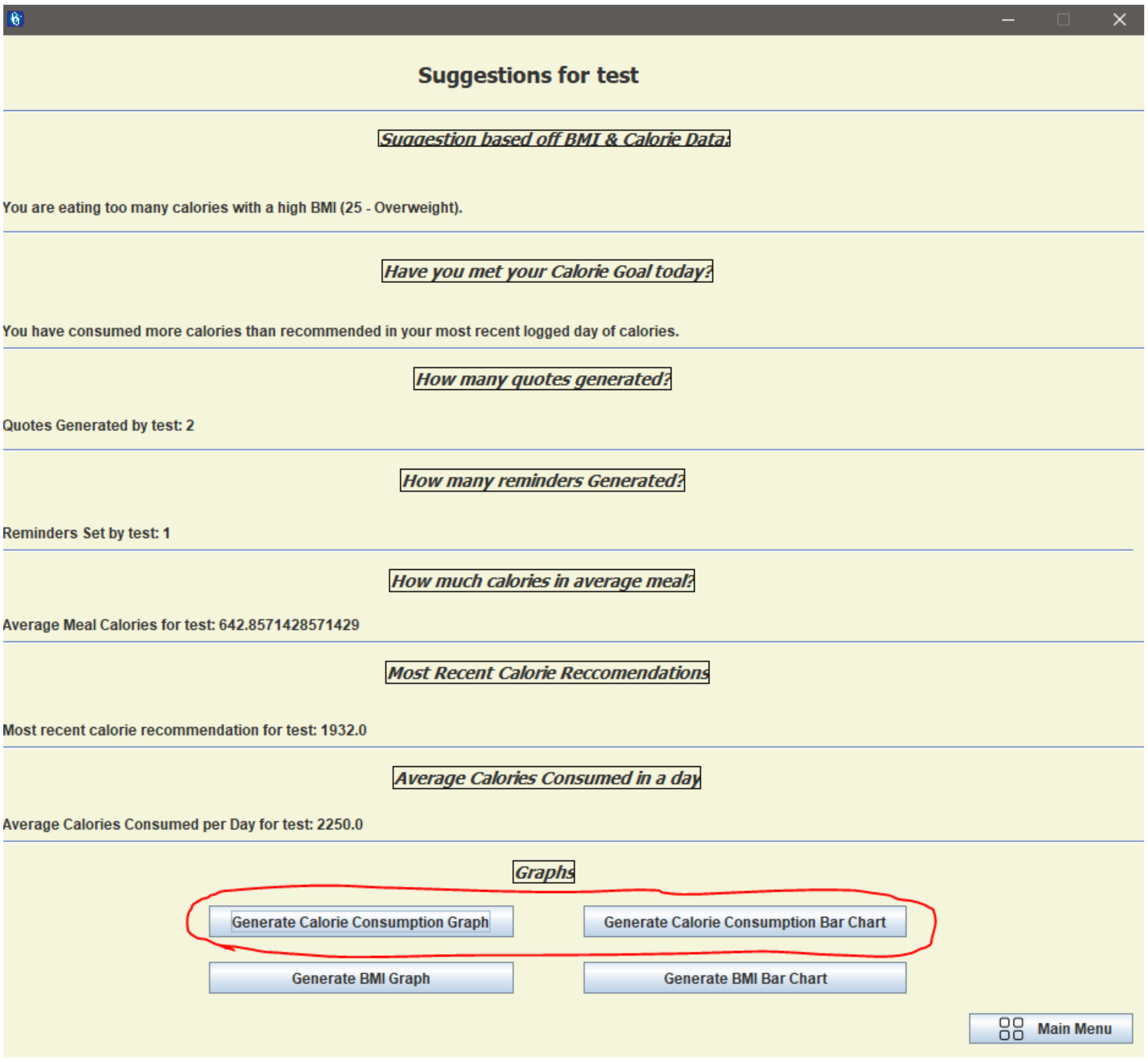
This code has quite a big query. To explain this query its good to think about it as if it was broken up. It makes it easier to read. It can be looked at as 2 querys. The first is selecting the sum of the calories consumed on each date. It is then counting the distinct dates in the table. What then happens is the sum of the calories is devided by the number of distinct dates and this give us the average calories consumed each day. Now that the calcuations are done the second part of the statement which is actually at the beginning of the query then selects the Avg(DailyCalories) AS AverageCaloriesPerDay note how the frst part of the Query I explained is in brackets. This is called a nested query.



```
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e);
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (pst != null) {
                try {
                    pst.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (con != null) {
                try {
                    con.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

At the end here the connections are closed.

It is now time to go through the graphs and bar charts. As seen on the stats and suggestions page there is a graph and a bar chart for the calorie data and there is a graph and a bar chart used for the BMI data. I will explain the bar chat and the graph for the Calorie tracker. I will not do the BMI as these are both similar.


Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

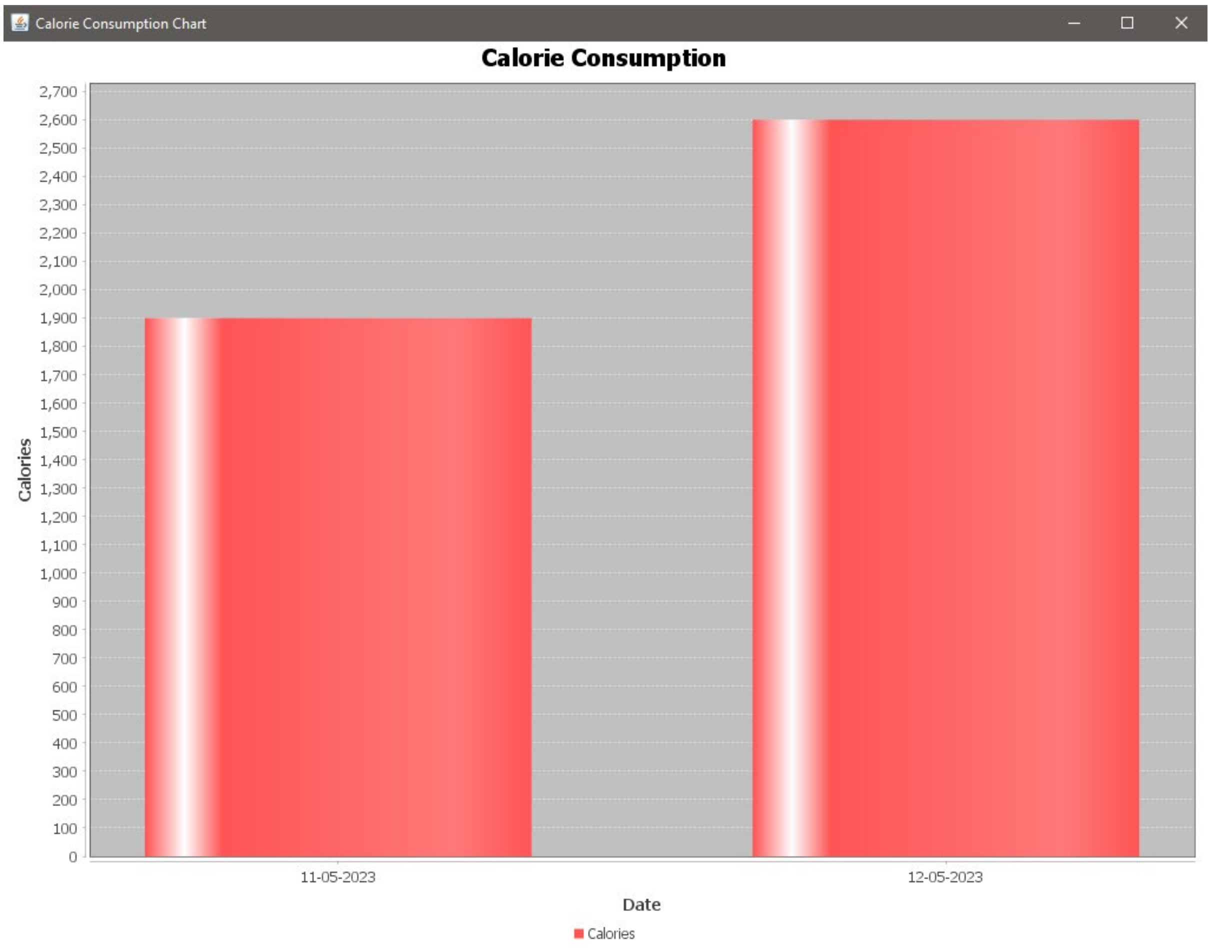
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

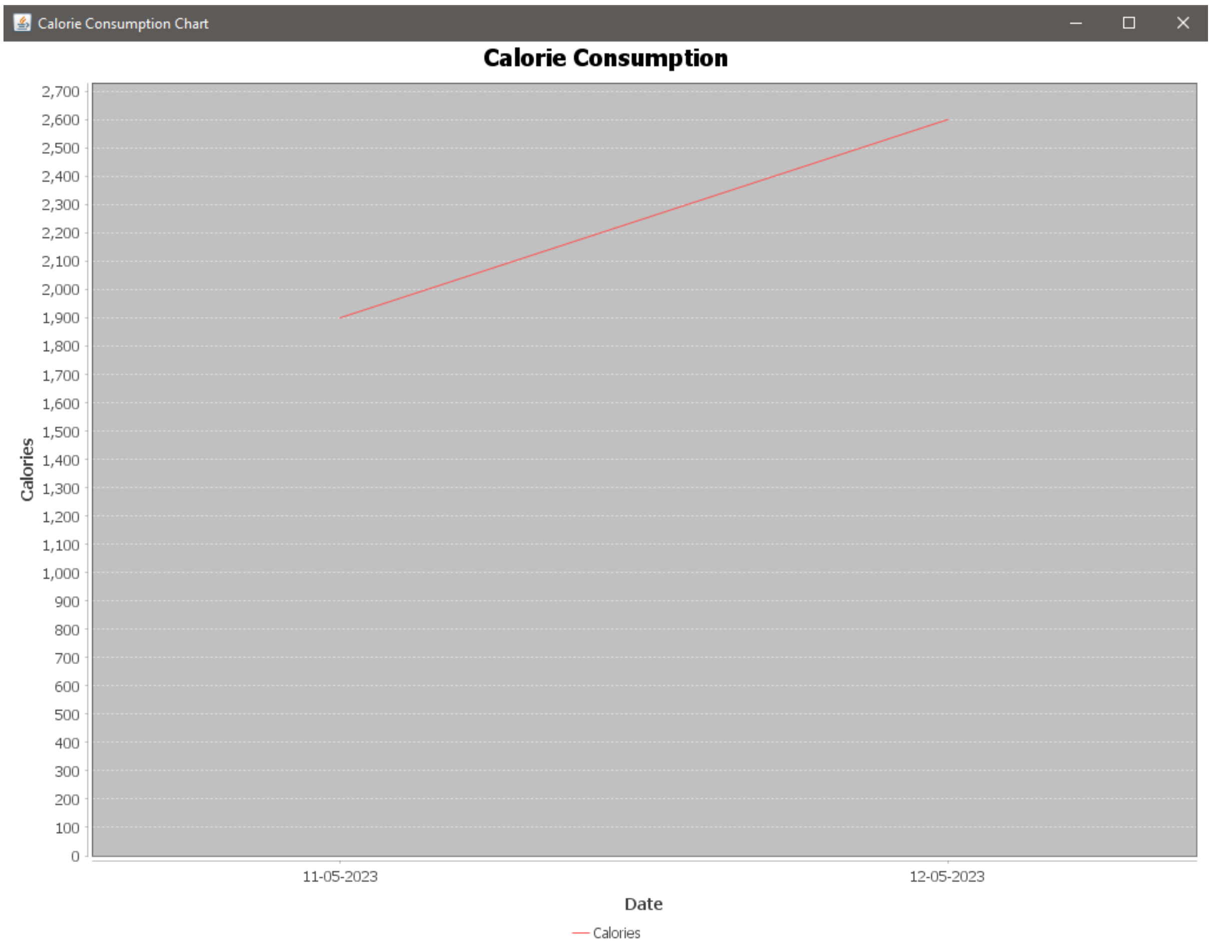
Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

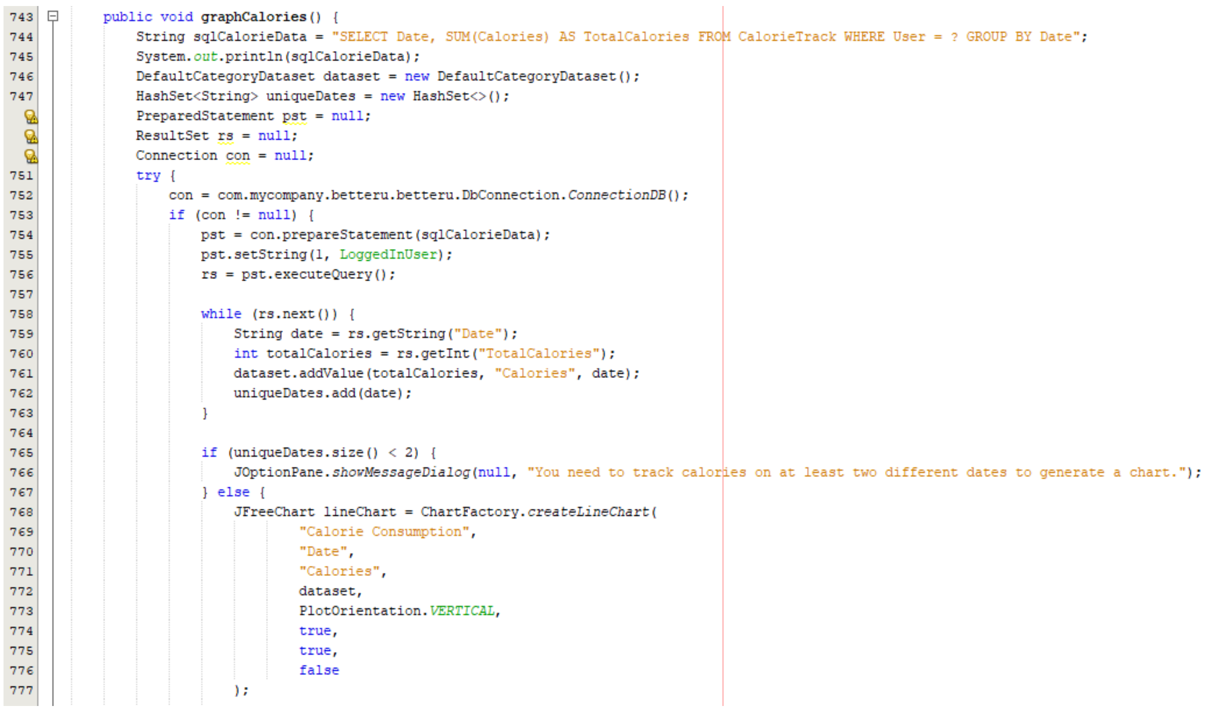
When these are ran this is what it looks like. First ill do the bar chart. Bar chat below.



Next is the calorie consumption graph. Graph below.



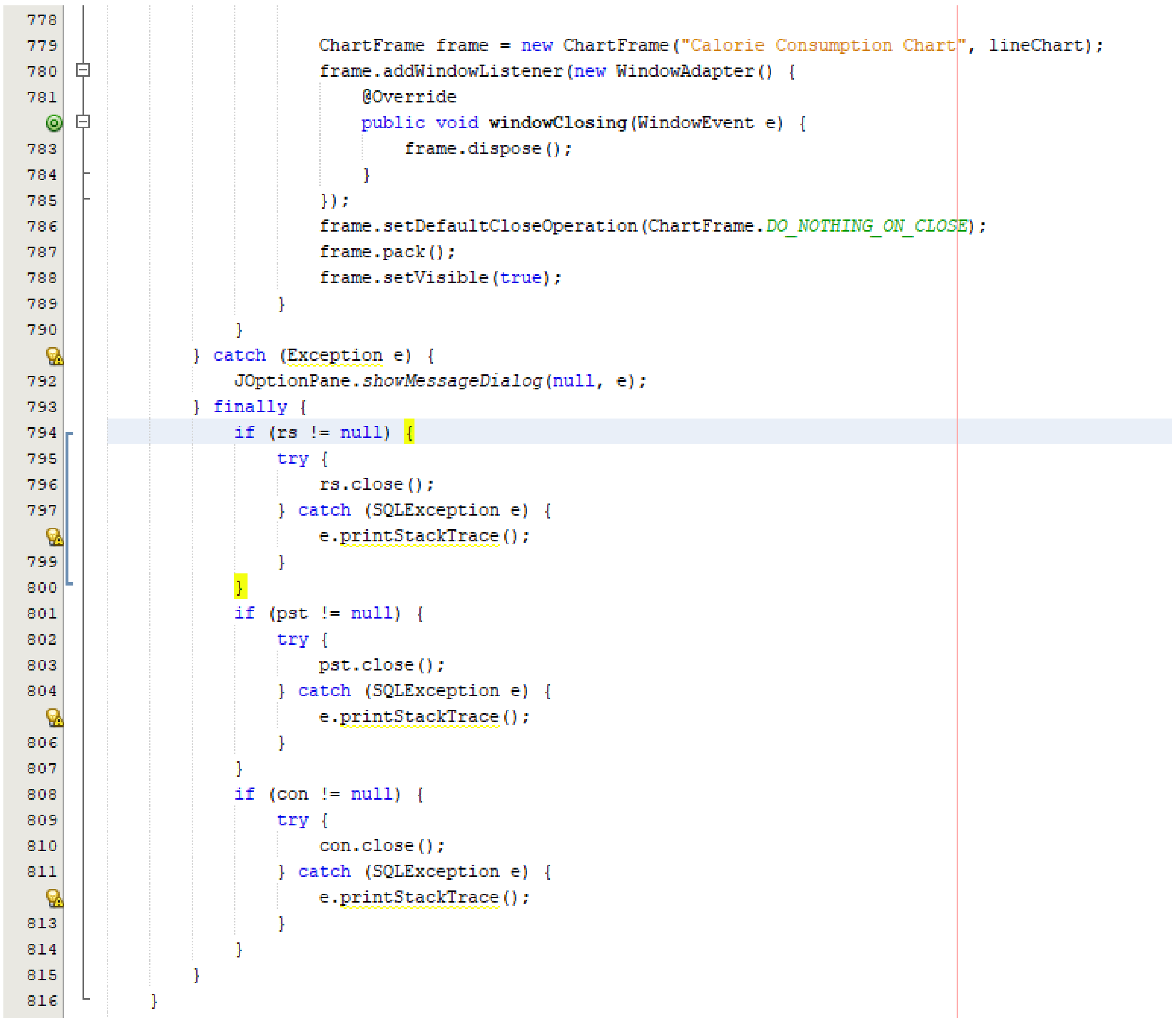
Below I will show the code for the graph. The graph is made using the JFreeChart dependency.



```
743     public void graphCalories() {
744         String sqlCalorieData = "SELECT Date, SUM(Calories) AS TotalCalories FROM CalorieTrack WHERE User = ? GROUP BY Date";
745         System.out.println(sqlCalorieData);
746         DefaultCategoryDataset dataset = new DefaultCategoryDataset();
747         HashSet<String> uniqueDates = new HashSet<>();
            PreparedStatement pst = null;
            ResultSet rs = null;
            Connection con = null;
751         try {
752             con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
753             if (con != null) {
754                 pst = con.prepareStatement(sqlCalorieData);
755                 pst.setString(1, LoggedInUser);
756                 rs = pst.executeQuery();
757 
758                 while (rs.next()) {
759                     String date = rs.getString("Date");
760                     int totalCalories = rs.getInt("TotalCalories");
761                     dataset.addValue(totalCalories, "Calories", date);
762                     uniqueDates.add(date);
763                 }
764 
765                 if (uniqueDates.size() < 2) {
766                     JOptionPane.showMessageDialog(null, "You need to track calories on at least two different dates to generate a chart.");
767                 } else {
768                     JFreeChart lineChart = ChartFactory.createLineChart(
769                             "Calorie Consumption",
770                             "Date",
771                             "Calories",
772                             dataset,
773                             PlotOrientation.VERTICAL,
774                             true,
775                             true,
776                             false
777                     );
```

The query on line 744 is getting the total number of calories per date for user for the graph.

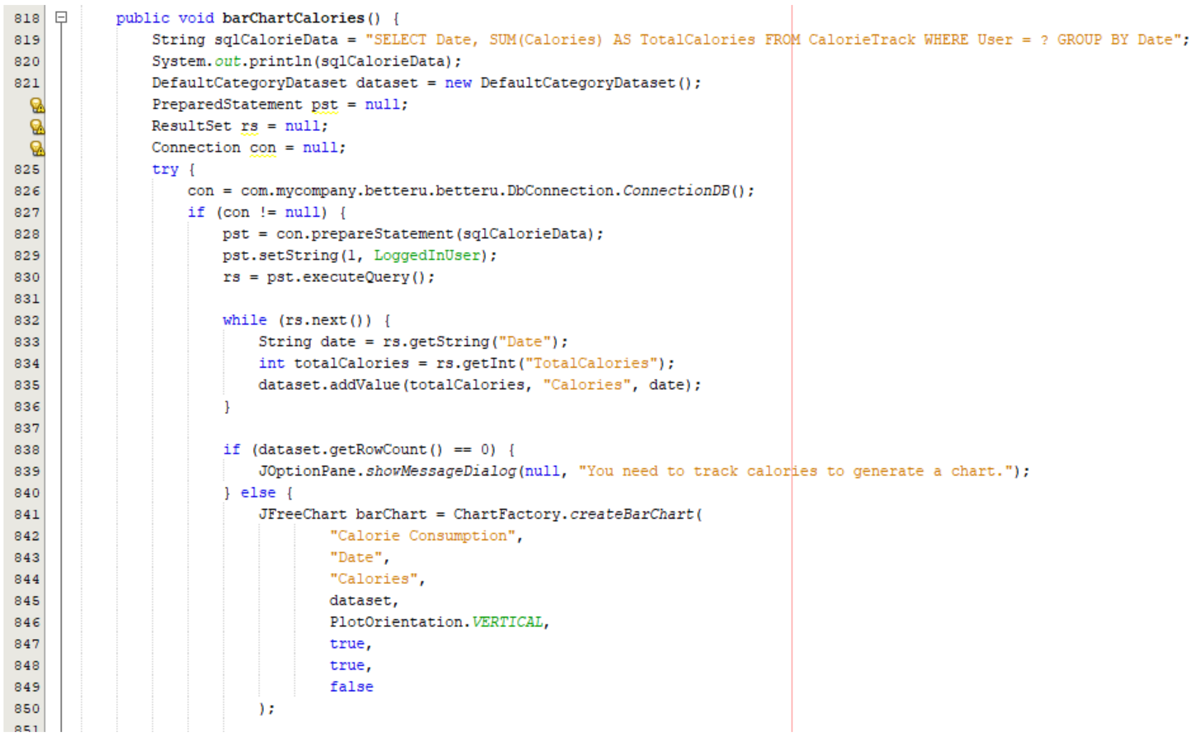
Providing the user has calories on at least 2 days a user can generate a chart. The chart frame is established on line 768.



```
778
779                     ChartFrame frame = new ChartFrame("Calorie Consumption Chart", lineChart);
780                     frame.addWindowListener(new WindowAdapter() {
781                         @Override
                            public void windowClosing(WindowEvent e) {
783                             frame.dispose();
784                         }
785                     });
786                     frame.setDefaultCloseOperation(ChartFrame.DO_NOTHING_ON_CLOSE);
787                     frame.pack();
788                     frame.setVisible(true);
789                 }
790             }
            } catch (Exception e) {
792             JOptionPane.showMessageDialog(null, e);
793         } finally {
794             if (rs != null) {
795                 try {
796                     rs.close();
797                 } catch (SQLException e) {
                        e.printStackTrace();
799                 }
800             }
801             if (pst != null) {
802                 try {
803                     pst.close();
804                 } catch (SQLException e) {
                        e.printStackTrace();
806                 }
807             }
808             if (con != null) {
809                 try {
810                     con.close();
811                 } catch (SQLException e) {
                        e.printStackTrace();
813                 }
814             }
815         }
816     }
```

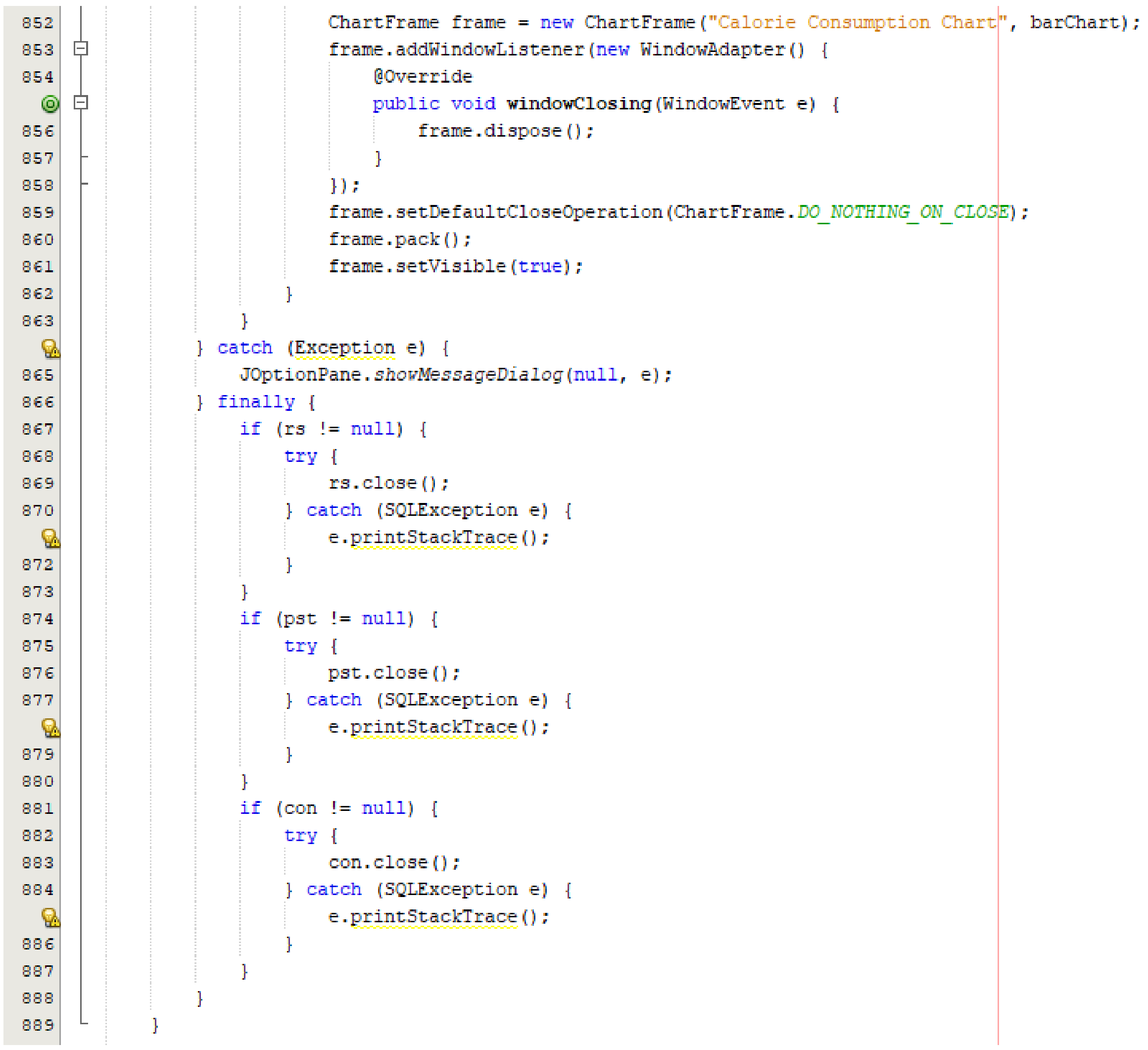
on line 779 The chart frame is created. After this the finally block closes all the resources used for the chart.

I will now show the code for the bar chart.



```
818     public void barChartCalories() {
819         String sqlCalorieData = "SELECT Date, SUM(Calories) AS TotalCalories FROM CalorieTrack WHERE User = ? GROUP BY Date";
820         System.out.println(sqlCalorieData);
821         DefaultCategoryDataset dataset = new DefaultCategoryDataset();
            PreparedStatement pst = null;
            ResultSet rs = null;
            Connection con = null;
825         try {
826             con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
827             if (con != null) {
828                 pst = con.prepareStatement(sqlCalorieData);
829                 pst.setString(1, LoggedInUser);
830                 rs = pst.executeQuery();
831
832                 while (rs.next()) {
833                     String date = rs.getString("Date");
834                     int totalCalories = rs.getInt("TotalCalories");
835                     dataset.addValue(totalCalories, "Calories", date);
836                 }
837
838                 if (dataset.getRowCount() == 0) {
839                     JOptionPane.showMessageDialog(null, "You need to track calories to generate a chart.");
840                 } else {
841                     JFreeChart barChart = ChartFactory.createBarChart(
842                             "Calorie Consumption",
843                             "Date",
844                             "Calories",
845                             dataset,
846                             PlotOrientation.VERTICAL,
847                             true,
848                             true,
849                             false
850                     );
```

On line 819 the sum of the calories for each date that is in the database is collected where the user is = LoggedInUser. In this case logged in user is test. On line 838 there is a check to make sure that there is calorie data that can be plotted to the graph. On line 841 the bar chart is established.

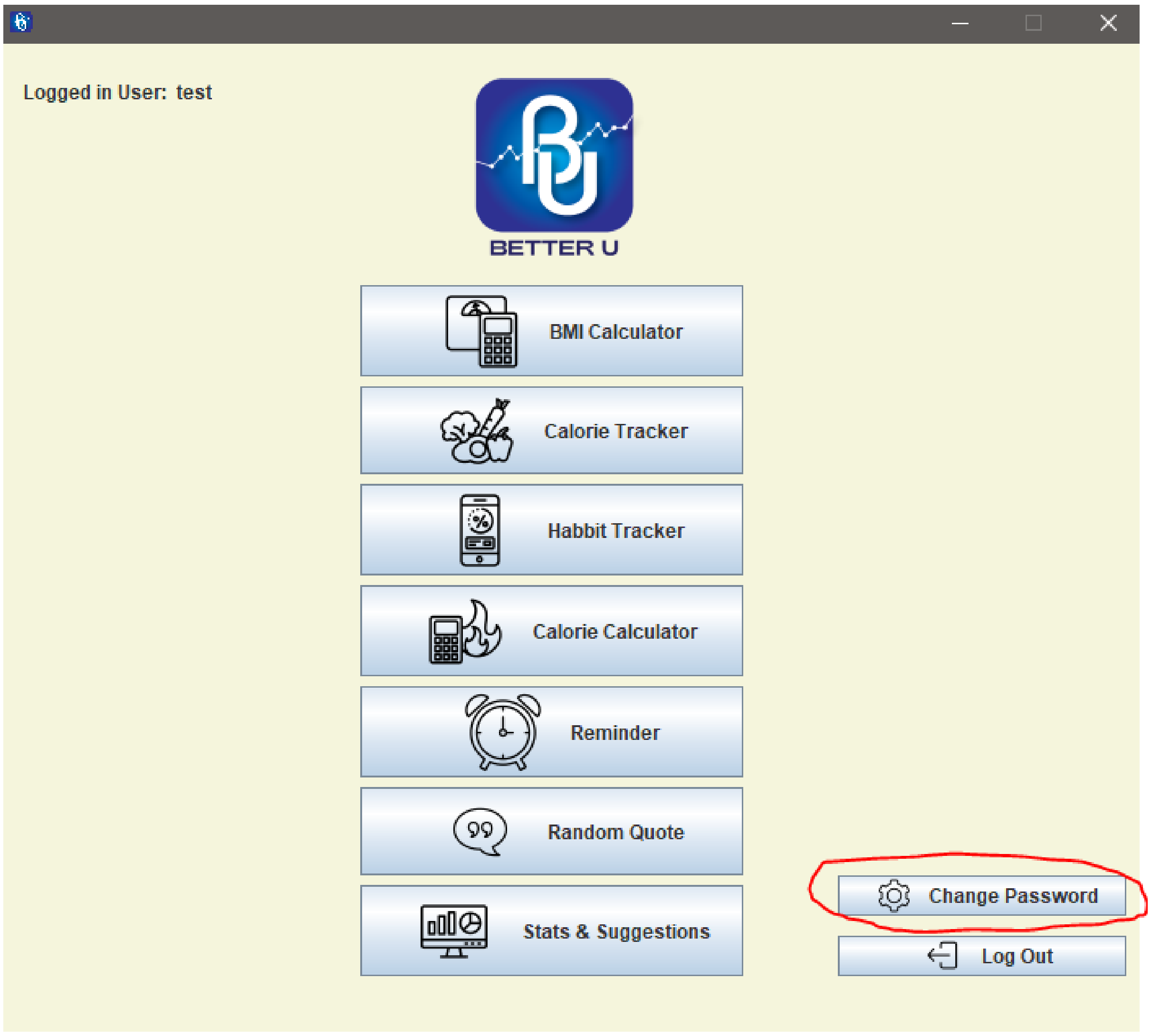


```
852                 ChartFrame frame = new ChartFrame("Calorie Consumption Chart", barChart);
853                 frame.addWindowListener(new WindowAdapter() {
854                     @Override
                        public void windowClosing(WindowEvent e) {
856                         frame.dispose();
857                     }
858                 });
859                 frame.setDefaultCloseOperation(ChartFrame.DO_NOTHING_ON_CLOSE);
860                 frame.pack();
861                 frame.setVisible(true);
862             }
863         }
        } catch (Exception e) {
865         JOptionPane.showMessageDialog(null, e);
866     } finally {
867         if (rs != null) {
868             try {
869                 rs.close();
870             } catch (SQLException e) {
                    e.printStackTrace();
872             }
873         }
874         if (pst != null) {
875             try {
876                 pst.close();
877             } catch (SQLException e) {
                    e.printStackTrace();
879             }
880         }
881         if (con != null) {
882             try {
883                 con.close();
884             } catch (SQLException e) {
                    e.printStackTrace();
886             }
887         }
888     }
889 }
```

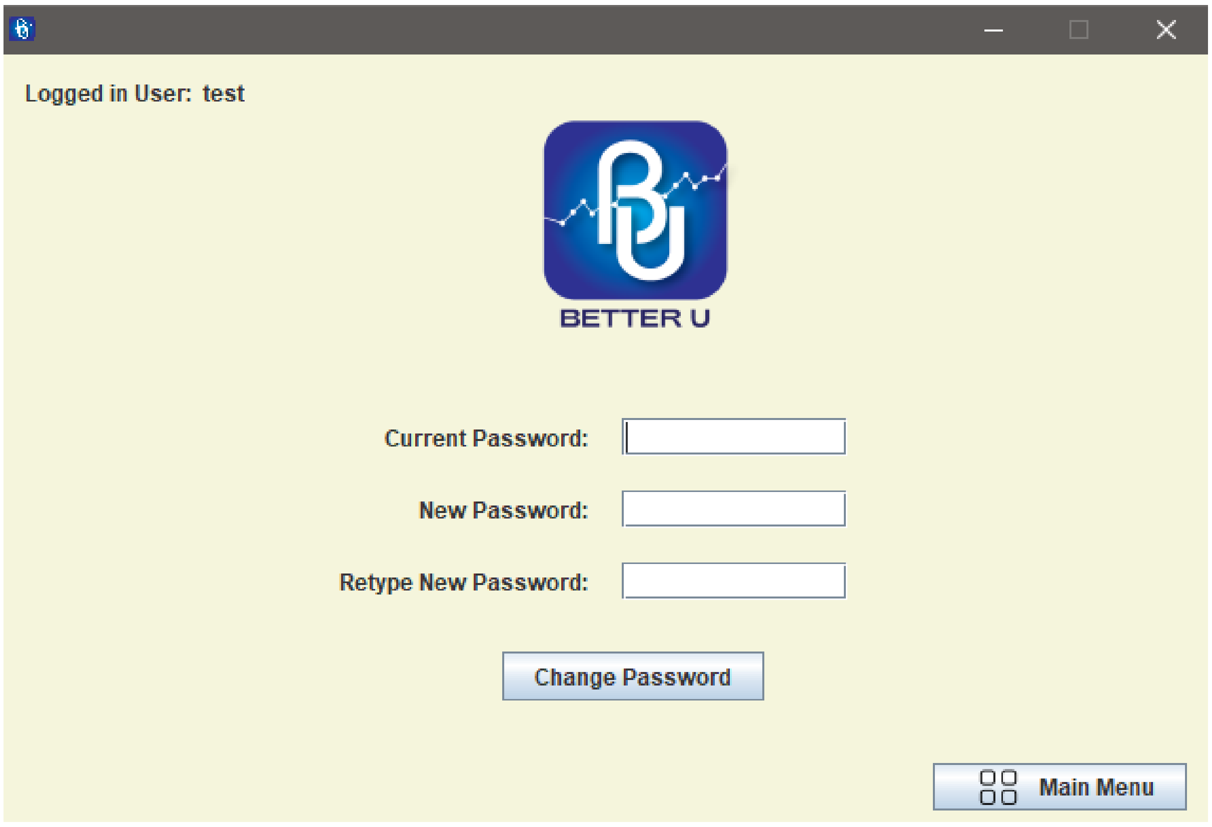
On line 852 the chart is created. A new frame is set to open on the lines that follow 853-863. The finally block then closes the resources needed for creating the graph.

This concludes my stats and suggestions page

The very last thing to shown on the menu is the change password button.



Upon opening up the change password button here is the window.

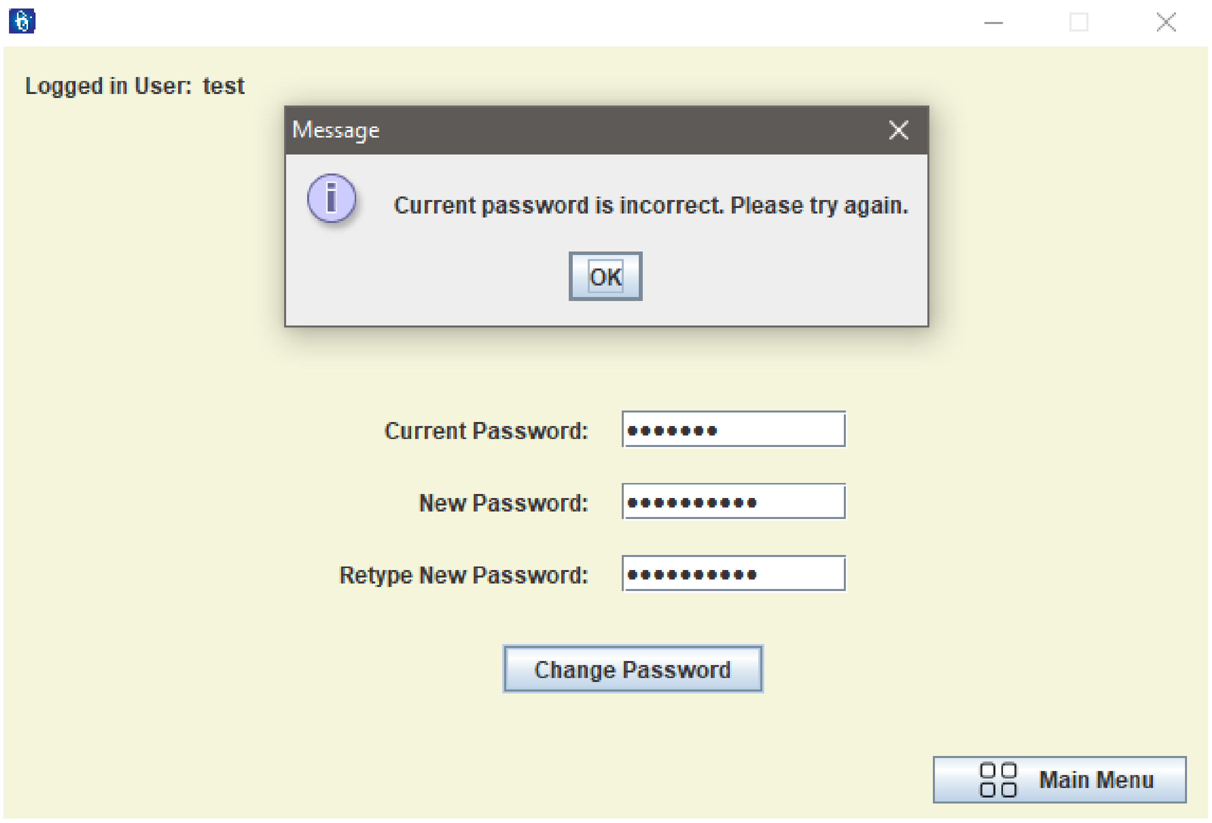

Logged in User: test

Current Password:

New Password:

Retype New Password:

The user has to enter in their current password, new password and to retype their new password. If a user fills in this information their password is changed. I will show what haooens if a user types in the info but gets the current password wrong.


Logged in User: test

Message

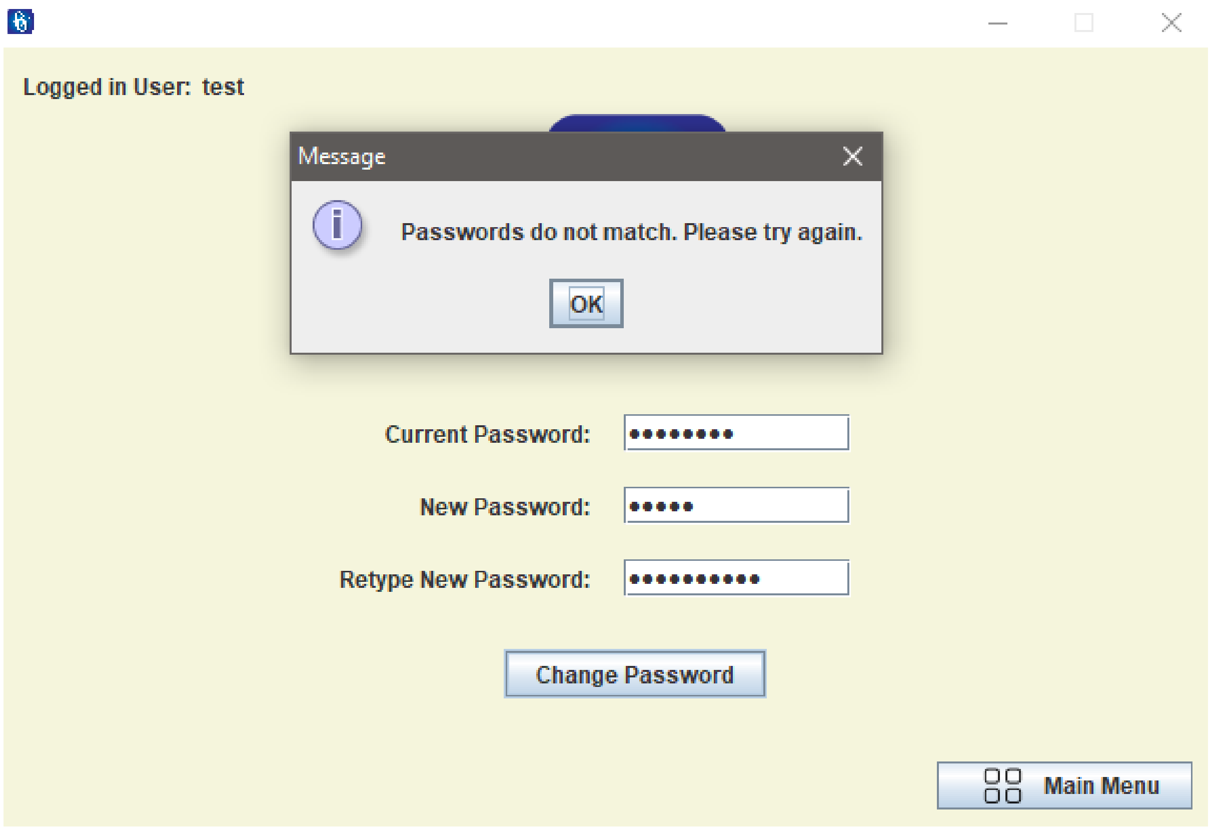
Current password is incorrect. Please try again.

Current Password: •••••••

New Password: •••••••••••

Retype New Password: •••••••••••

If the user gets the current password right but dosnt have two passwords that match for the new password is below.



Logged in User: test

Message

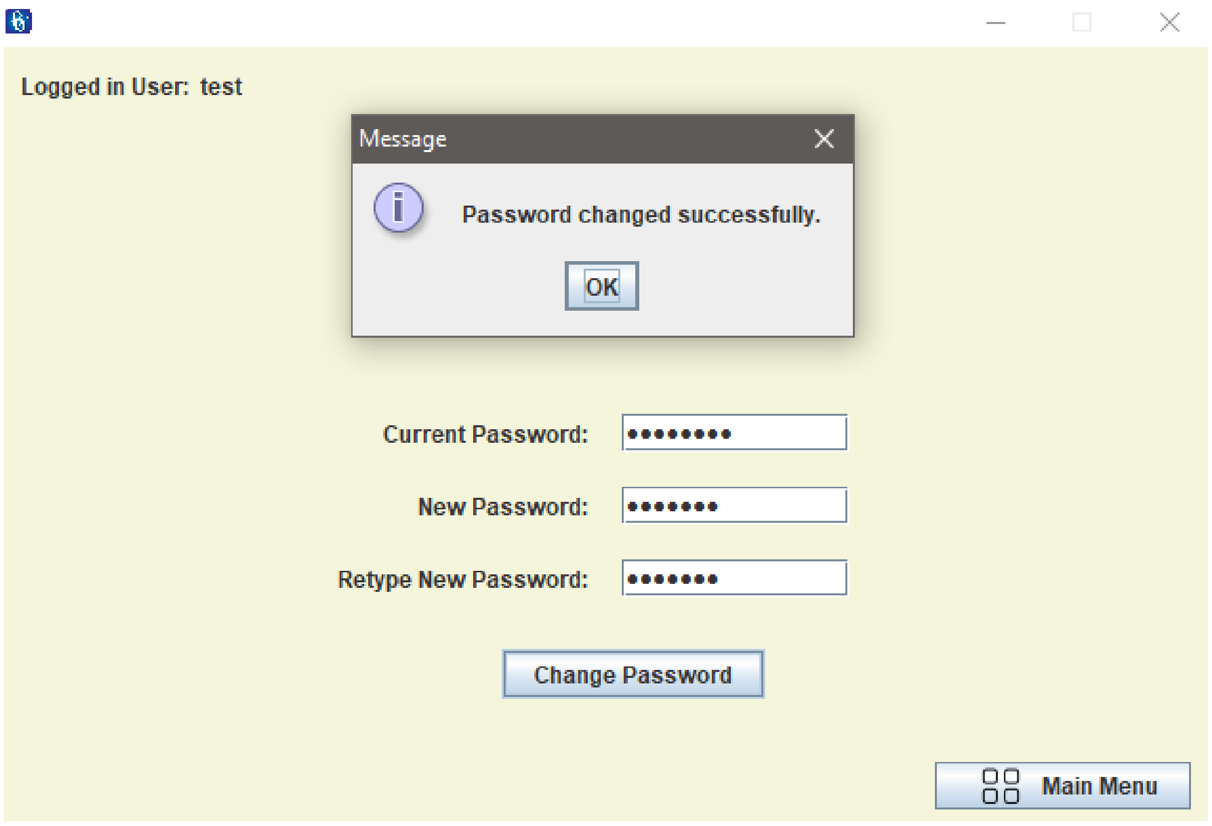
Passwords do not match. Please try again.

Current Password: ●●●●●●●●

New Password: ●●●●●

Retype New Password: ●●●●●●●●●●

Now its time for a succuessful change.



Logged in User: test

Message

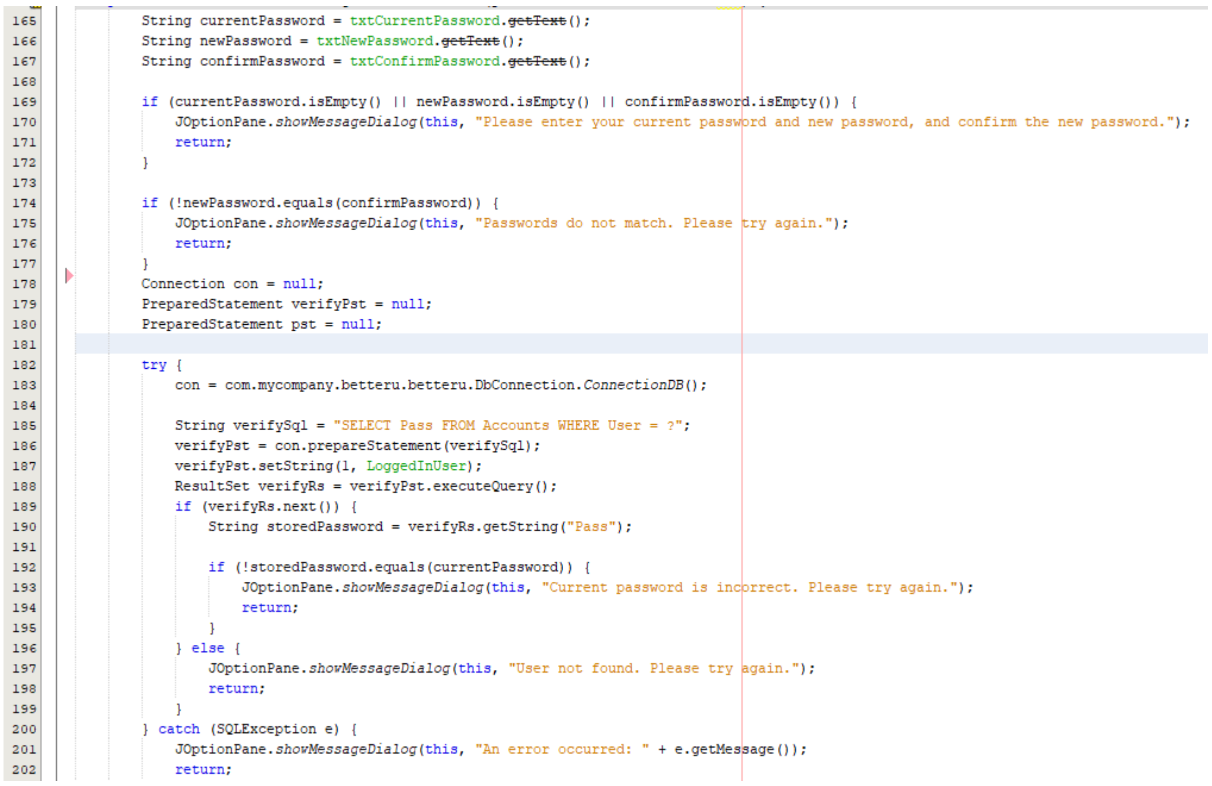
Password changed successfully.

Current Password: ●●●●●●●●

New Password: ●●●●●●●

Retype New Password: ●●●●●●●

I will now show the code behind the change password button.



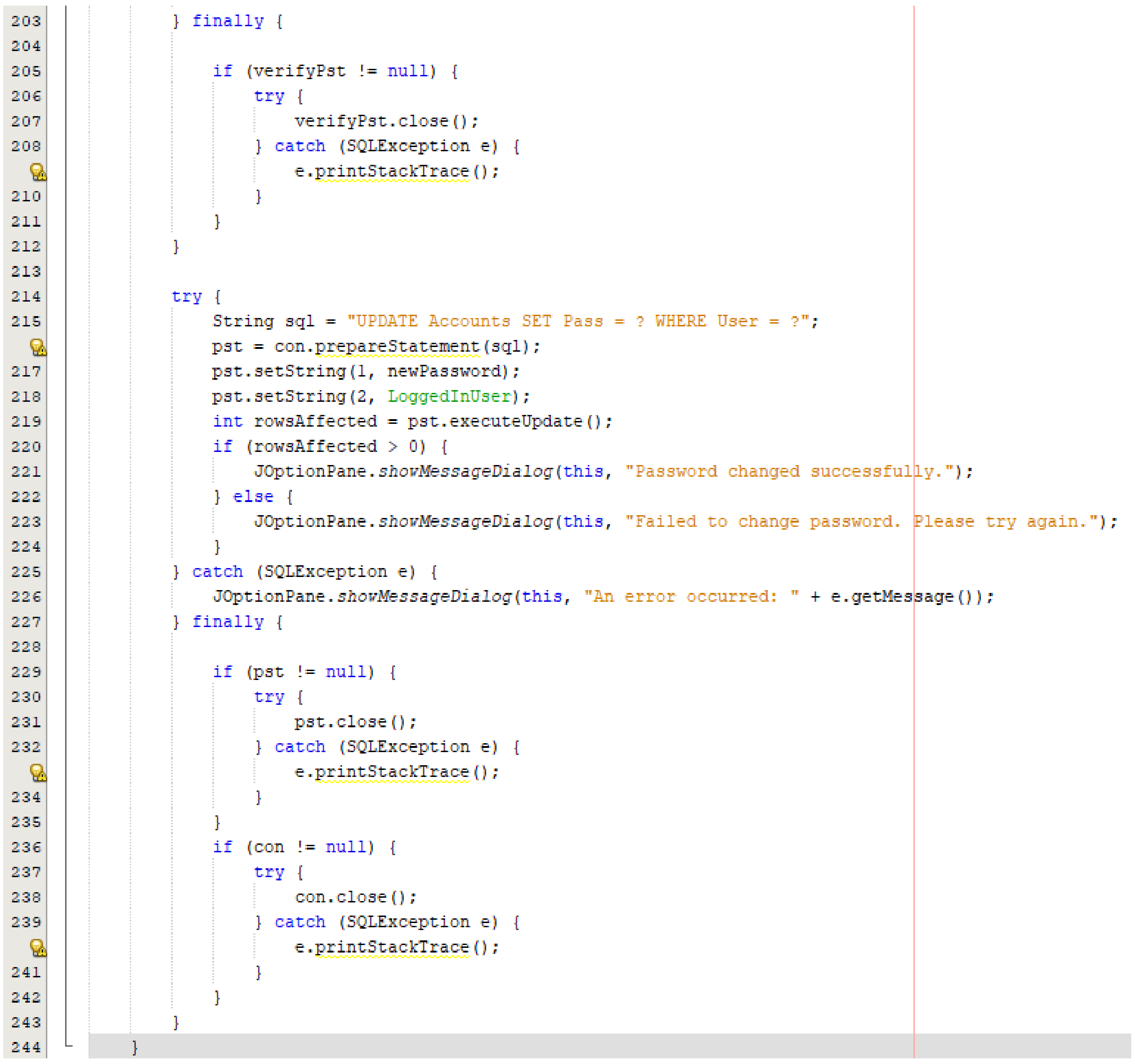
```
165         String currentPassword = txtCurrentPassword.~~getText~~();
166         String newPassword = txtNewPassword.~~getText~~();
167         String confirmPassword = txtConfirmPassword.~~getText~~();
168 
169         if (currentPassword.isEmpty() || newPassword.isEmpty() || confirmPassword.isEmpty()) {
170             JOptionPane.showMessageDialog(this, "Please enter your current password and new password, and confirm the new password.");
171             return;
172         }
173 
174         if (!newPassword.equals(confirmPassword)) {
175             JOptionPane.showMessageDialog(this, "Passwords do not match. Please try again.");
176             return;
177         }
178         Connection con = null;
179         PreparedStatement verifyPst = null;
180         PreparedStatement pst = null;
181 
182         try {
183             con = com.mycompany.betteru.betteru.DbConnection.ConnectionDB();
184 
185             String verifySql = "SELECT Pass FROM Accounts WHERE User = ?";
186             verifyPst = con.prepareStatement(verifySql);
187             verifyPst.setString(1, LoggedInUser);
188             ResultSet verifyRs = verifyPst.executeQuery();
189             if (verifyRs.next()) {
190                 String storedPassword = verifyRs.getString("Pass");
191 
192                 if (!storedPassword.equals(currentPassword)) {
193                     JOptionPane.showMessageDialog(this, "Current password is incorrect. Please try again.");
194                     return;
195                 }
196             } else {
197                 JOptionPane.showMessageDialog(this, "User not found. Please try again.");
198                 return;
199             }
200         } catch (SQLException e) {
201             JOptionPane.showMessageDialog(this, "An error occurred: " + e.getMessage());
202             return;
```

At the beginning of this code we have the 3 fields for the password getting taken in. This is the current password the new password and retype new password. These are getting assigned to variables. This is on line 165-167.

Line 169-172 checks to see if a any of these fields are empty and if they are the system prompts the user to please enter curent password, new password and to retype their new password.

Line 174-176 makes sure that the new password and the retype new password field match echother. If they don't the user is prompted that they don't.

The variables for database connection are getting initialised on line 178-180. On line 182, a try block is runnning which contains the database connection. The database connection is on line 183. Line 185 gets the password for the current user so that it can be verified against what the user typed in. Line 192 verifies if the current password is correct. If it is not it prompts the user that the password is incorrect.



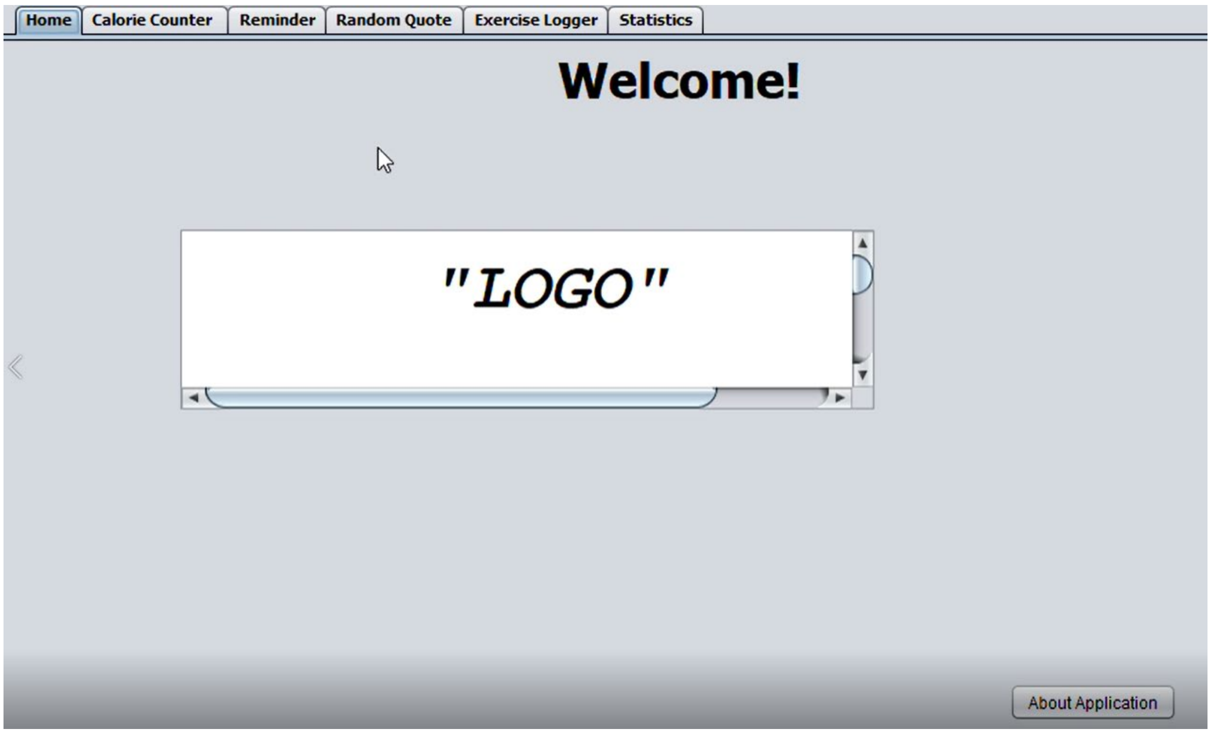
```
203         } finally {
204 
205             if (verifyPst != null) {
206                 try {
207                     verifyPst.close();
208                 } catch (SQLException e) {
                        e.printStackTrace();
210                 }
211             }
212         }
213 
214         try {
215             String sql = "UPDATE Accounts SET Pass = ? WHERE User = ?";
                pst = con.prepareStatement(sql);
217             pst.setString(1, newPassword);
218             pst.setString(2, LoggedInUser);
219             int rowsAffected = pst.executeUpdate();
220             if (rowsAffected > 0) {
221                 JOptionPane.showMessageDialog(this, "Password changed successfully.");
222             } else {
223                 JOptionPane.showMessageDialog(this, "Failed to change password. Please try again.");
224             }
225         } catch (SQLException e) {
226             JOptionPane.showMessageDialog(this, "An error occurred: " + e.getMessage());
227         } finally {
228 
229             if (pst != null) {
230                 try {
231                     pst.close();
232                 } catch (SQLException e) {
                        e.printStackTrace();
234                 }
235             }
236             if (con != null) {
237                 try {
238                     con.close();
239                 } catch (SQLException e) {
                        e.printStackTrace();
241                 }
242             }
243         }
244     }
```

On line 214 the try block contains the code where the Password is updated. Line 215 contains the statement for updating the password where user is equal to LoggedInUser. Line 220 checks to see if any rows have been changed if they have that means the password change was a success. Otherwise it will tell the user that the password change was not a success. There is a catch block to catch the SQL Exception on line 225. After this the finally block closes the resources. This concludes the implementation of the system.

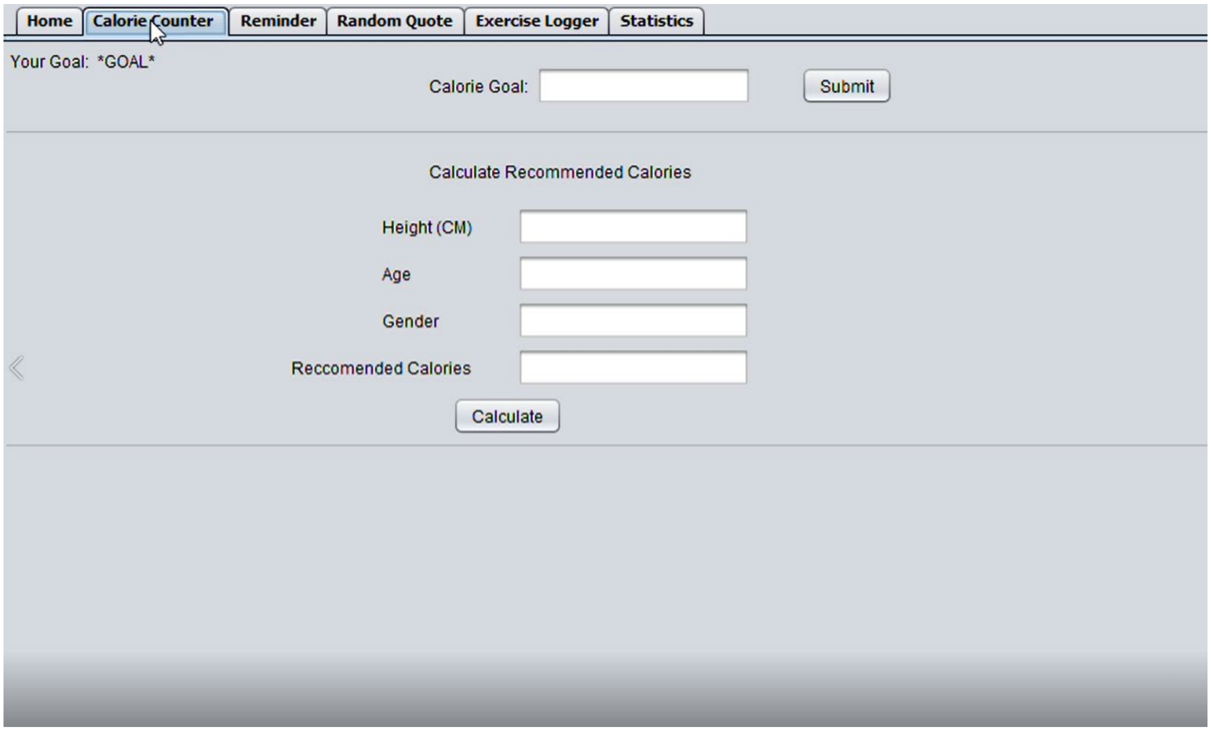
2.5. Graphical User Interface (GUI)

In this section I will discuss the GUI elements of my program. I had done a lot of this during my implementation so I will keep this section brief. I will first start off by showing screenshots of my prototype and then I will go through my actual application and explain both.

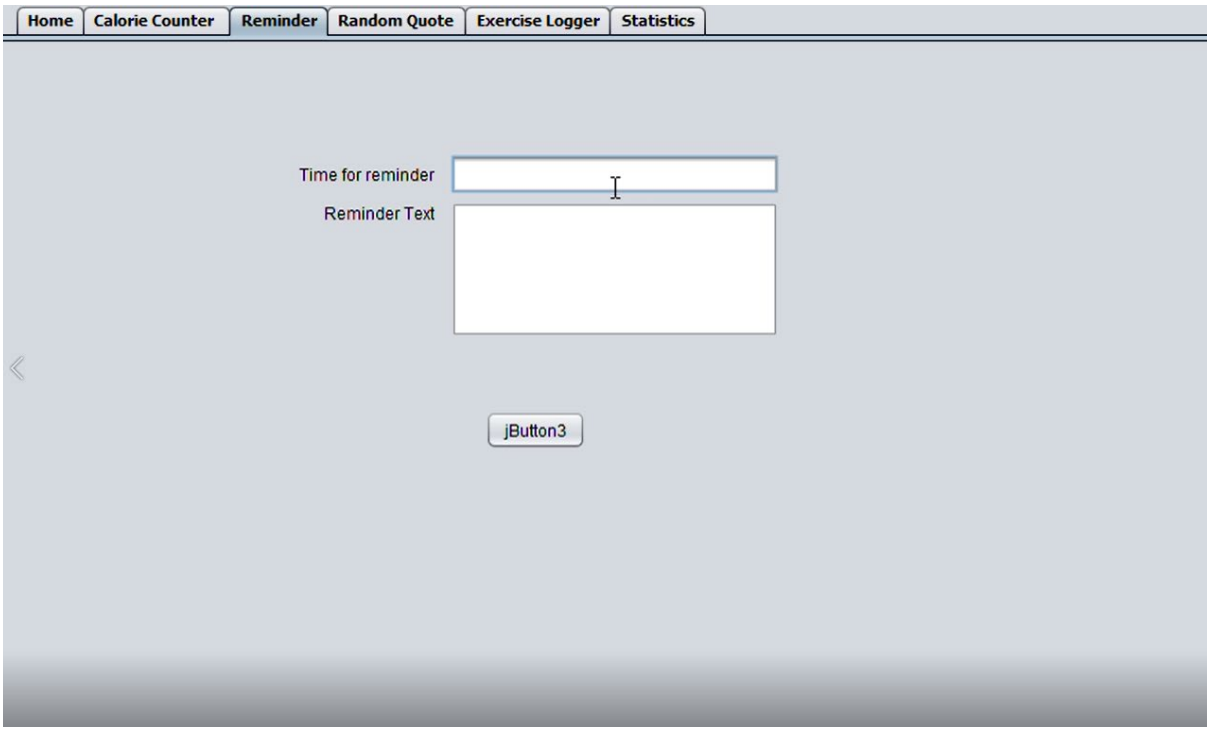
Prototype:



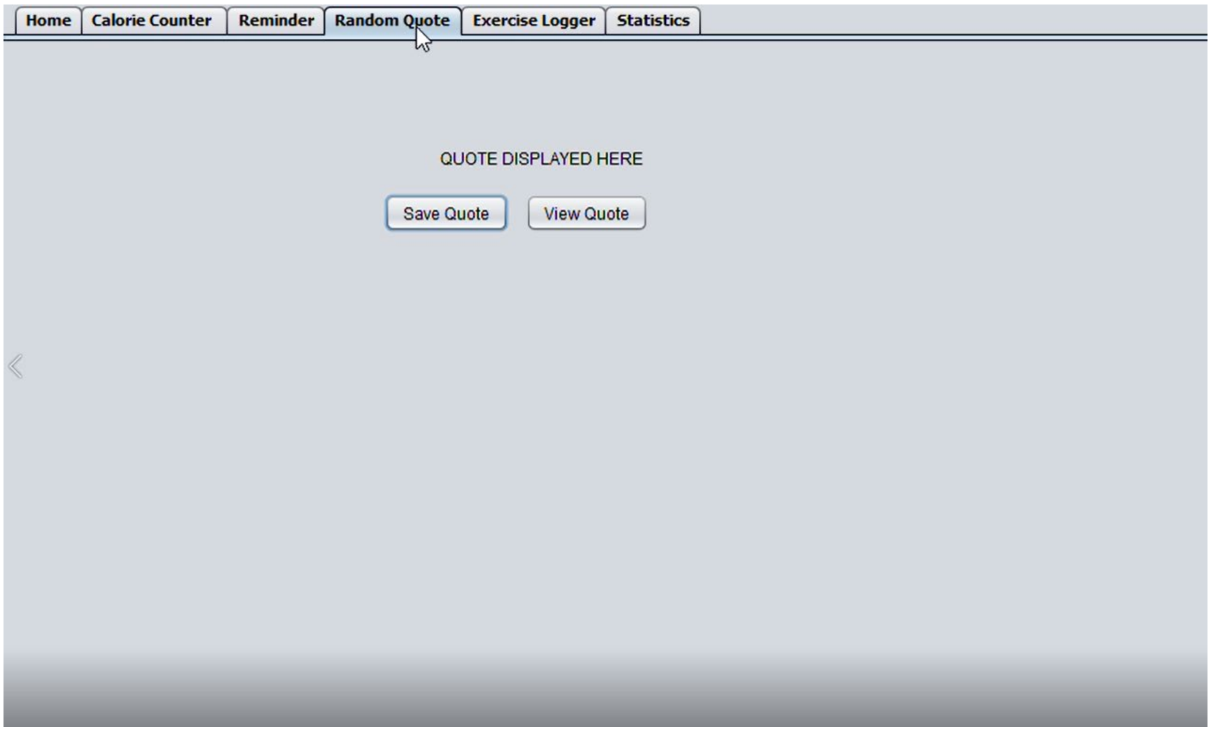
Above is the prototype home screen. As Seen in the prototype I had originally planned to use a tabbed interface. After trying this out I decided that it would be too messy to implement my project in this way so I decided against that. I decided to have a new window and class for function.



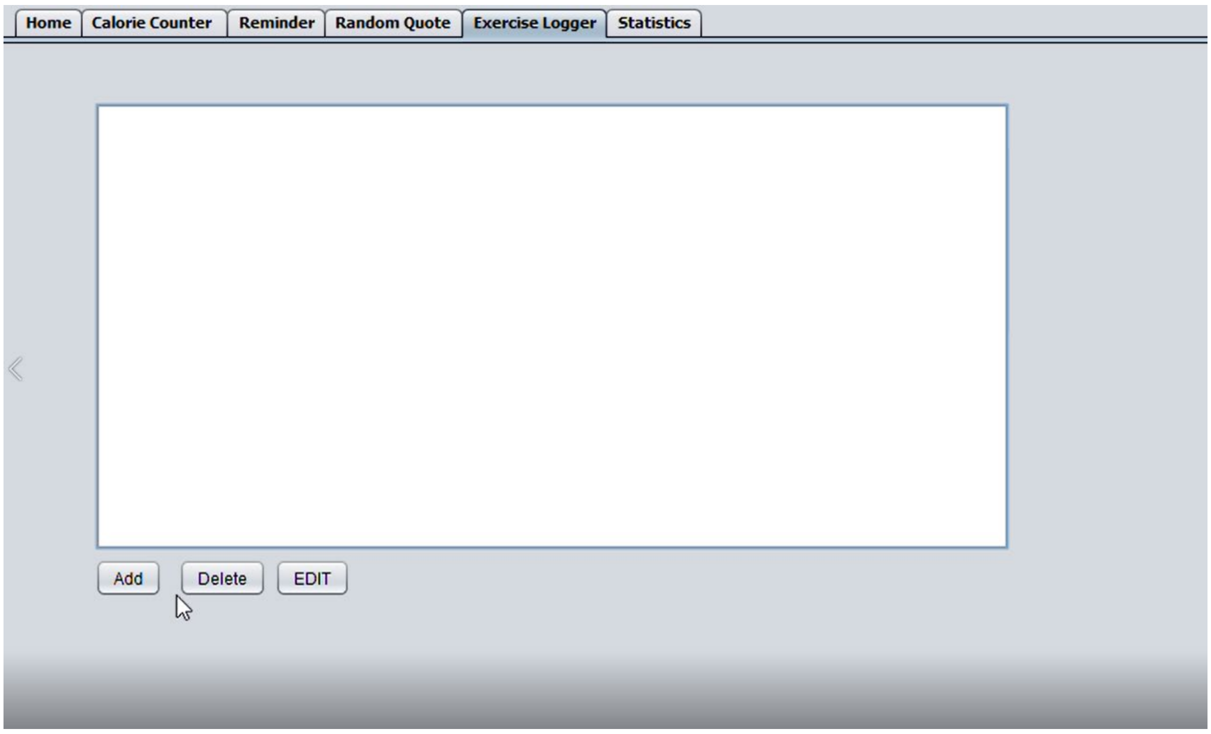
Above is the calorie counter part of my prototype this has some general fields for counting calories and also a field to enter a calorie goal.



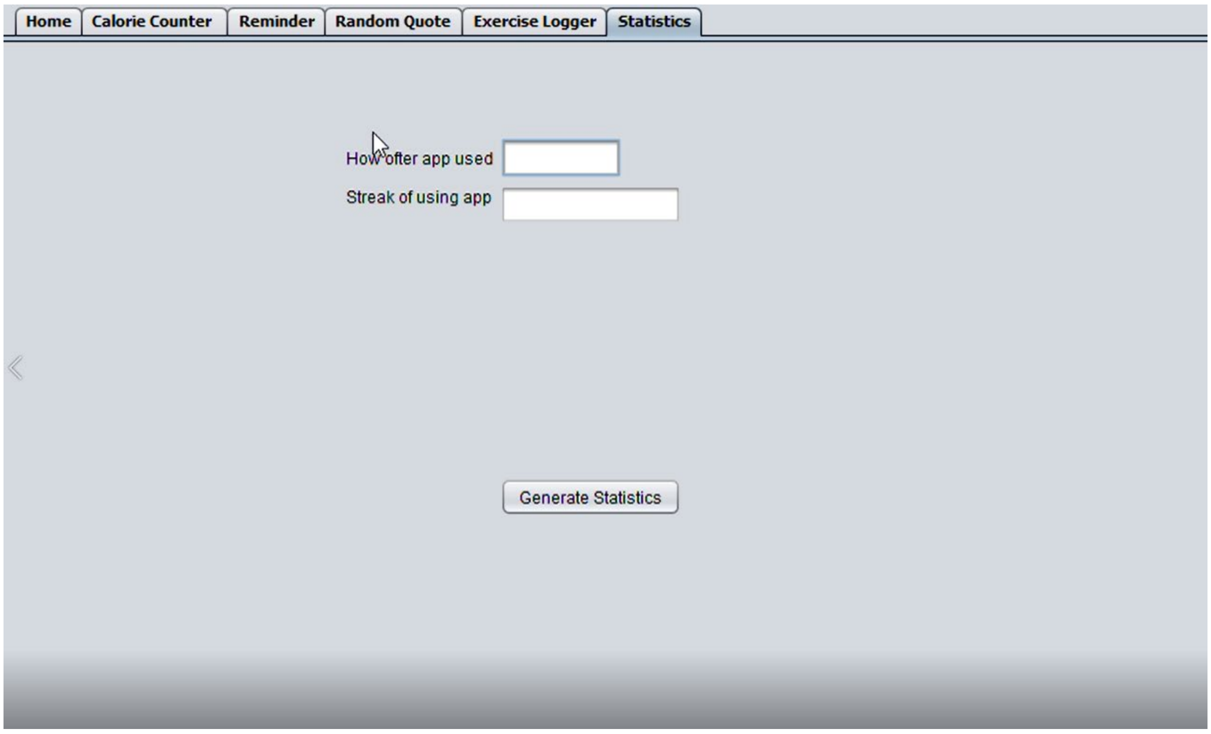
Above is the screen for a reminder prototype. It has a time for a reminder and a reminder text.



The random quote prototype has two buttons one to generate the code and one to save the code.



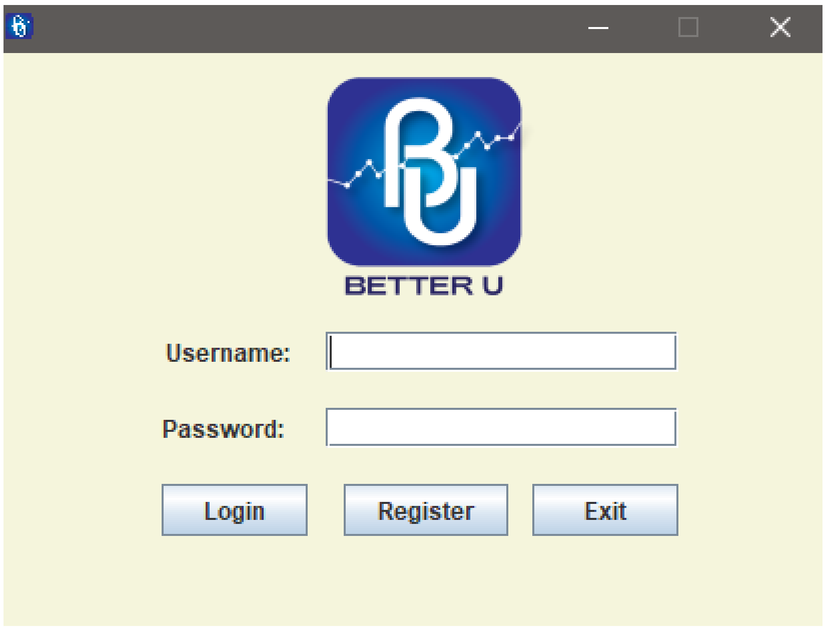
Here for the exercise logger prototype, we can see it is trying to mimic a crud application.



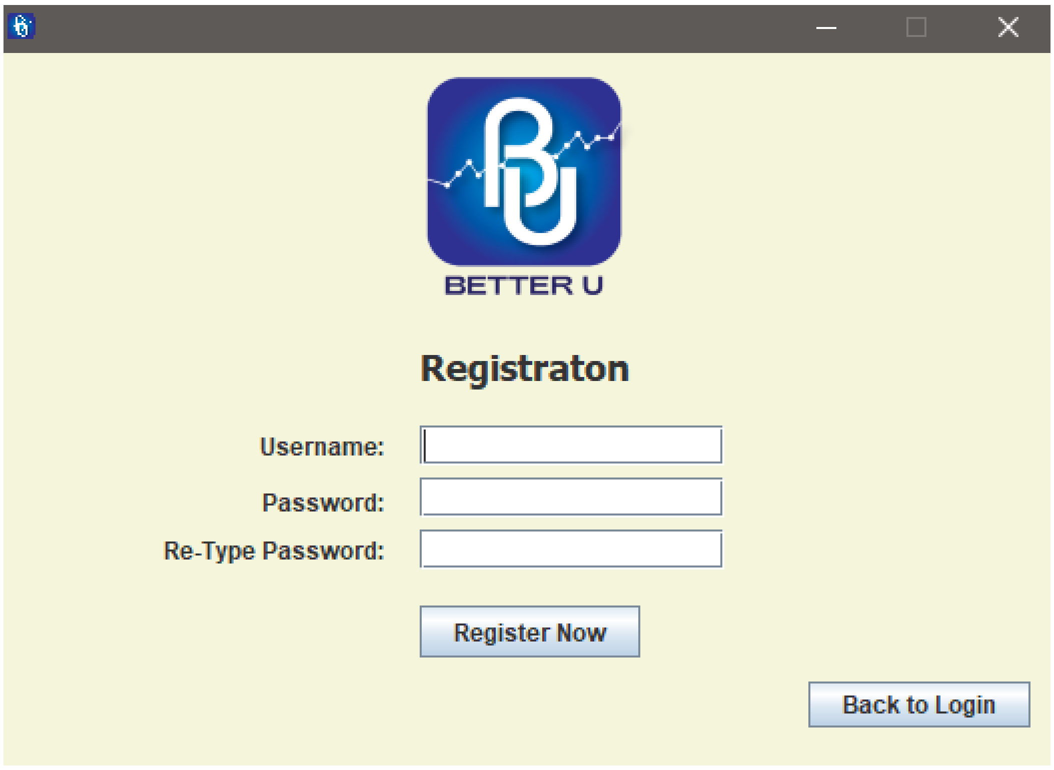
Here in the statistics tab the prototype has a button for generating statistics and some fields that are to be used for the statistics.

After going through my prototype its strange to see the early-stage ideas of my program and now after fully developing it what it turned out like. Below I will put the screenshots of my fully developed application.

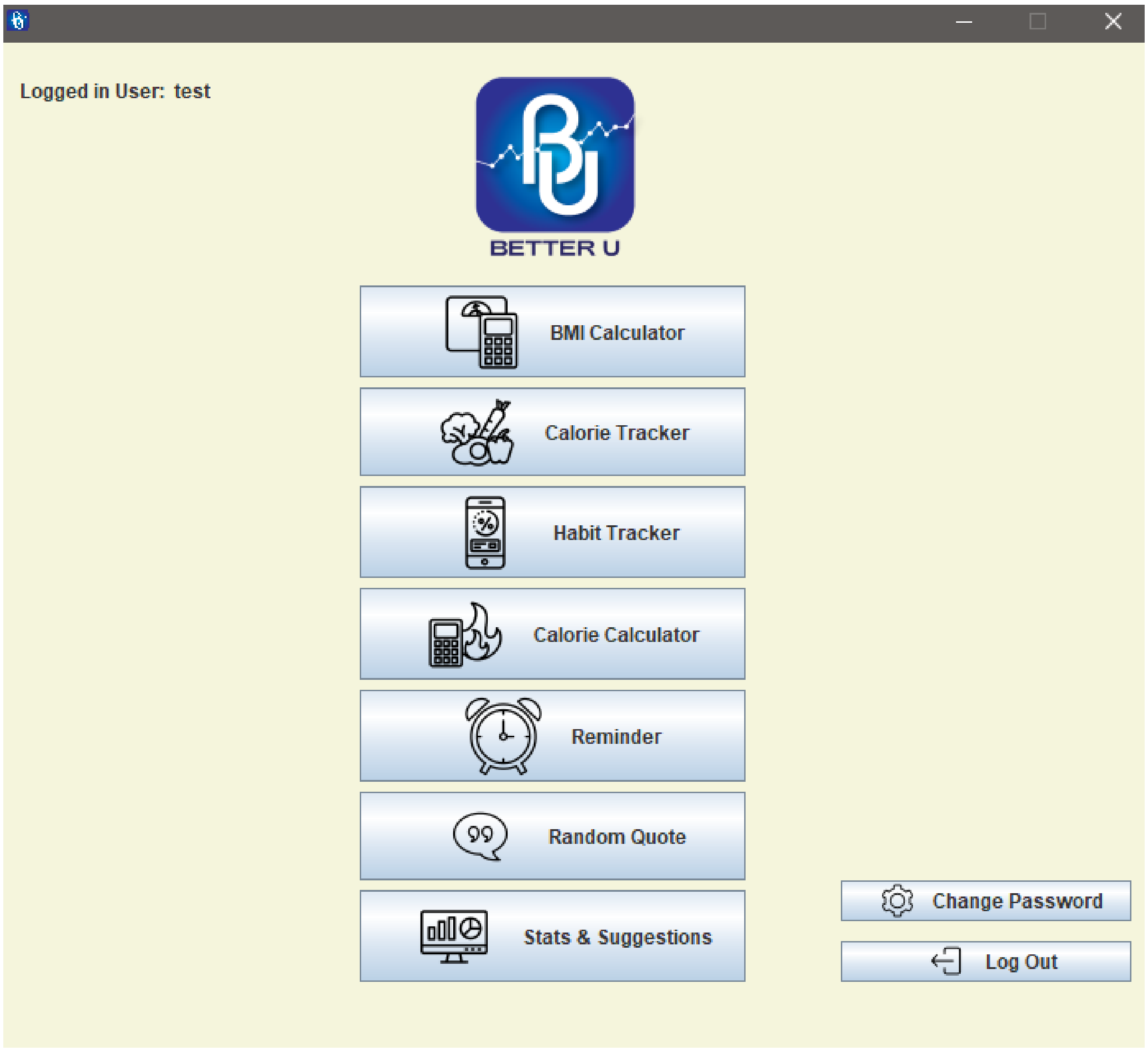
Login



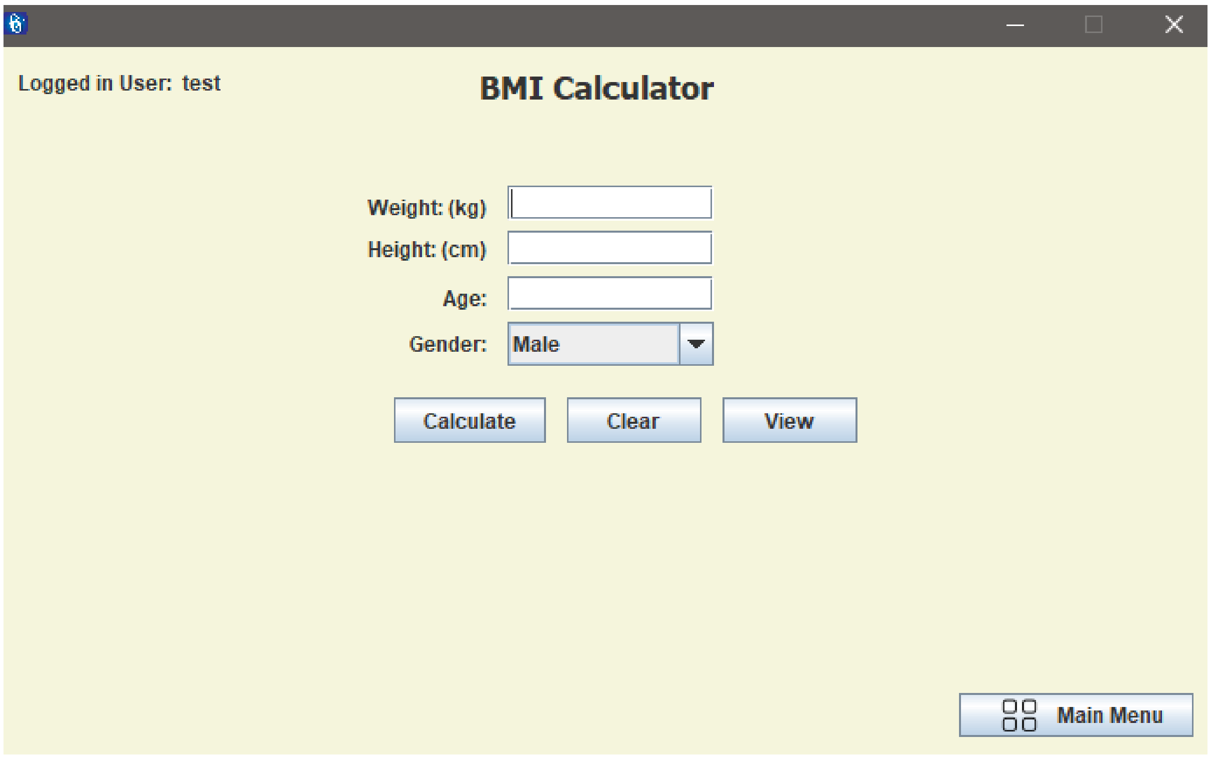
Register



Main Menu



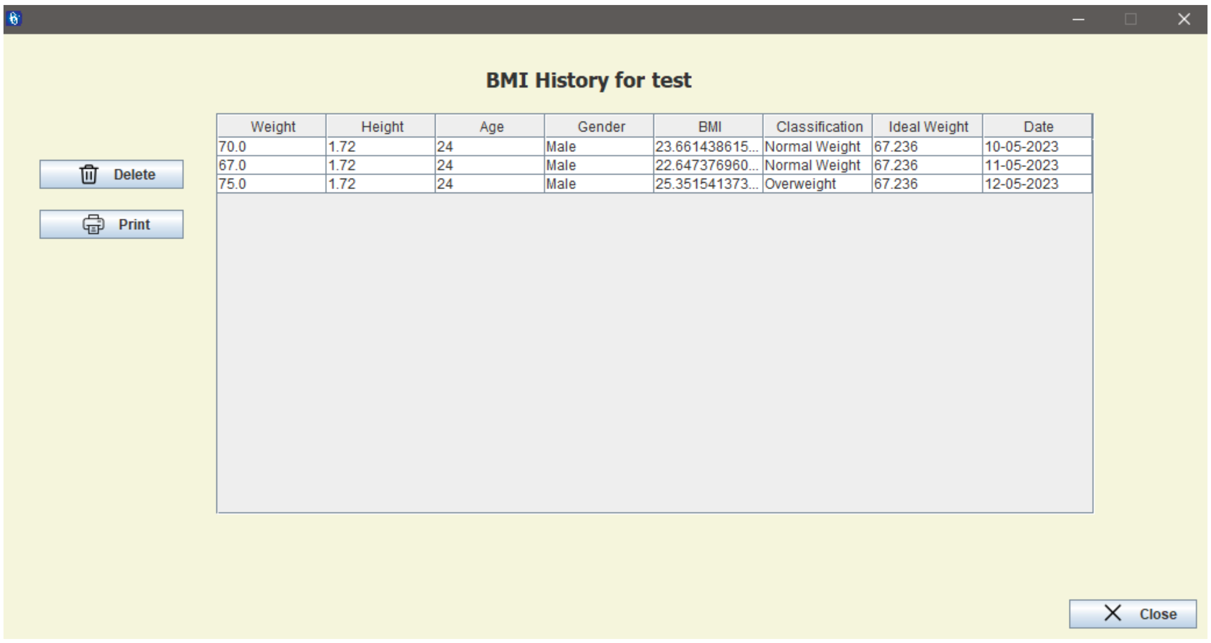
BMI Calculator


Logged in User: test

BMI Calculator

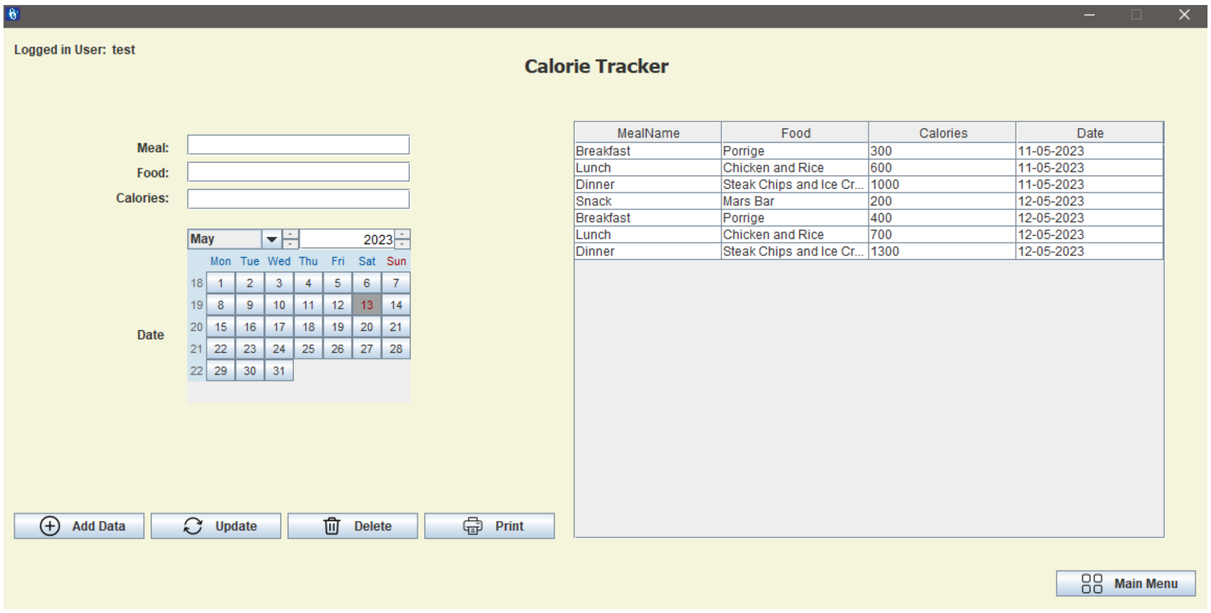
Weight: (kg)
Height: (cm)
Age:
Gender: Male

View BMI Calculator


BMI History for test

Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date
70.0	1.72	24	Male	23.661438615...	Normal Weight	67.236	10-05-2023
67.0	1.72	24	Male	22.647376960...	Normal Weight	67.236	11-05-2023
75.0	1.72	24	Male	25.351541373...	Overweight	67.236	12-05-2023

Calorie Calculator



Logged in User: test

Calorie Tracker

Meal:

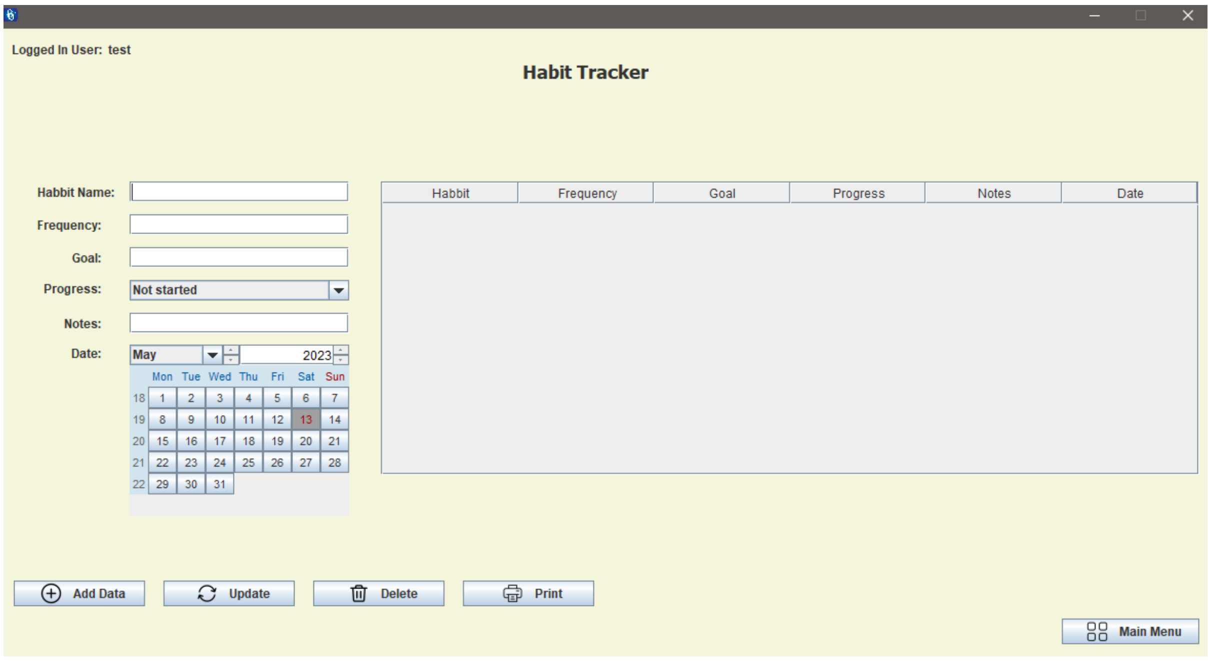
Food:

Calories:

Date

MealName	Food	Calories	Date
Breakfast	Porrige	300	11-05-2023
Lunch	Chicken and Rice	600	11-05-2023
Dinner	Steak Chips and Ice Cr...	1000	11-05-2023
Snack	Mars Bar	200	12-05-2023
Breakfast	Porrige	400	12-05-2023
Lunch	Chicken and Rice	700	12-05-2023
Dinner	Steak Chips and Ice Cr...	1300	12-05-2023

Habit Tracker



Logged In User: test

Habit Tracker

Habbit Name:

Frequency:

Goal:

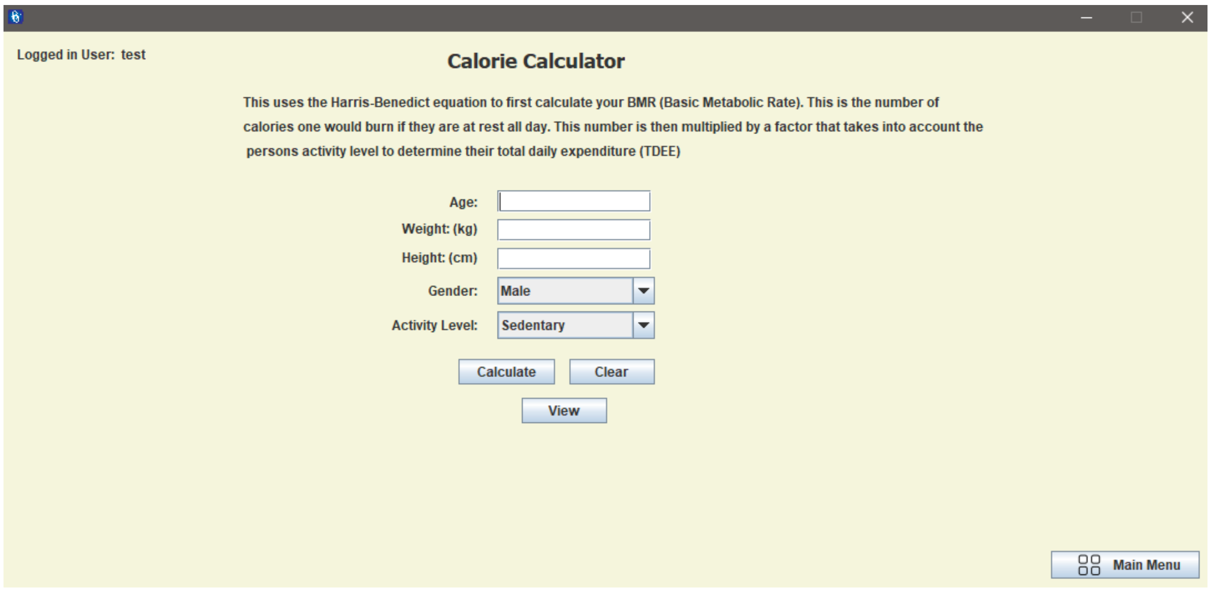
Progress: Not started

Notes:

Date:

Habbit	Frequency	Goal	Progress	Notes	Date

Calorie Calculator



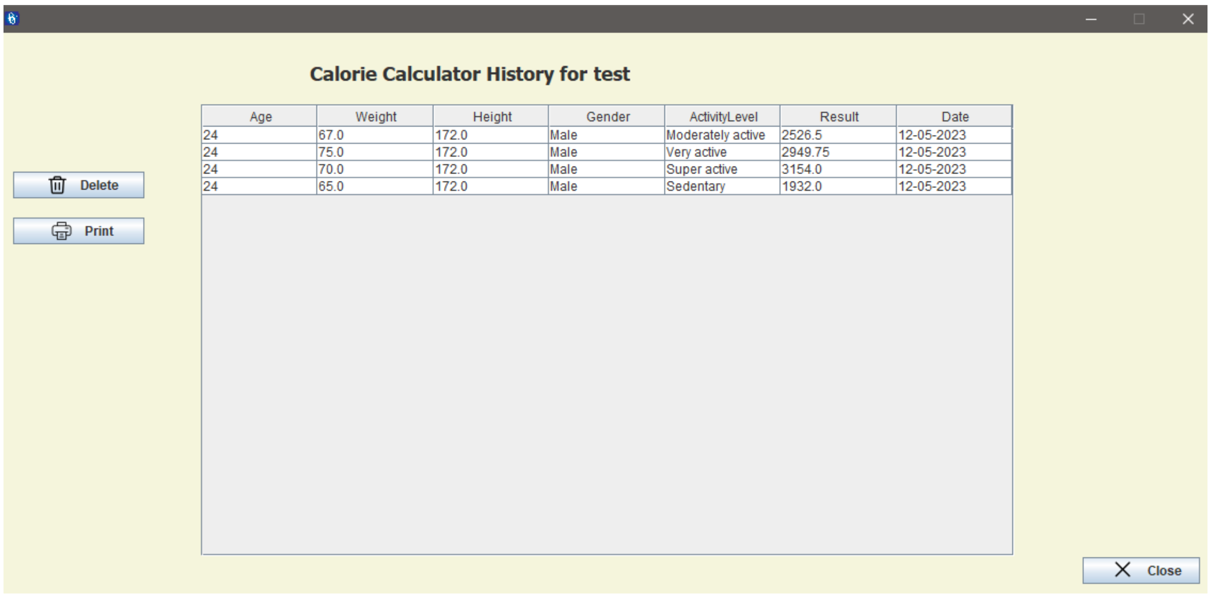
Logged in User: test

Calorie Calculator

This uses the Harris-Benedict equation to first calculate your BMR (Basic Metabolic Rate). This is the number of calories one would burn if they are at rest all day. This number is then multiplied by a factor that takes into account the persons activity level to determine their total daily expenditure (TDEE)

Age:	
Weight: (kg)	
Height: (cm)	
Gender:	Male
Activity Level:	Sedentary

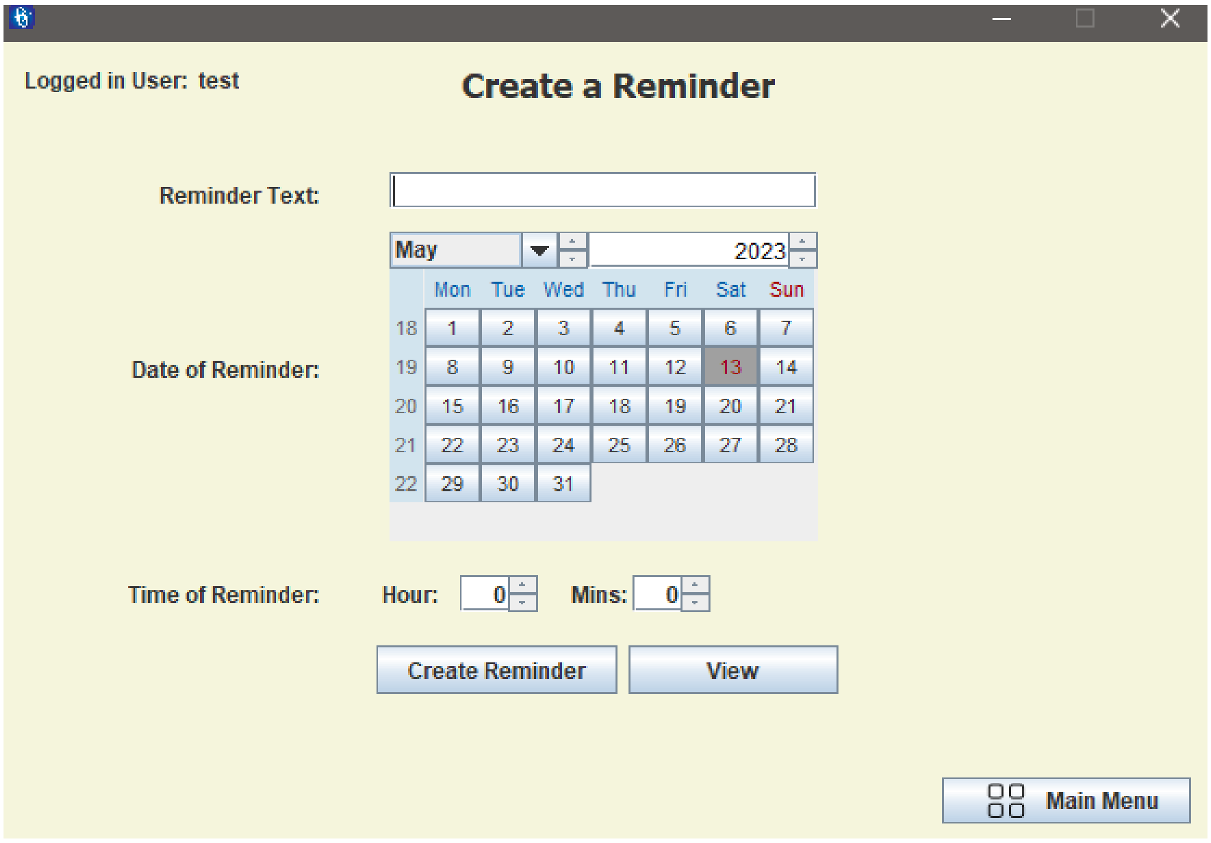
View Calorie Calculator



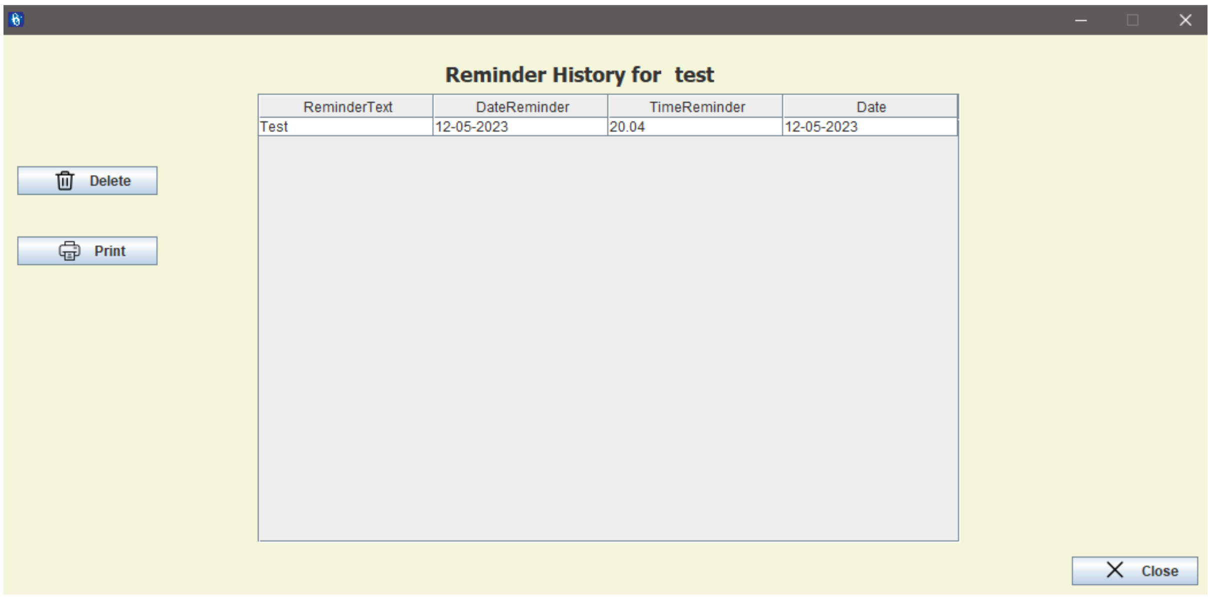
Calorie Calculator History for test

Age	Weight	Height	Gender	ActivityLevel	Result	Date
24	67.0	172.0	Male	Moderately active	2526.5	12-05-2023
24	75.0	172.0	Male	Very active	2949.75	12-05-2023
24	70.0	172.0	Male	Super active	3154.0	12-05-2023
24	65.0	172.0	Male	Sedentary	1932.0	12-05-2023

Reminder

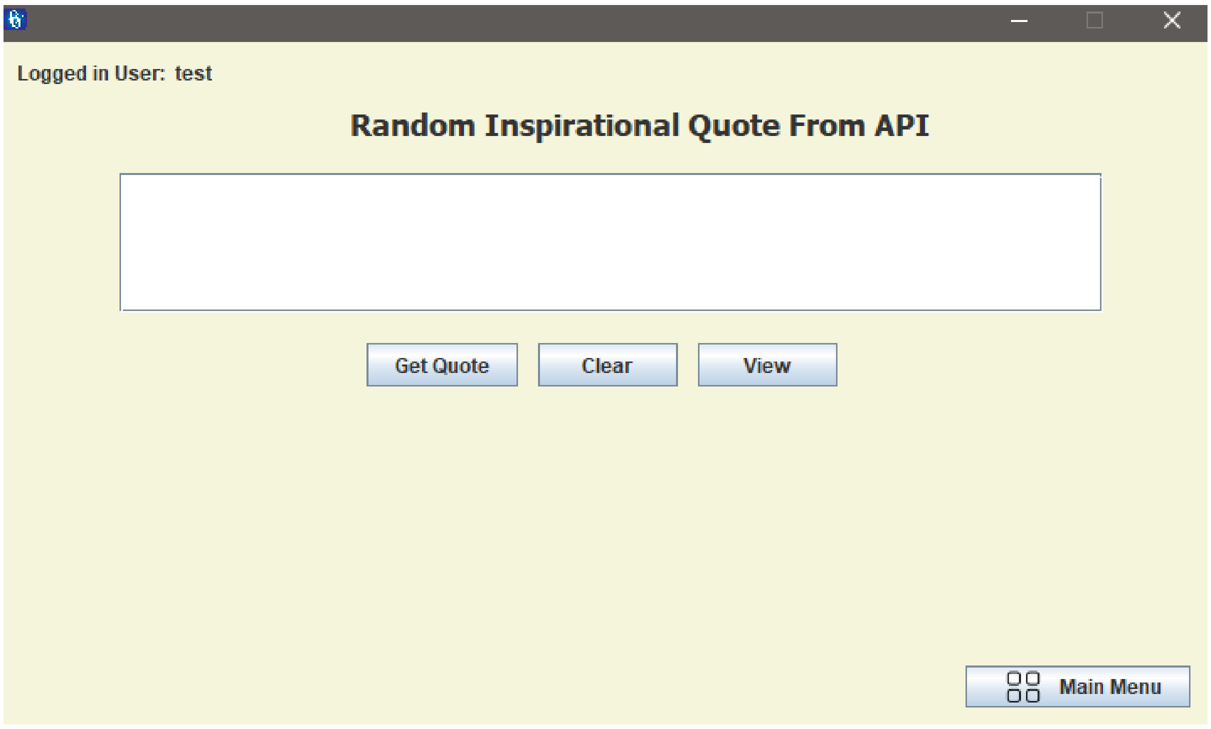


View Reminder

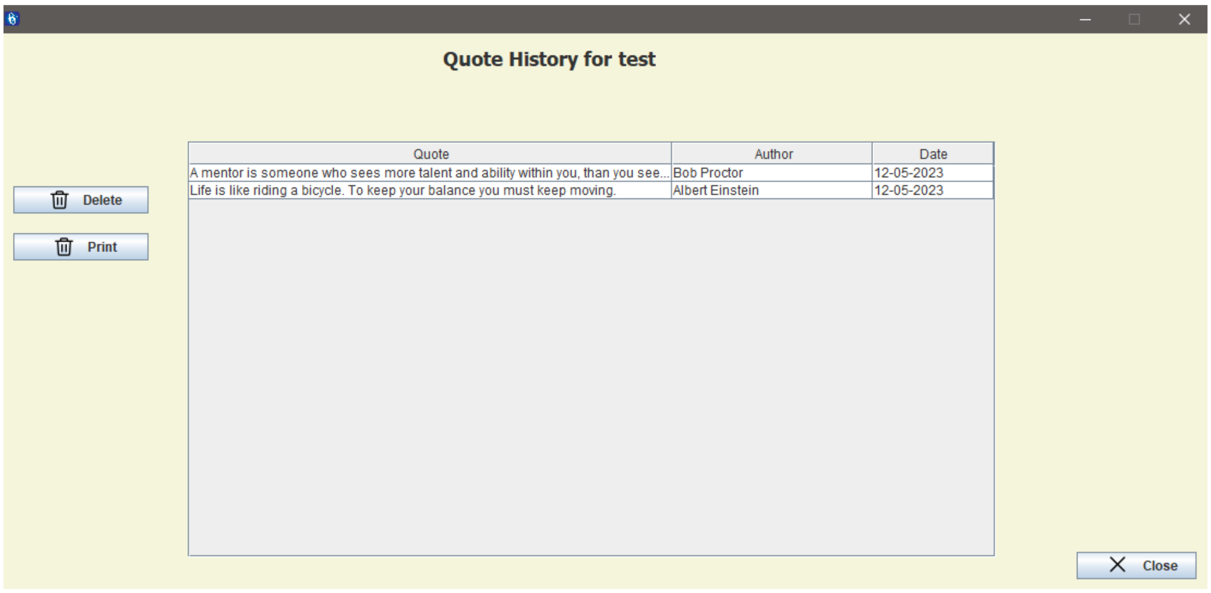


ReminderText	DateReminder	TimeReminder	Date
Test	12-05-2023	20.04	12-05-2023

Random Quote

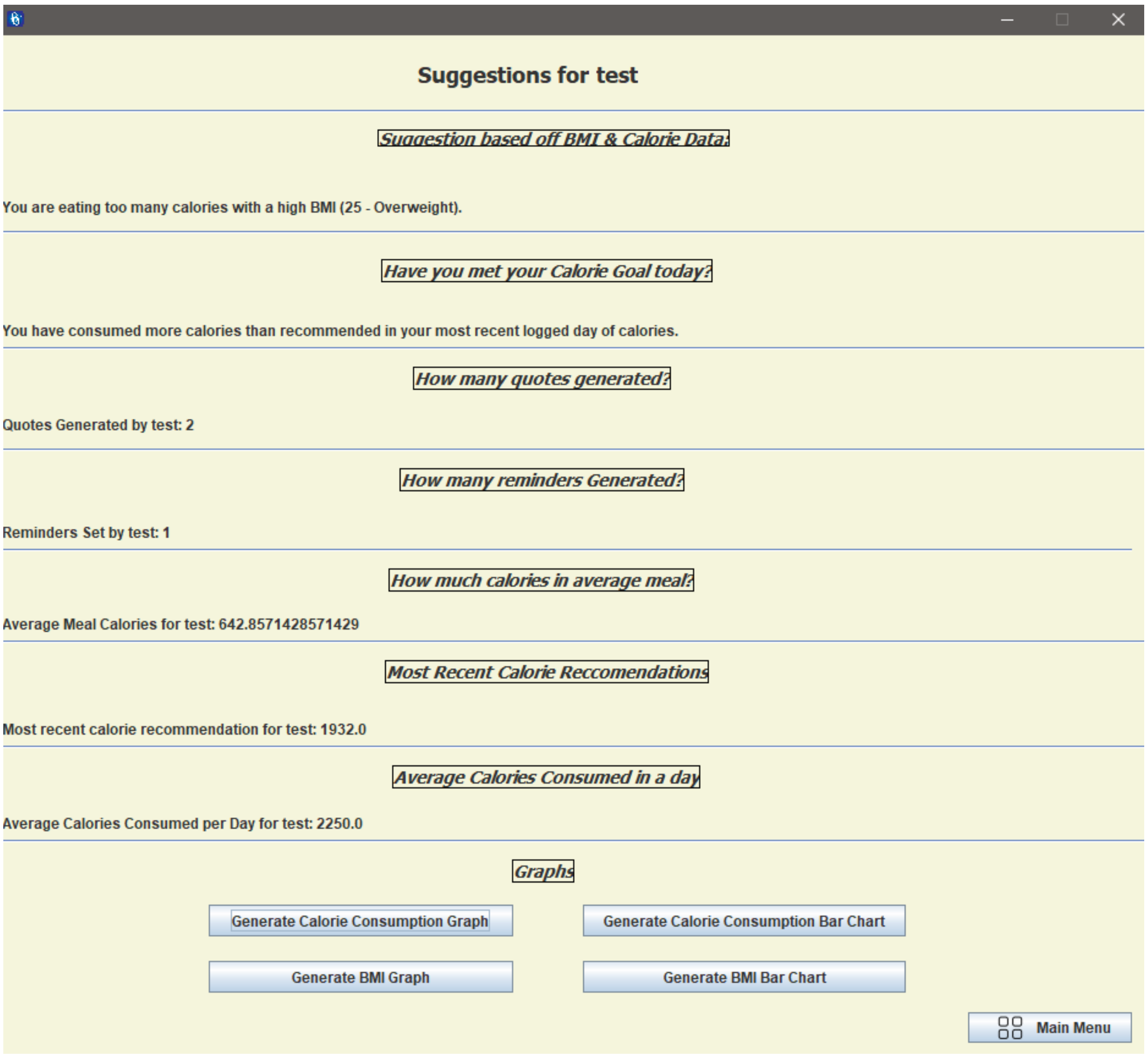


View Random Quote



Quote	Author	Date
A mentor is someone who sees more talent and ability within you, than you see...	Bob Proctor	12-05-2023
Life is like riding a bicycle. To keep your balance you must keep moving.	Albert Einstein	12-05-2023

Stats and Statistics



Suggestions for test

Suggestion based off BMI & Calorie Data:

You are eating too many calories with a high BMI (25 - Overweight).

Have you met your Calorie Goal today?

You have consumed more calories than recommended in your most recent logged day of calories.

How many quotes generated?

Quotes Generated by test: 2

How many reminders Generated?

Reminders Set by test: 1

How much calories in average meal?

Average Meal Calories for test: 642.8571428571429

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test: 1932.0

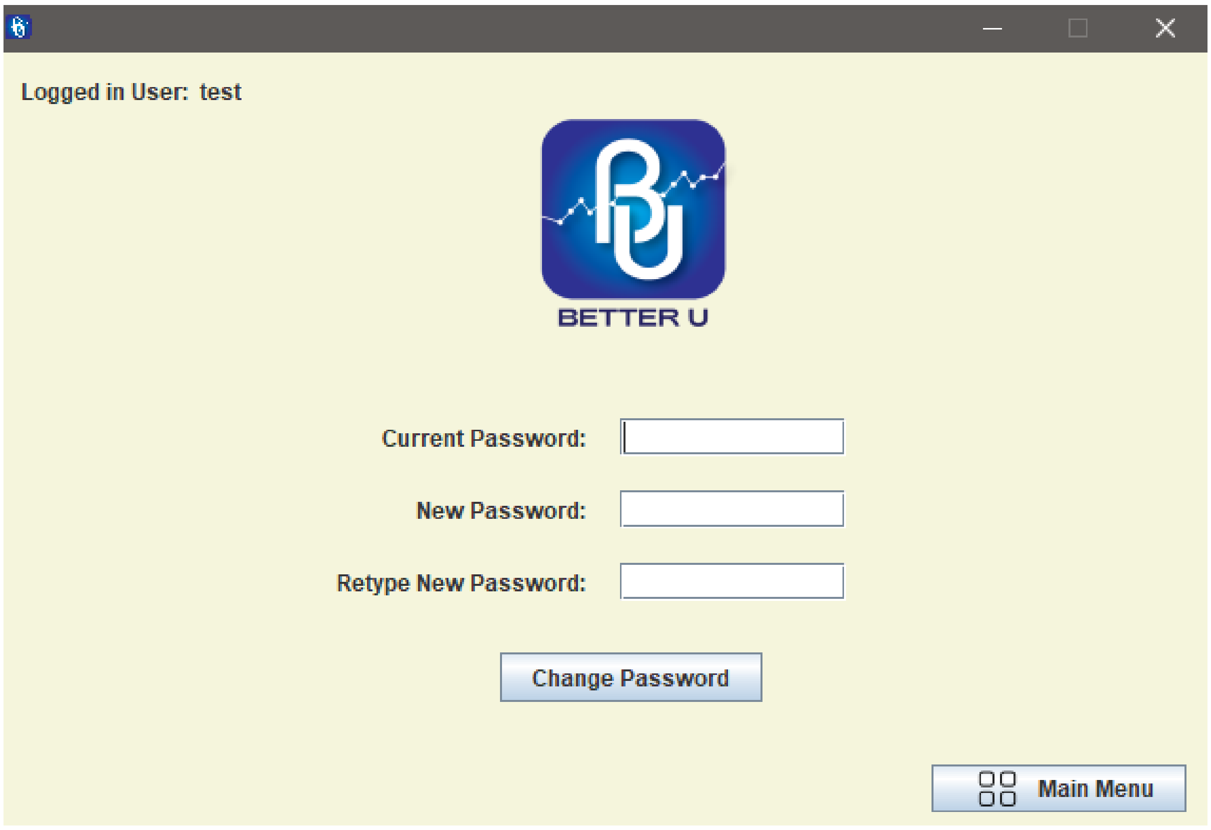
Average Calories Consumed in a day

Average Calories Consumed per Day for test: 2250.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

Change Password



2.6. Testing

As I had done quite a bit of testing in the implementation section of this report, I will just test the main functions of this application. The navigational buttons and supporting buttons work as seen in the implementation section.

For my testing I will be using Unit tests on the different features of my program. I chose to do unit testing as I feel it is the method of testing that best fitted my needs. Unit testing can provide great insights into how a system functions and preforms. For this the system is gone through from start to finish and all the functions tested.

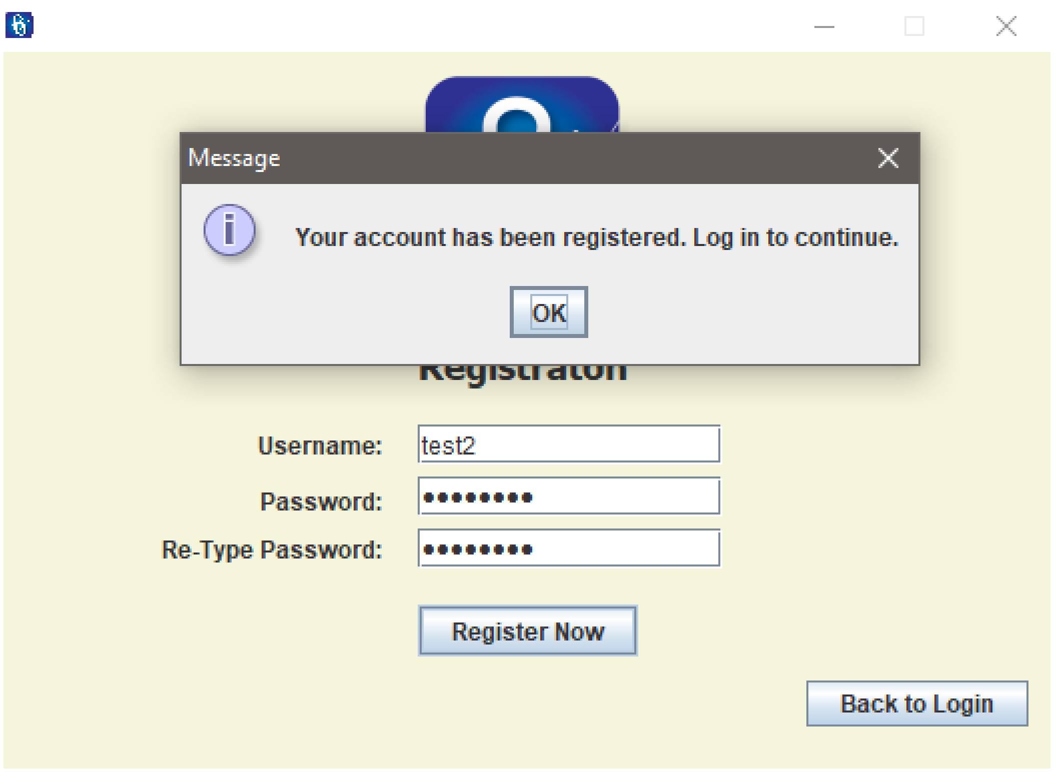
****

Name of Test Case: Registration

Description of Test Case: User goes to register in the BetterU application.

Expected Result of Test Case: A user creates an account.

Actual Result of Test Case: Account Created


Message

Your account has been registered. Log in to continue.

Error handling can be seen in the implementation for the above function.

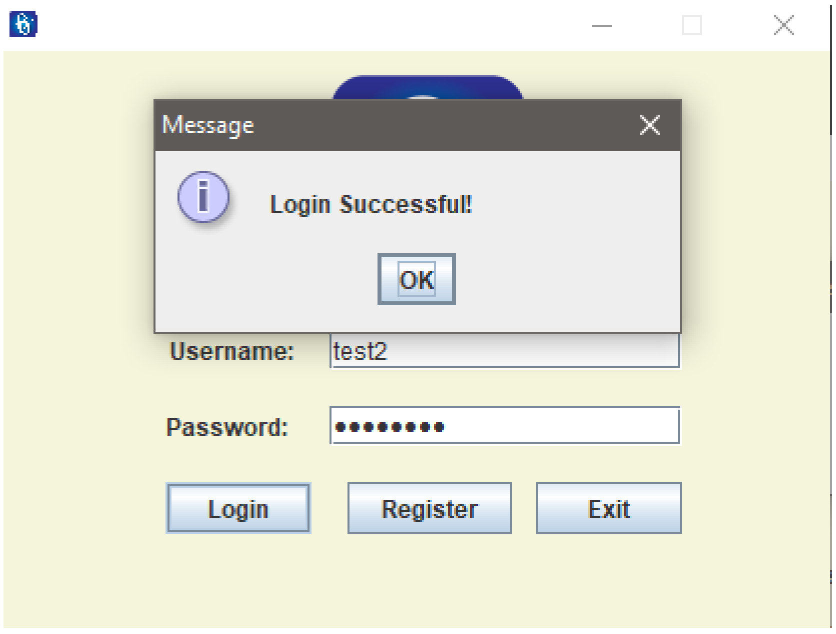
****

Name of Test Case: Login

Description of Test Case: User goes to login in the BetterU application.

Expected Result of Test Case: A user logs in

Actual Result of Test Case: User Logged in.


Message

Login Successful!

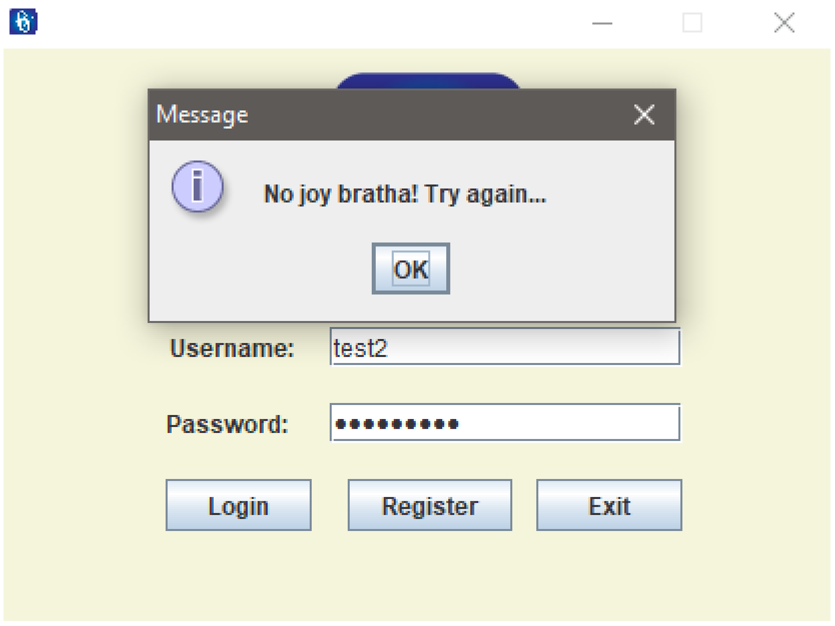
****

***Name of Test Case**: User tries to log in with wrong credentials.*

Description of Test Case: User enters incorrect information into the login.

Expected Result of Test Case: User gets told wrong username or password.

Actual Result of Test Case: User gets error message.



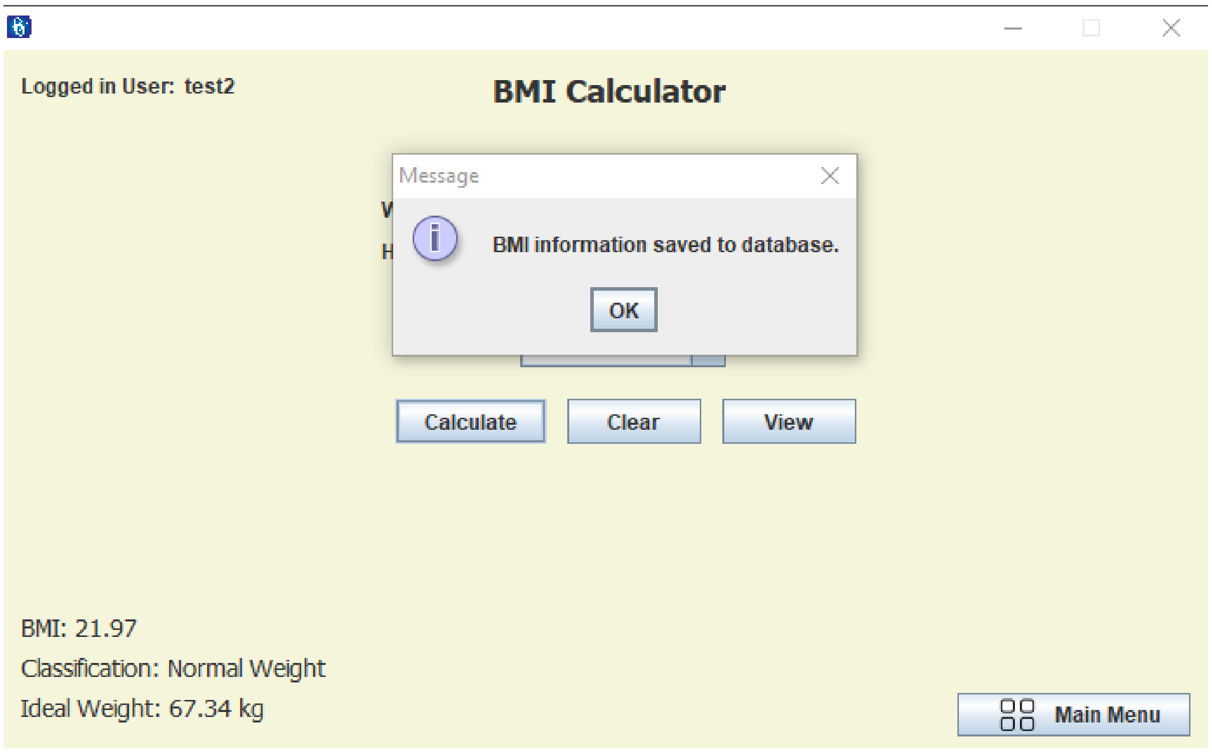
****

***Name of Test Case**: User calculates BMI.*

Description of Test Case: User calculates their BMI using the BMI calculator.

Expected Result of Test Case: User gets to see their BMI.

Actual Result of Test Case: User sees BMI.


Logged in User: test2

BMI Calculator

Message

BMI information saved to database.

BMI: 21.97

Classification: Normal Weight

Ideal Weight: 67.34 kg

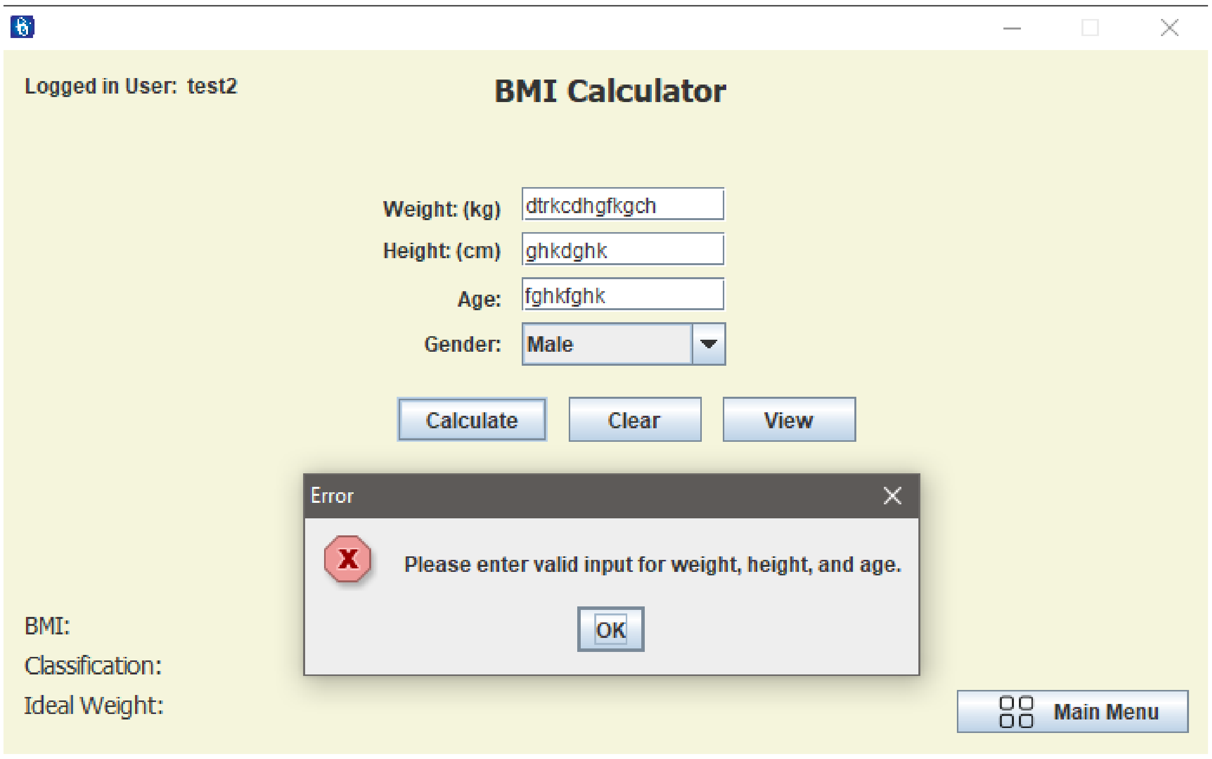
****

Name of Test Case: User enters nonsense into BMI calculator.

Description of Test Case: Nonsense input added.

Expected Result of Test Case: User gets warning message.

Actual Result of Test Case: User gets error.


Logged in User: test2

BMI Calculator

Weight: (kg) dtrkcdhgfkgch

Height: (cm) ghkdghk

Age: fghkfghk

Gender: Male

Error

Please enter valid input for weight, height, and age.

BMI:

Classification:

Ideal Weight:

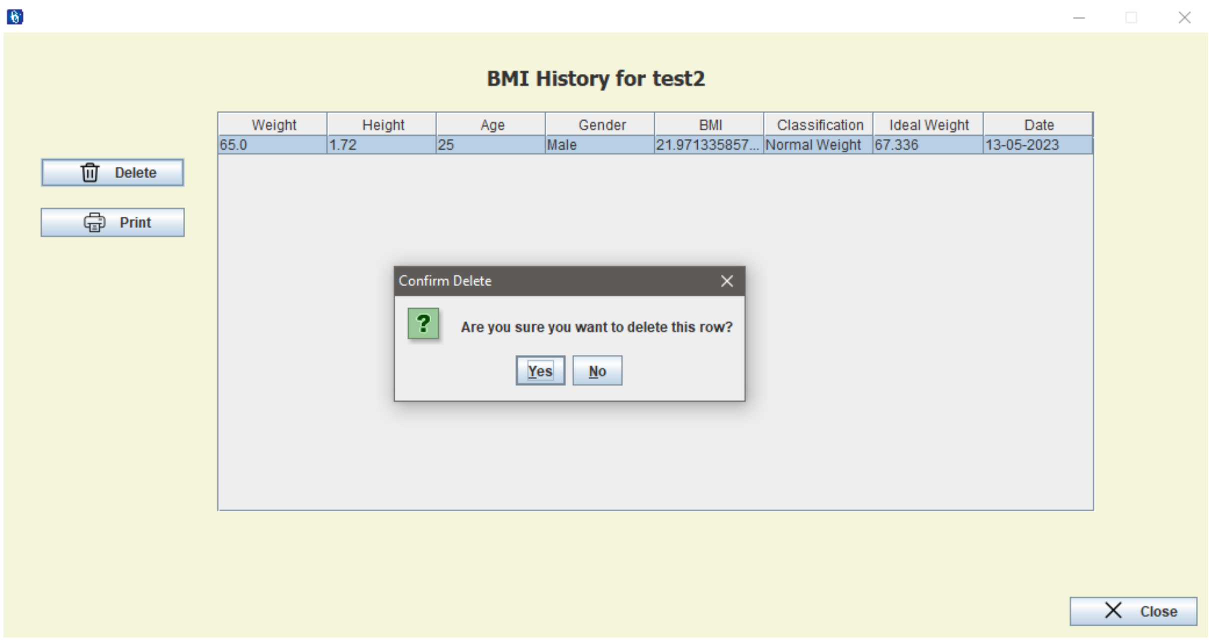
****

Name of Test Case: User deletes entry from view BMI calculator.

Description of Test Case: User deletes past BMI calculation.

Expected Result of Test Case: Database entry deleted.

Actual Result of Test Case: Database entry deleted.

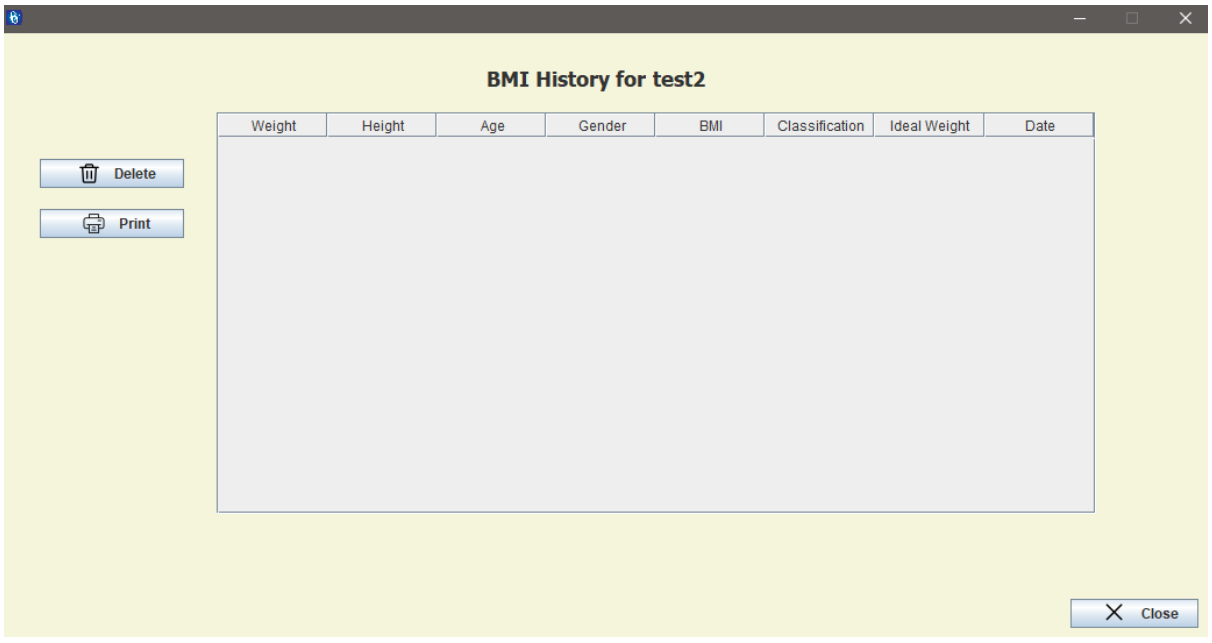


Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date
65.0	1.72	25	Male	21.971335857...	Normal Weight	67.336	13-05-2023

Confirm Delete

Are you sure you want to delete this row?

Yes No



Weight	Height	Age	Gender	BMI	Classification	Ideal Weight	Date

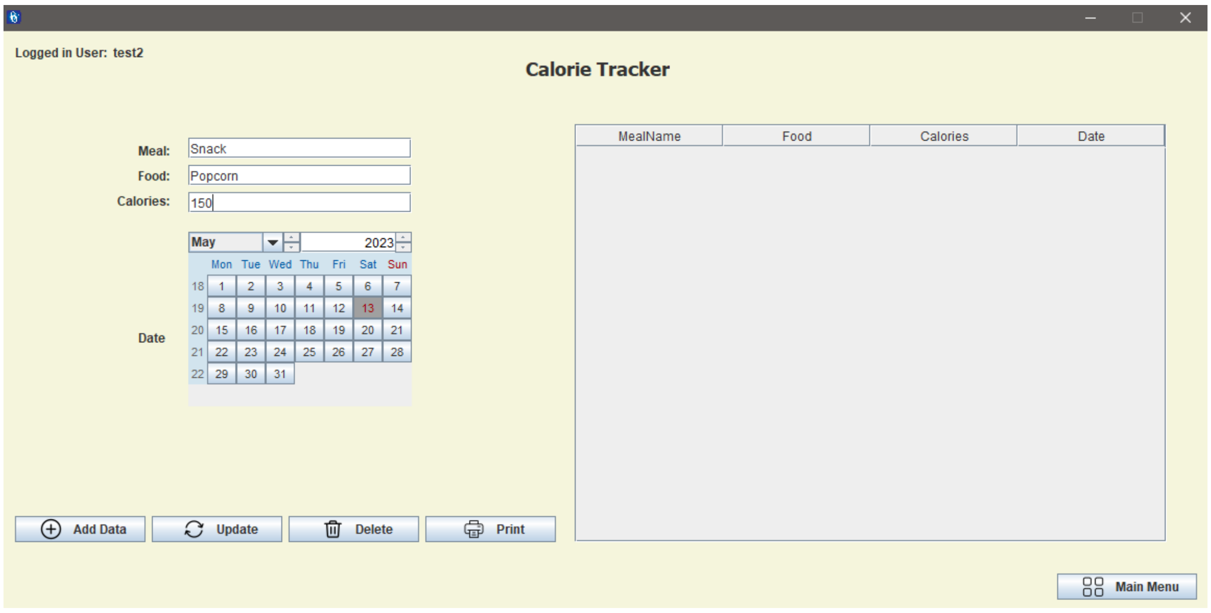
****

Name of Test Case: User enters entry into Calorie Tracker.

Description of Test Case: User enters entry into Calorie Tracker.

Expected Result of Test Case: Entry added.

Actual Result of Test Case: Entry added.


Logged in User: test2

Calorie Tracker

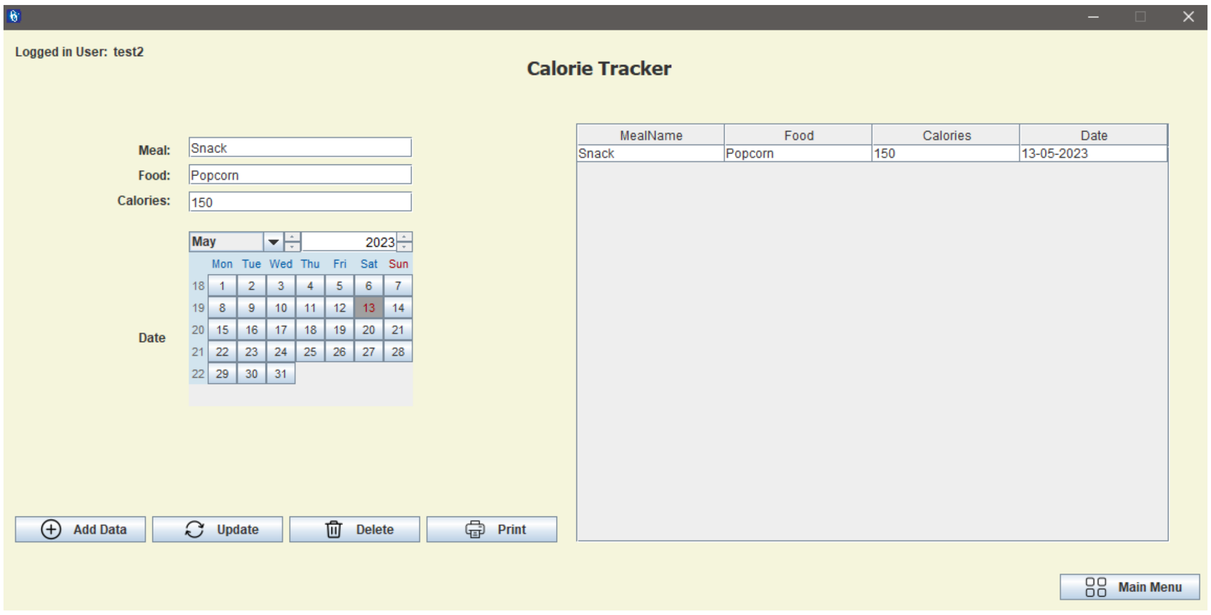
Meal: Snack

Food: Popcorn

Calories: 150

Date: May 2023, 13 selected

MealName	Food	Calories	Date


Logged in User: test2

Calorie Tracker

Meal: Snack

Food: Popcorn

Calories: 150

Date: May 2023, 13 selected

MealName	Food	Calories	Date
Snack	Popcorn	150	13-05-2023

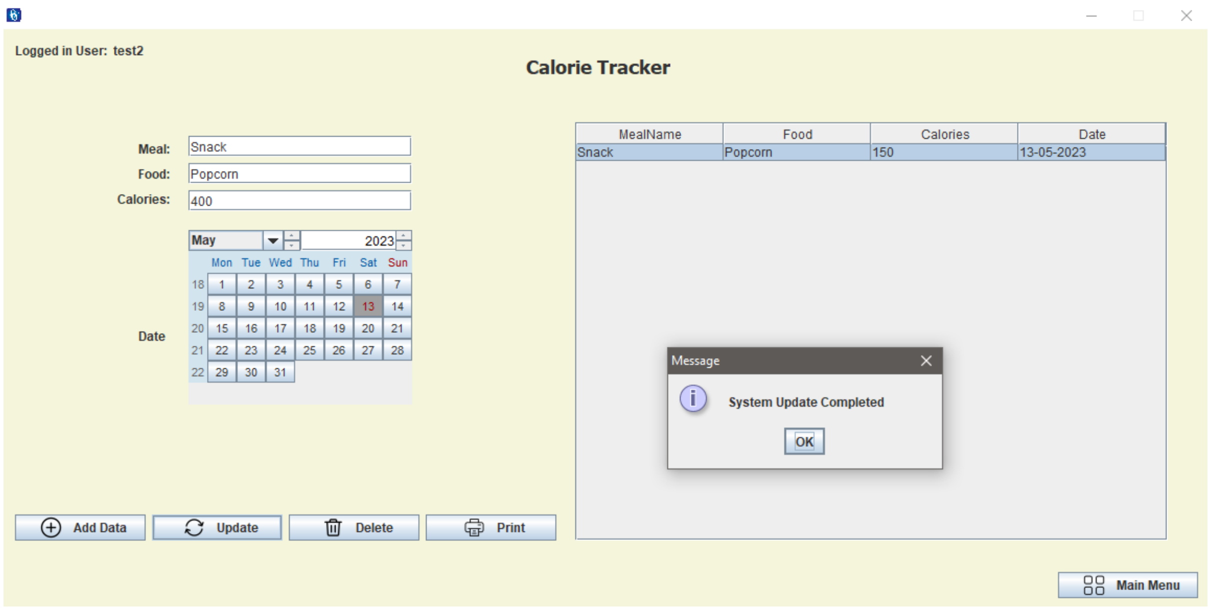
****

Name of Test Case: User updates entry in Calorie Tracker.

Description of Test Case: User updates entry in Calorie Tracker.

Expected Result of Test Case: Entry updated.

Actual Result of Test Case: Entry updated.


Logged in User: test2

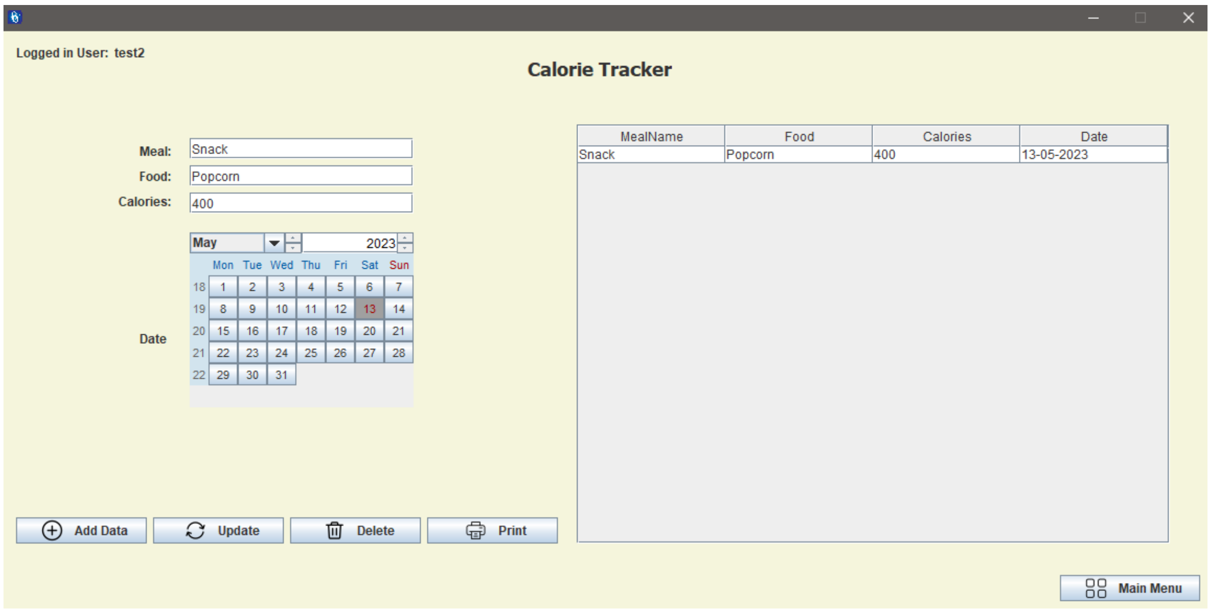
Calorie Tracker

Meal: Snack
Food: Popcorn
Calories: 400
Date: May 2023

MealName	Food	Calories	Date
Snack	Popcorn	150	13-05-2023

Message

System Update Completed


Logged in User: test2

Calorie Tracker

Meal: Snack
Food: Popcorn
Calories: 400
Date: May 2023

MealName	Food	Calories	Date
Snack	Popcorn	400	13-05-2023

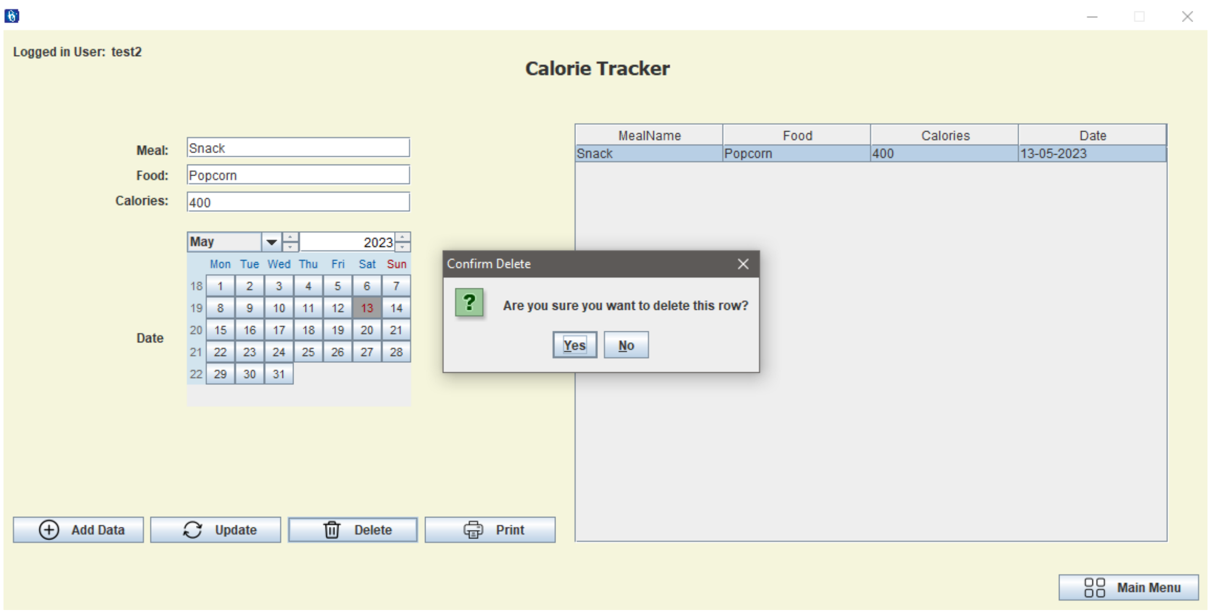
****

Name of Test Case: User Deletes Entry in Calorie Tracker.

Description of Test Case: User Deletes Entry in Calorie Tracker.

Expected Result of Test Case: Entry deleted.

Actual Result of Test Case: Entry deleted.


Logged in User: test2

Calorie Tracker

Meal: Snack

Food: Popcorn

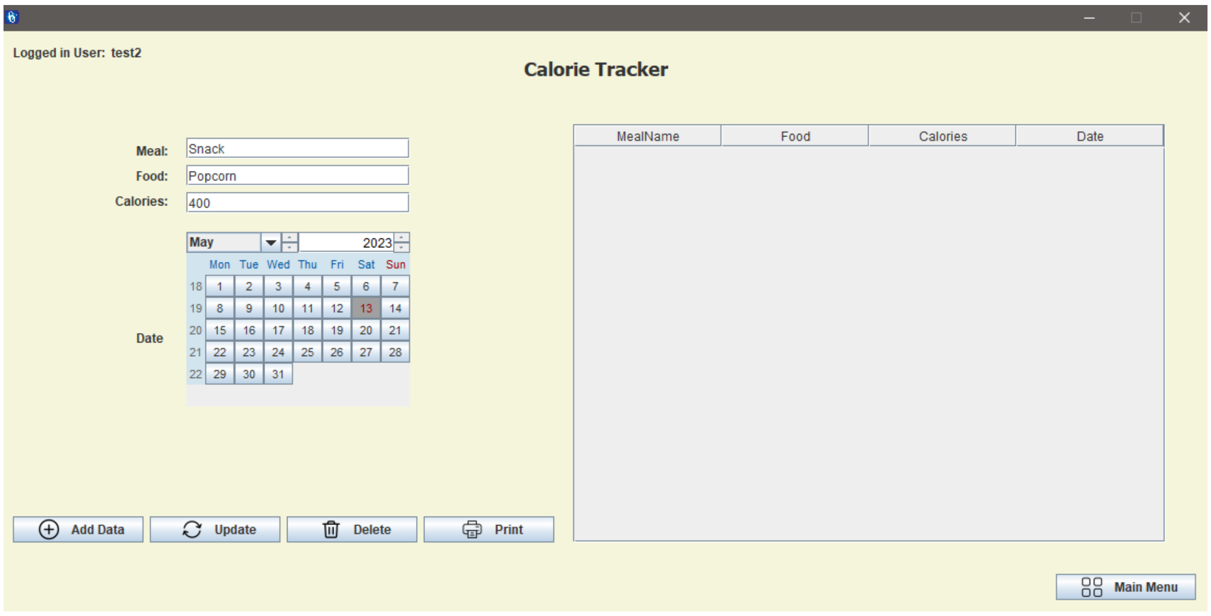
Calories: 400

Date

MealName	Food	Calories	Date
Snack	Popcorn	400	13-05-2023

Confirm Delete

Are you sure you want to delete this row?


Logged in User: test2

Calorie Tracker

Meal: Snack

Food: Popcorn

Calories: 400

Date

MealName	Food	Calories	Date

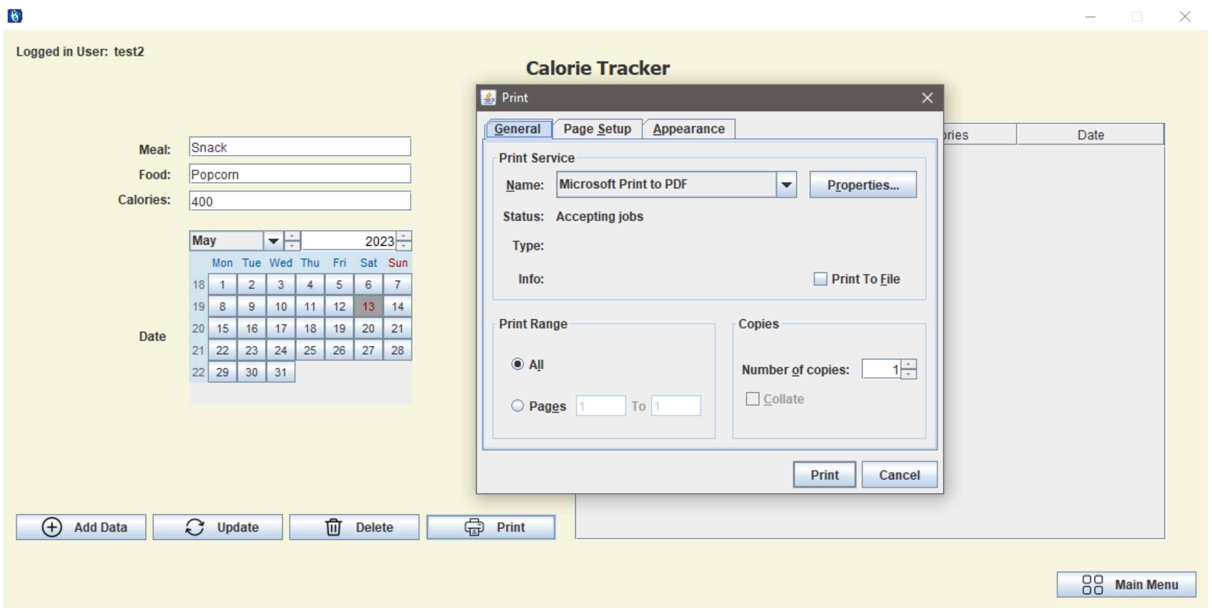
****

Name of Test Case: User presses print in Calorie Tracker.

Description of Test Case: Print button pressed.

Expected Result of Test Case: Print dialog opens.

Actual Result of Test Case: Print dialog opens.


Logged in User: test2

Calorie Tracker

Meal: Snack

Food: Popcorn

Calories: 400

Date: May 2023, 13 selected

Print

Print Service

Name: Microsoft Print to PDF

Status: Accepting jobs

Print Range: All

Number of copies: 1

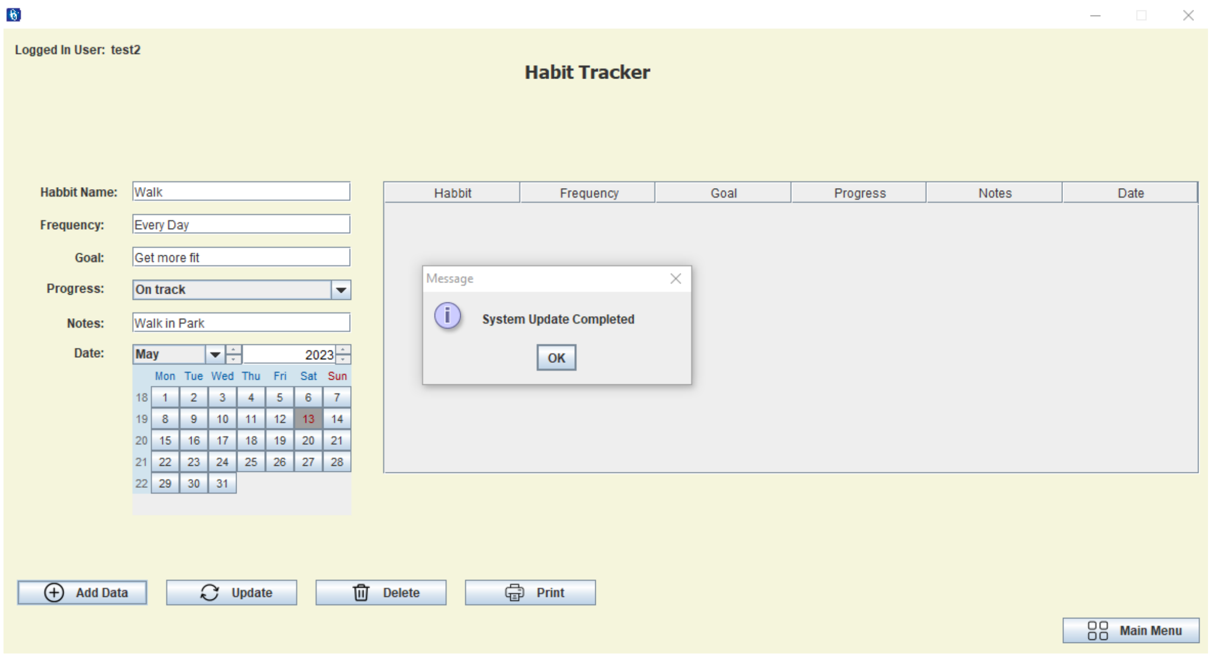
****

Name of Test Case*:* User enters entry into Habit Tracker.

Description of Test Case: User enters entry into Habit Tracker.

Expected Result of Test Case: Entry added.

Actual Result of Test Case: Entry added.


Logged In User: test2

Habit Tracker

Habbit Name: Walk

Frequency: Every Day

Goal: Get more fit

Progress: On track

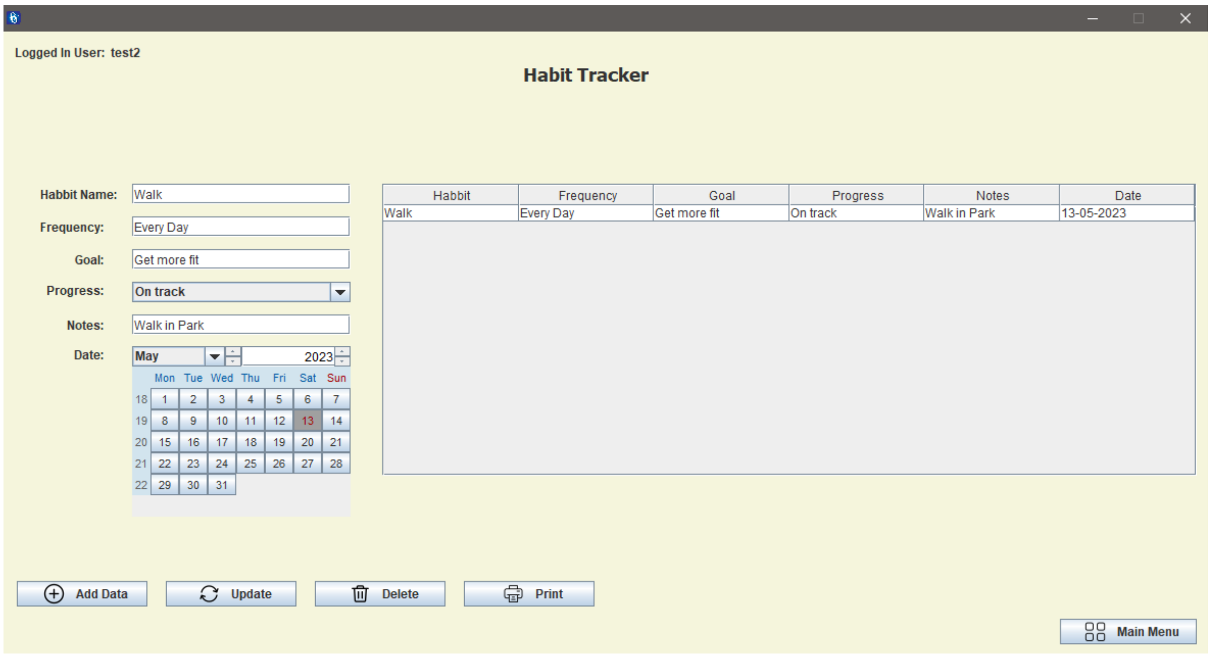
Notes: Walk in Park

Date: May 2023, 13 selected

Habbit	Frequency	Goal	Progress	Notes	Date

Message

System Update Completed


Logged In User: test2

Habit Tracker

Habbit Name:	Walk
Frequency:	Every Day
Goal:	Get more fit
Progress:	On track
Notes:	Walk in Park
Date:	May 2023 (13)

Habbit	Frequency	Goal	Progress	Notes	Date
Walk	Every Day	Get more fit	On track	Walk in Park	13-05-2023

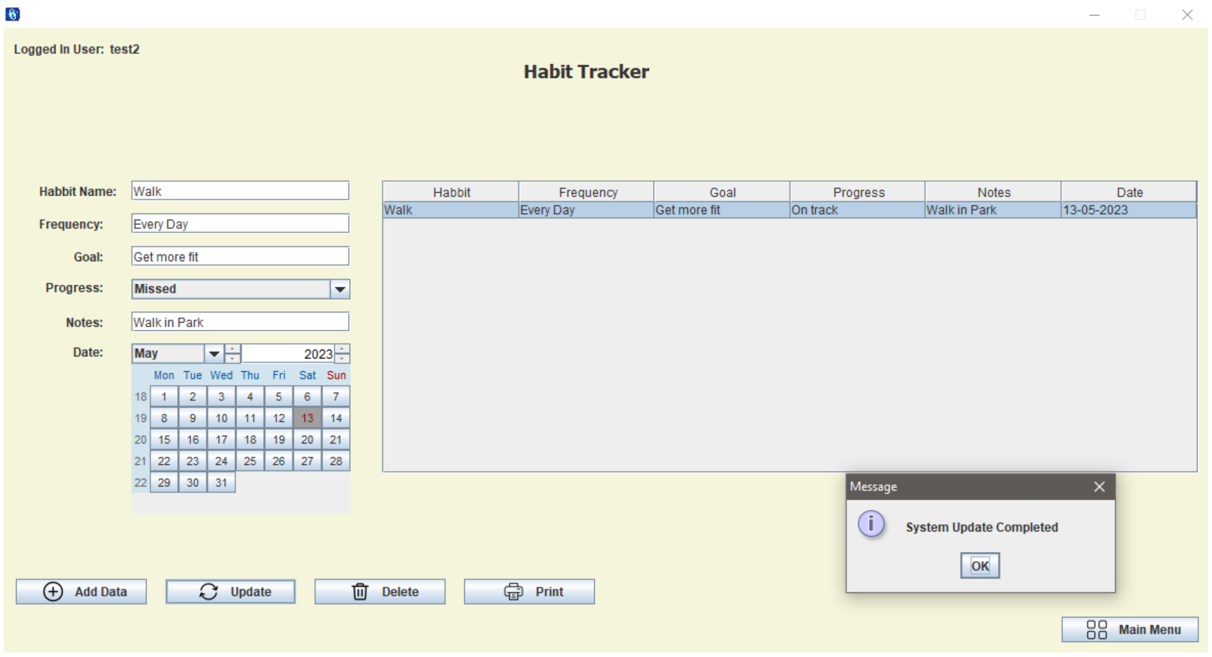
****

Name of Test Case: User updates entry in Habit Tracker.

Description of Test Case: User updates entry in Habit Tracker.

Expected Result of Test Case: Entry updated.

Actual Result of Test Case: Entry updated.


Logged In User: test2

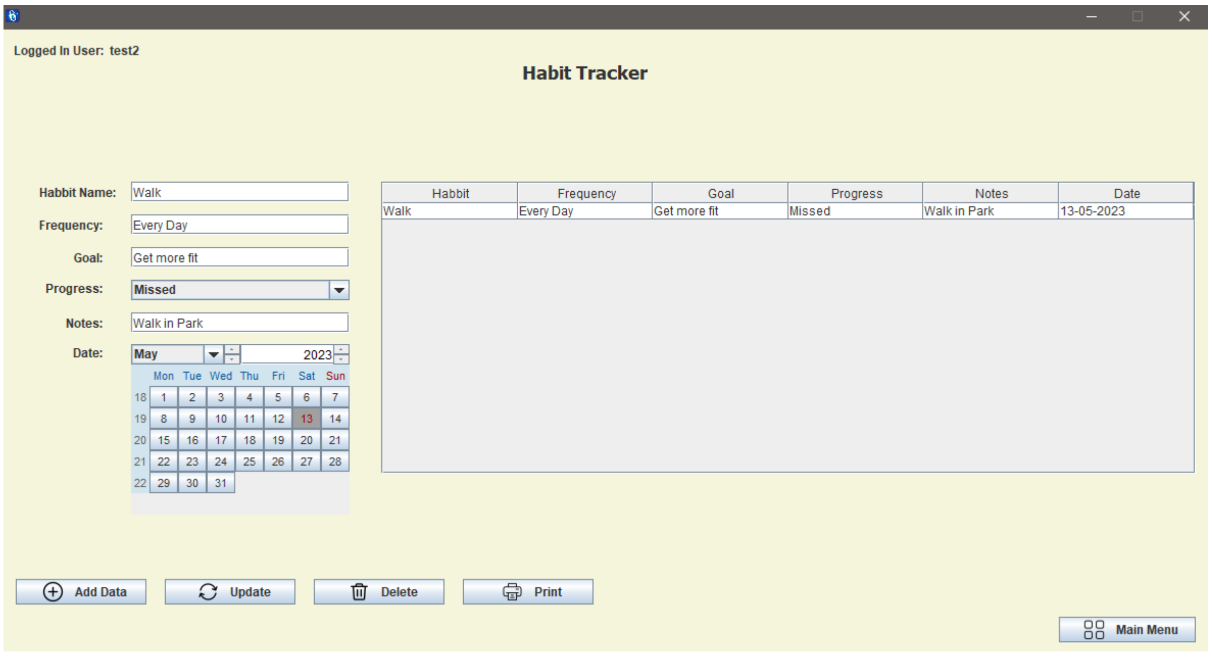
Habit Tracker

Habbit Name:	Walk
Frequency:	Every Day
Goal:	Get more fit
Progress:	Missed
Notes:	Walk in Park
Date:	May 2023 (13)

Habbit	Frequency	Goal	Progress	Notes	Date
Walk	Every Day	Get more fit	On track	Walk in Park	13-05-2023

Message

System Update Completed


Logged In User: test2

Habit Tracker

Habbit Name: Walk

Frequency: Every Day

Goal: Get more fit

Progress: Missed

Notes: Walk in Park

Date: May 2023

Habbit	Frequency	Goal	Progress	Notes	Date
Walk	Every Day	Get more fit	Missed	Walk in Park	13-05-2023

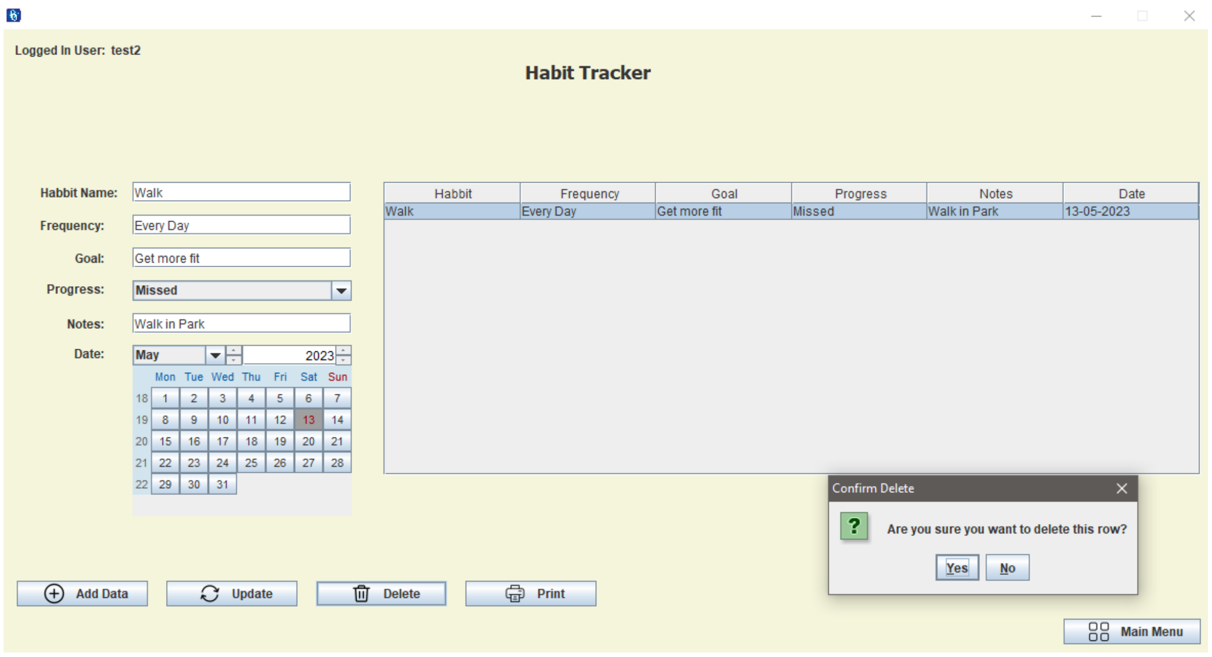
****

Name of Test Case: User Deletes Entry in Habit Tracker.

Description of Test Case: User Deletes Entry in Habit Tracker.

Expected Result of Test Case: Entry deleted.

Actual Result of Test Case: Entry deleted.


Logged In User: test2

Habit Tracker

Habbit Name: Walk

Frequency: Every Day

Goal: Get more fit

Progress: Missed

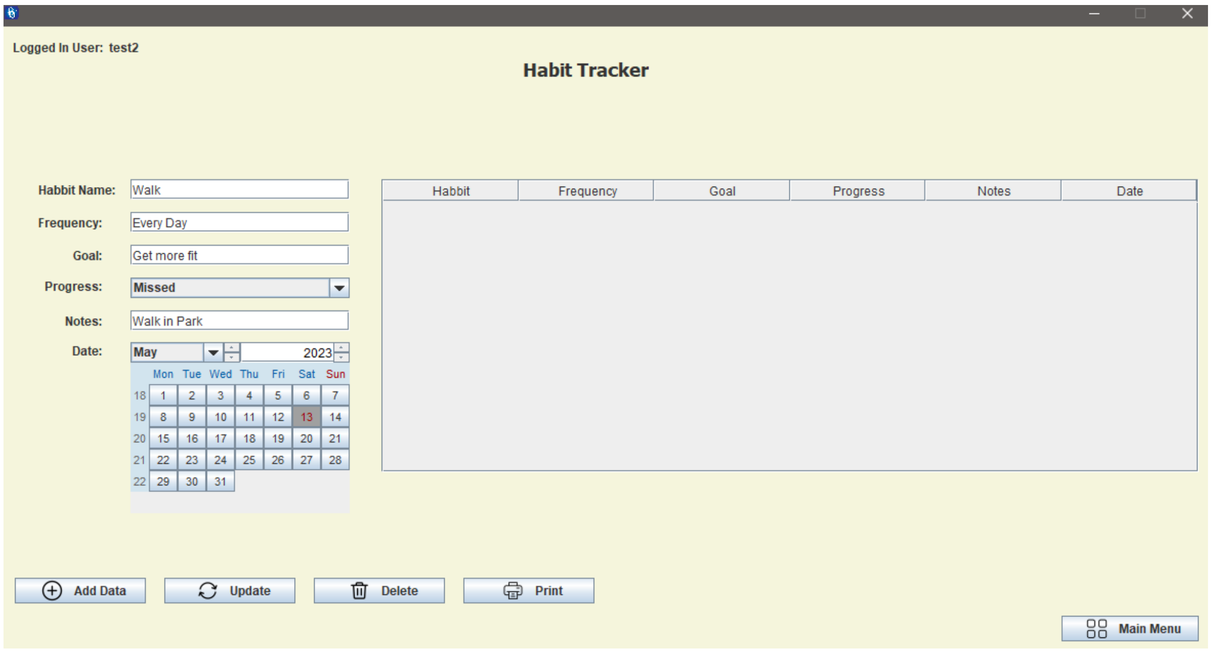
Notes: Walk in Park

Date: May 2023

Habbit	Frequency	Goal	Progress	Notes	Date
Walk	Every Day	Get more fit	Missed	Walk in Park	13-05-2023

Confirm Delete

Are you sure you want to delete this row?


Logged In User: test2

Habit Tracker

Habbit Name: Walk
Frequency: Every Day
Goal: Get more fit
Progress: Missed
Notes: Walk in Park
Date: May 2023

Habbit	Frequency	Goal	Progress	Notes	Date

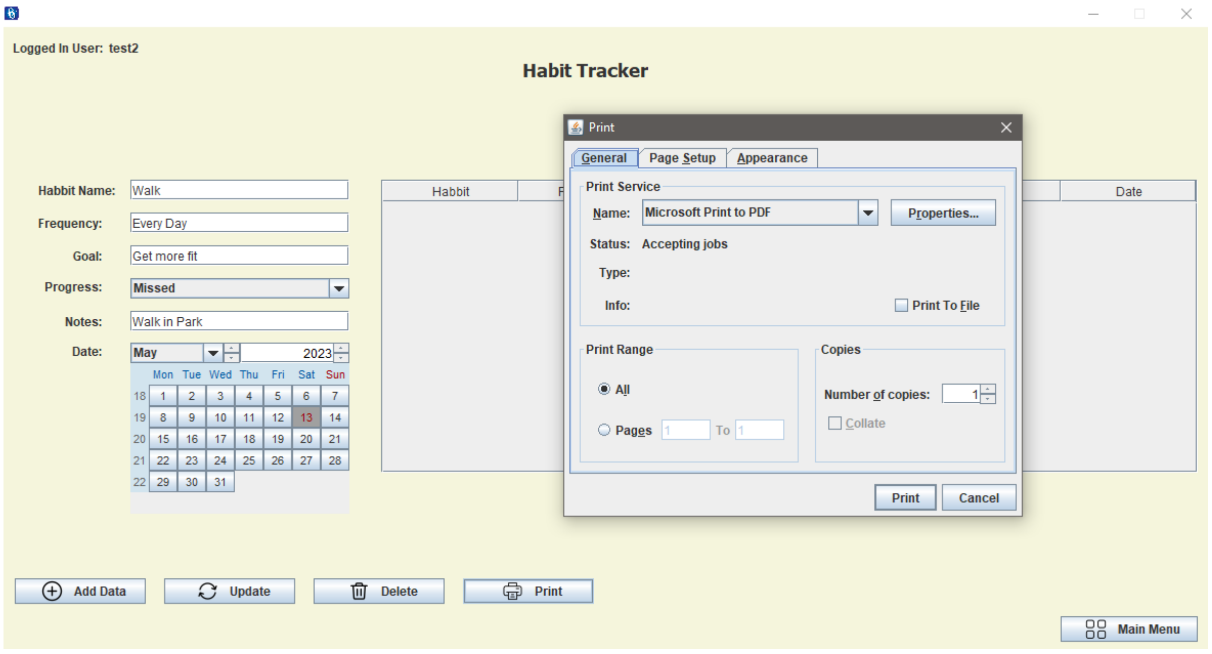
****

Name of Test Case: User presses print in Calorie Tracker.

Description of Test Case: Print button pressed.

Expected Result of Test Case: Print dialog opens.

Actual Result of Test Case: Print dialog opens.


Logged In User: test2

Habit Tracker

Habbit Name: Walk
Frequency: Every Day
Goal: Get more fit
Progress: Missed
Notes: Walk in Park
Date: May 2023

Print

Print Service
Name: Microsoft Print to PDF
Status: Accepting jobs
Type:
Info:
Print To File

Print Range
All
Pages 1 To 1

Copies
Number of copies: 1
Collate

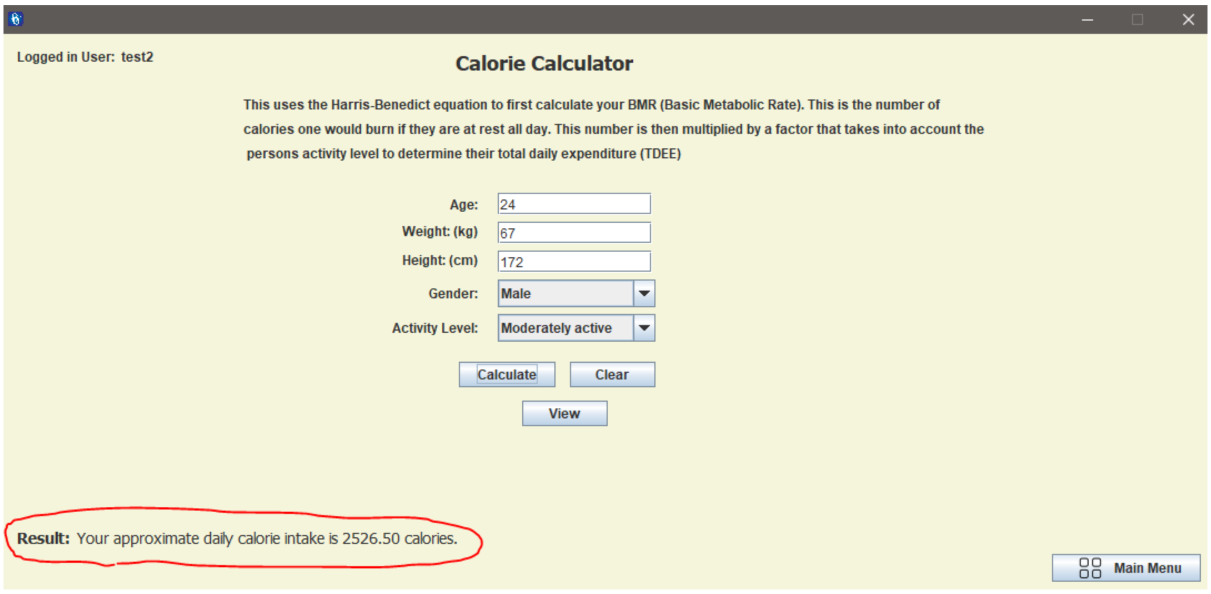
****

Name of Test Case: User calculates calories in Calorie Tracker.

Description of Test Case: User enters information to calculate calorie in Calorie Tracker.

Expected Result of Test Case: Calories are calculated.

Actual Result of Test Case: Calories are calculated and saved to database.



Logged in User: test2

Calorie Calculator

This uses the Harris-Benedict equation to first calculate your BMR (Basic Metabolic Rate). This is the number of calories one would burn if they are at rest all day. This number is then multiplied by a factor that takes into account the persons activity level to determine their total daily expenditure (TDEE)

Age: 24
Weight: (kg) 67
Height: (cm) 172
Gender: Male
Activity Level: Moderately active

Result: Your approximate daily calorie intake is 2526.50 calories.

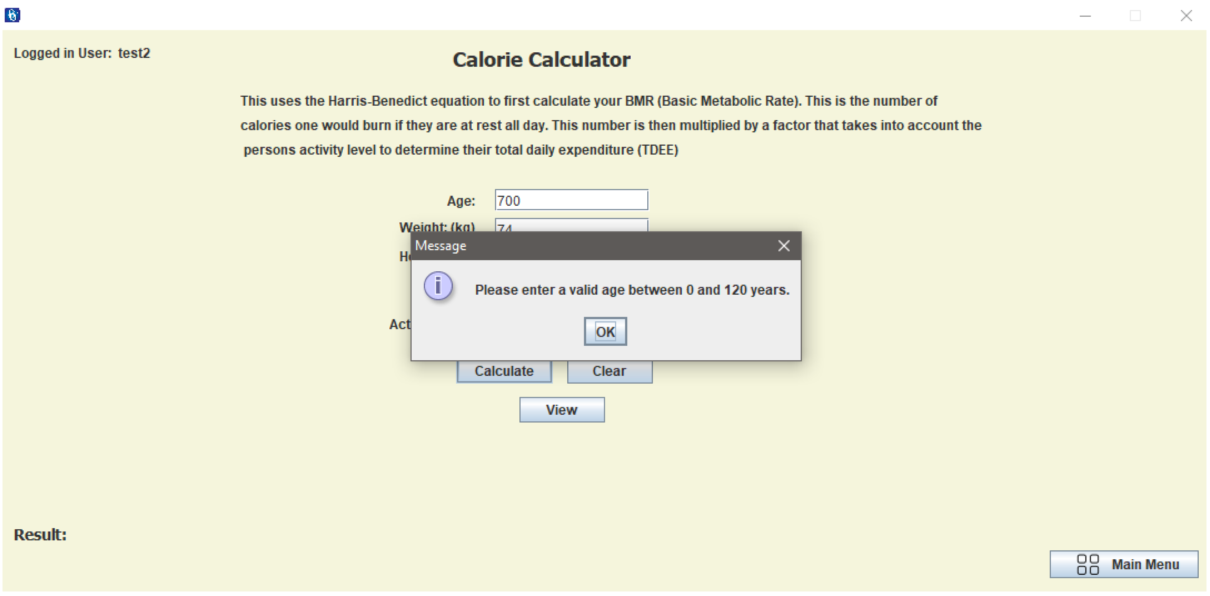
****

Name of Test Case*:* User enters unrealistic values in Calorie Tracker.

Description of Test Case: User enters unrealistic information in Calorie Tracker.

Expected Result of Test Case: Error message shown.

Actual Result of Test Case: Calories are calculated and saved to database.



Logged in User: test2

Calorie Calculator

This uses the Harris-Benedict equation to first calculate your BMR (Basic Metabolic Rate). This is the number of calories one would burn if they are at rest all day. This number is then multiplied by a factor that takes into account the persons activity level to determine their total daily expenditure (TDEE)

Age: 700
Weight: (kg) 74

Message

Please enter a valid age between 0 and 120 years.

Result:

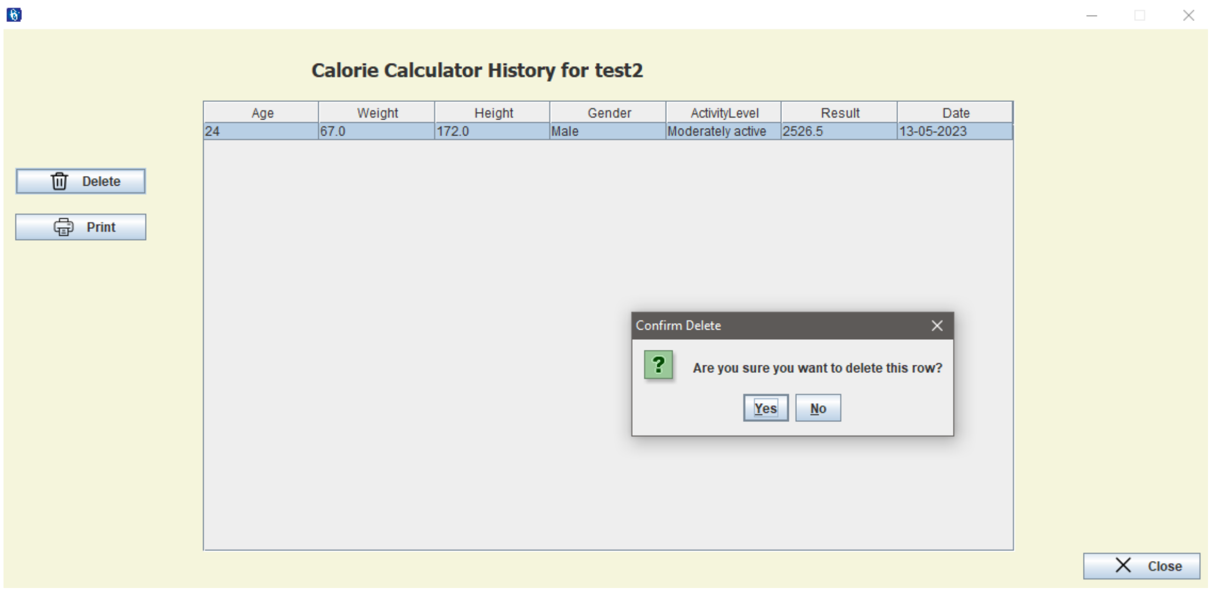
****

Name of Test Case*:* User deletes entry from view Calorie Calculator.

Description of Test Case: User deletes past Calorie Calculation.

Expected Result of Test Case: Database entry deleted.

Actual Result of Test Case: Database entry deleted.

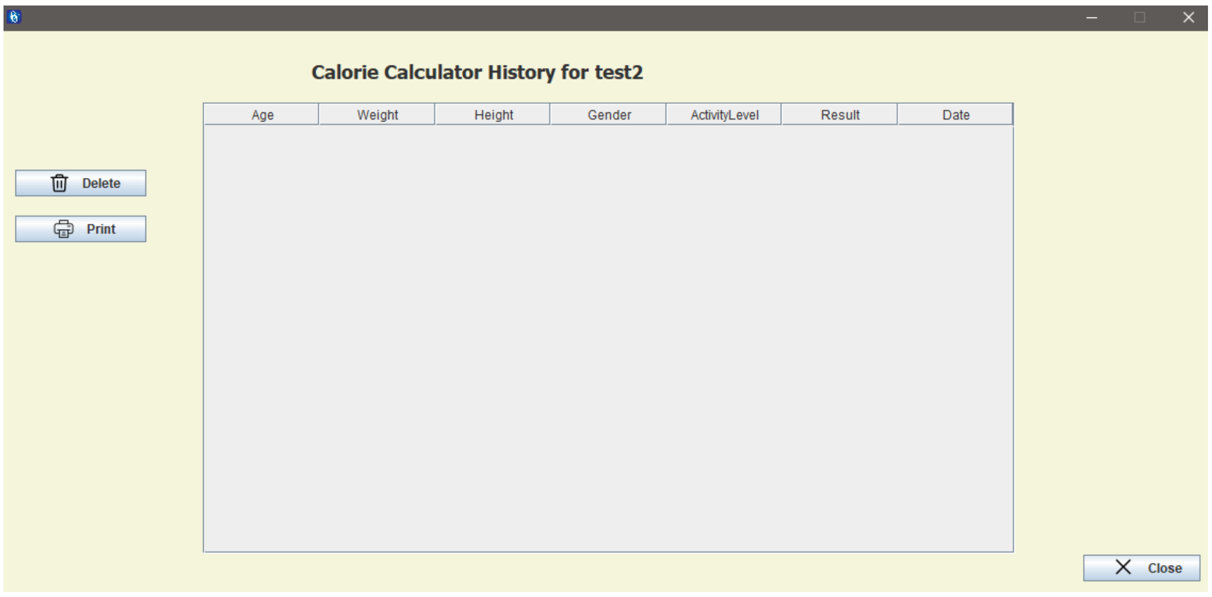


Calorie Calculator History for test2

Age	Weight	Height	Gender	ActivityLevel	Result	Date
24	67.0	172.0	Male	Moderately active	2526.5	13-05-2023

Confirm Delete

Are you sure you want to delete this row?



Calorie Calculator History for test2

Age	Weight	Height	Gender	ActivityLevel	Result	Date

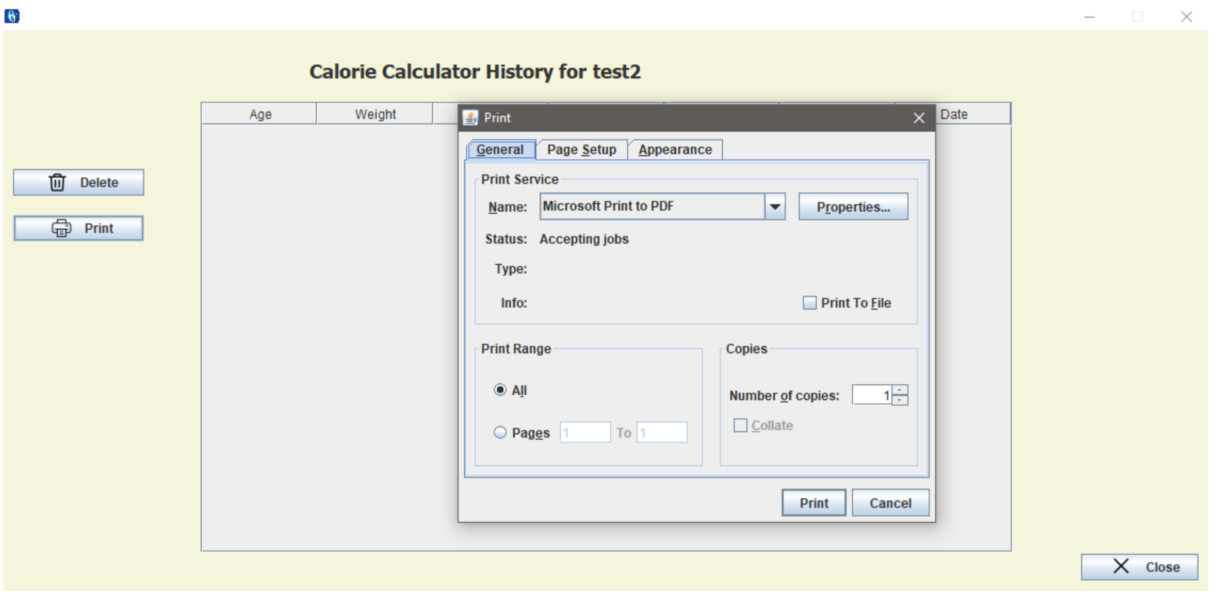
****

Name of Test Case*:* User presses print in Calorie Calculator.

Description of Test Case: User presses the print button in Calorie Calculator.

Expected Result of Test Case: Print dialogue opened.

Actual Result of Test Case: Print dialogue opened.


Calorie Calculator History for test2

Print

Print Service

Name: Microsoft Print to PDF

Status: Accepting jobs

Type:

Info:

Print Range: All

Number of copies: 1

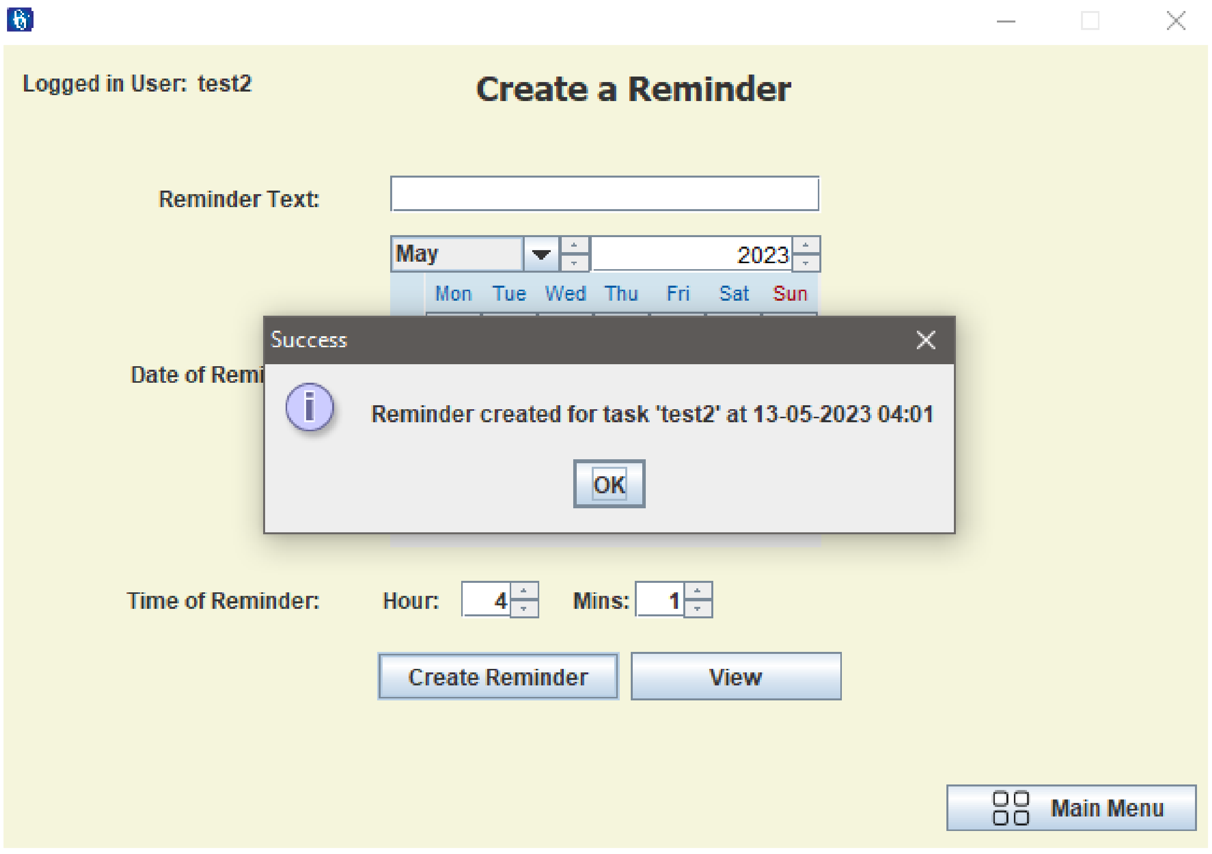
****

Name of Test Case: User sets reminder.

Description of Test Case: User sets a reminder in the reminder feature.

Expected Result of Test Case: Reminder goes off.

Actual Result of Test Case: Reminder goes off.


Logged in User: test2

Create a Reminder

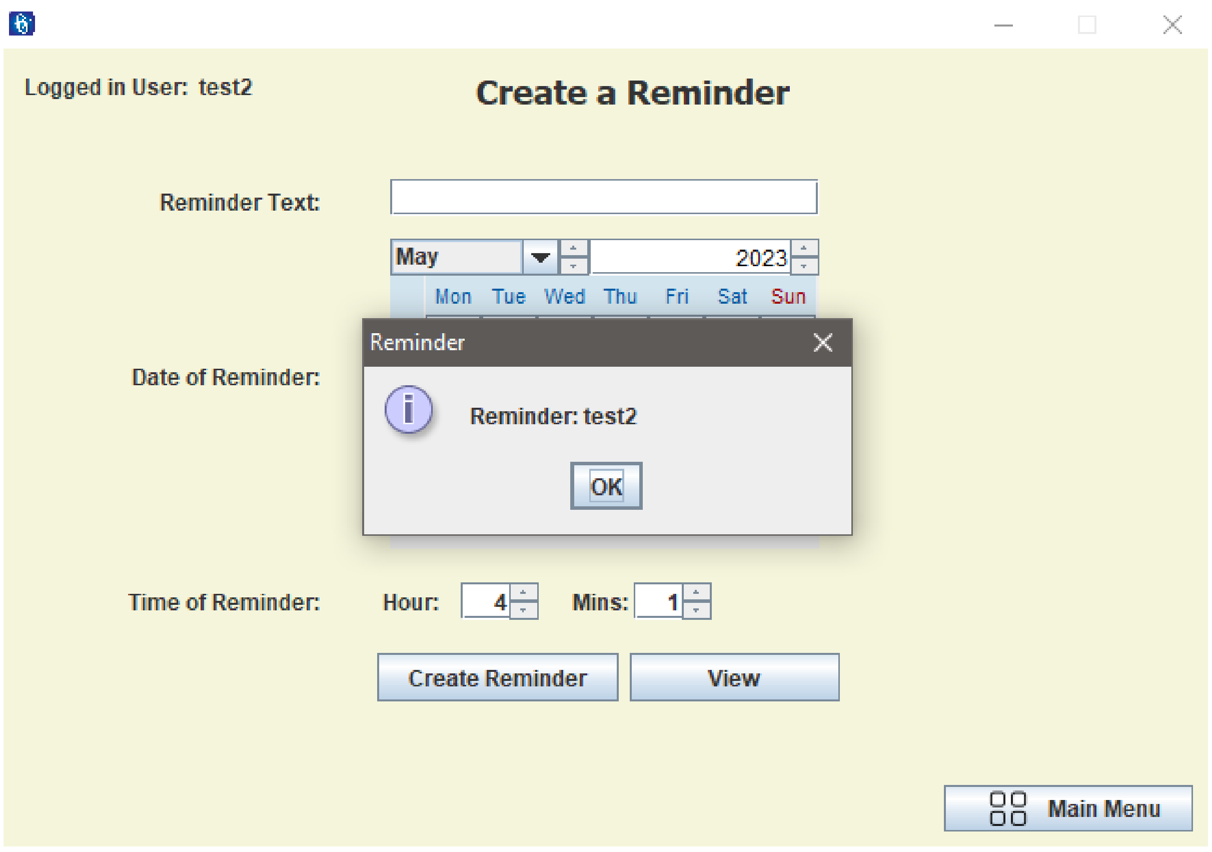
Reminder Text:

Date of Remi

Success

Reminder created for task 'test2' at 13-05-2023 04:01

Time of Reminder: Hour: 4 Mins: 1


Logged in User: test2

Create a Reminder

Reminder Text:

Date of Reminder:

Reminder

Reminder: test2

Time of Reminder: Hour: 4 Mins: 1

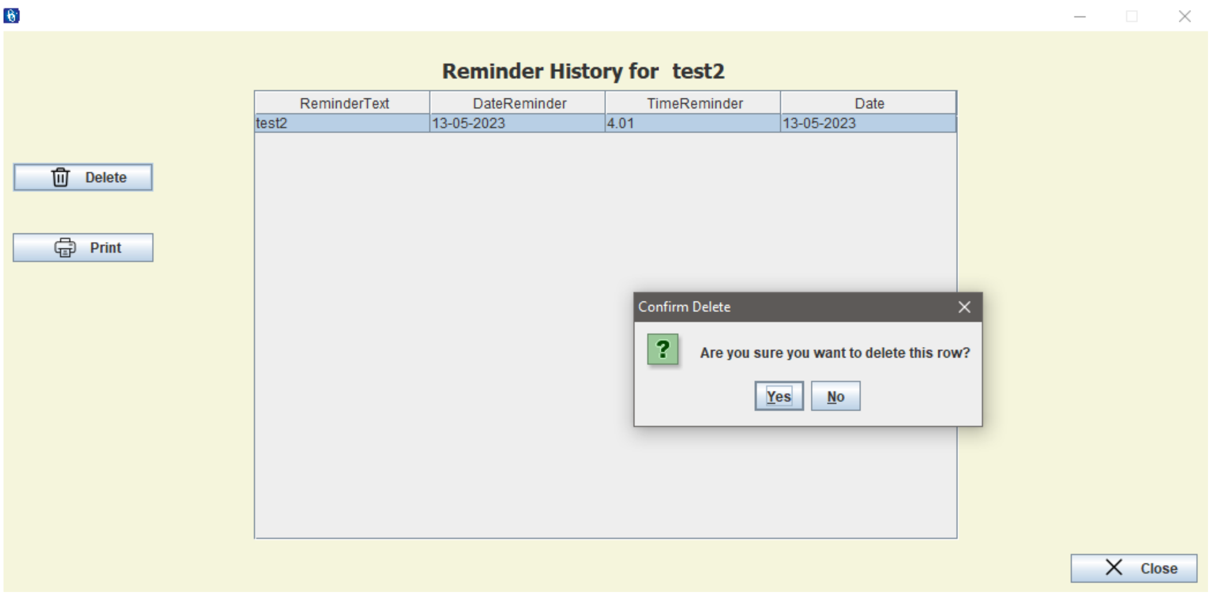
****

Name of Test Case: User deletes entry from view Reminder.

Description of Test Case: User deletes past Reminder.

Expected Result of Test Case: Database entry deleted.

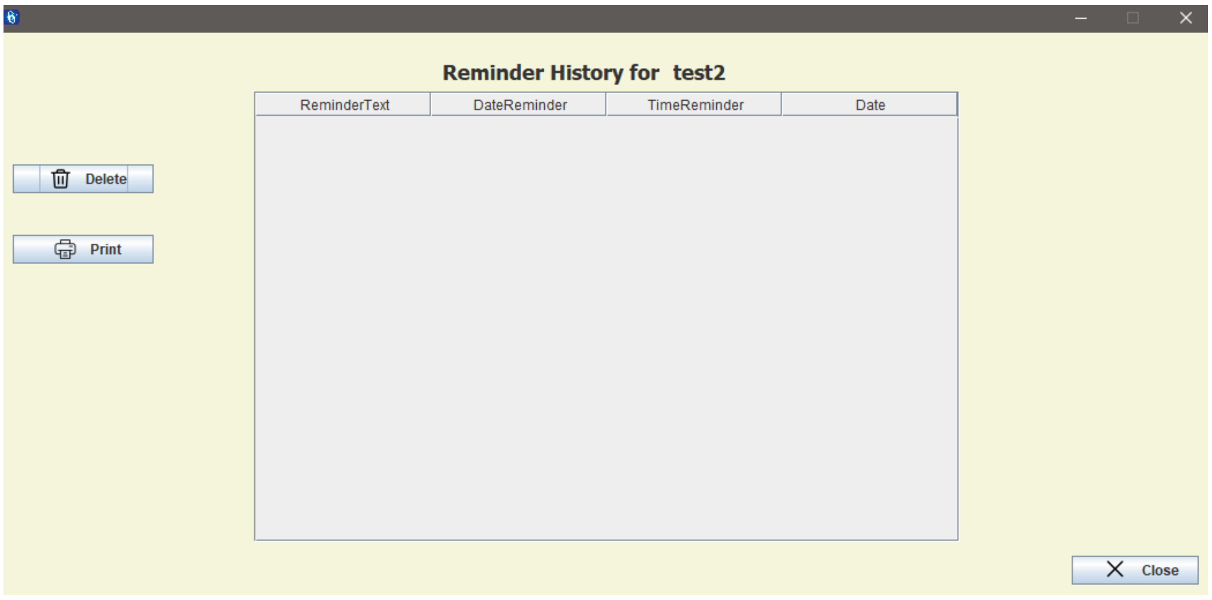
Actual Result of Test Case: Database entry deleted.


Reminder History for test2

ReminderText	DateReminder	TimeReminder	Date
test2	13-05-2023	4.01	13-05-2023

Confirm Delete

Are you sure you want to delete this row?



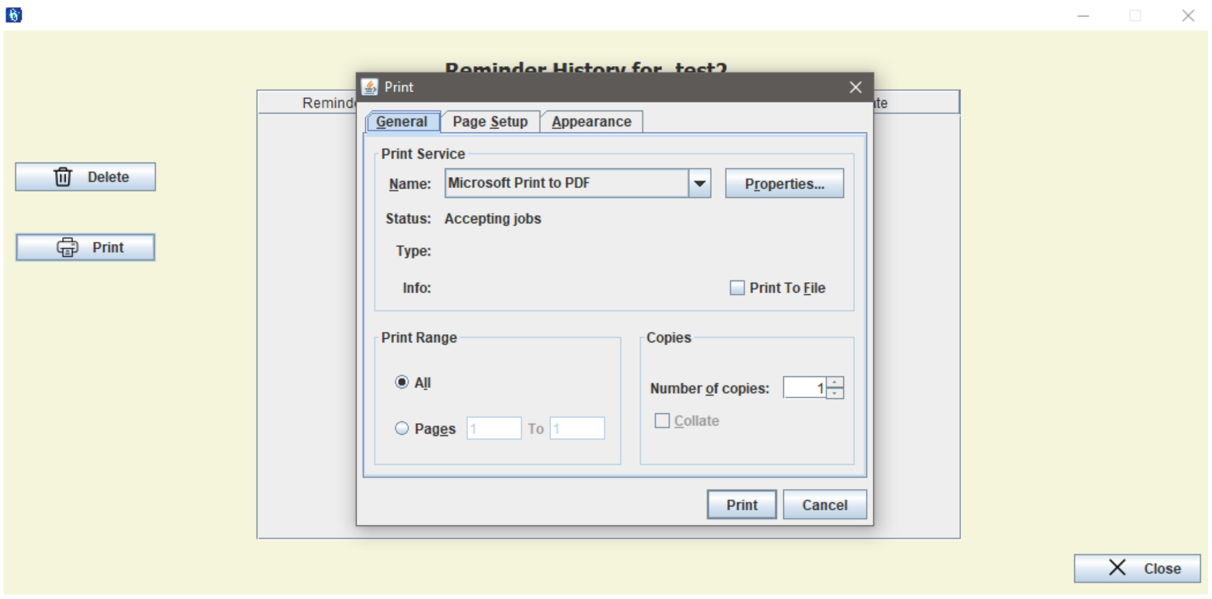
****

Name of Test Case*:* User presses print in Reminder.

Description of Test Case: User presses the print button in Reminder.

Expected Result of Test Case: Print dialogue opened.

Actual Result of Test Case: Print dialogue opened.



Print Service

Name: Microsoft Print to PDF

Status: Accepting jobs

Type:

Info:

Print To File: ☐

Print Range: [✓ All] Pages 1 To 1

Copies

Number of copies: 1

Collate: ☐

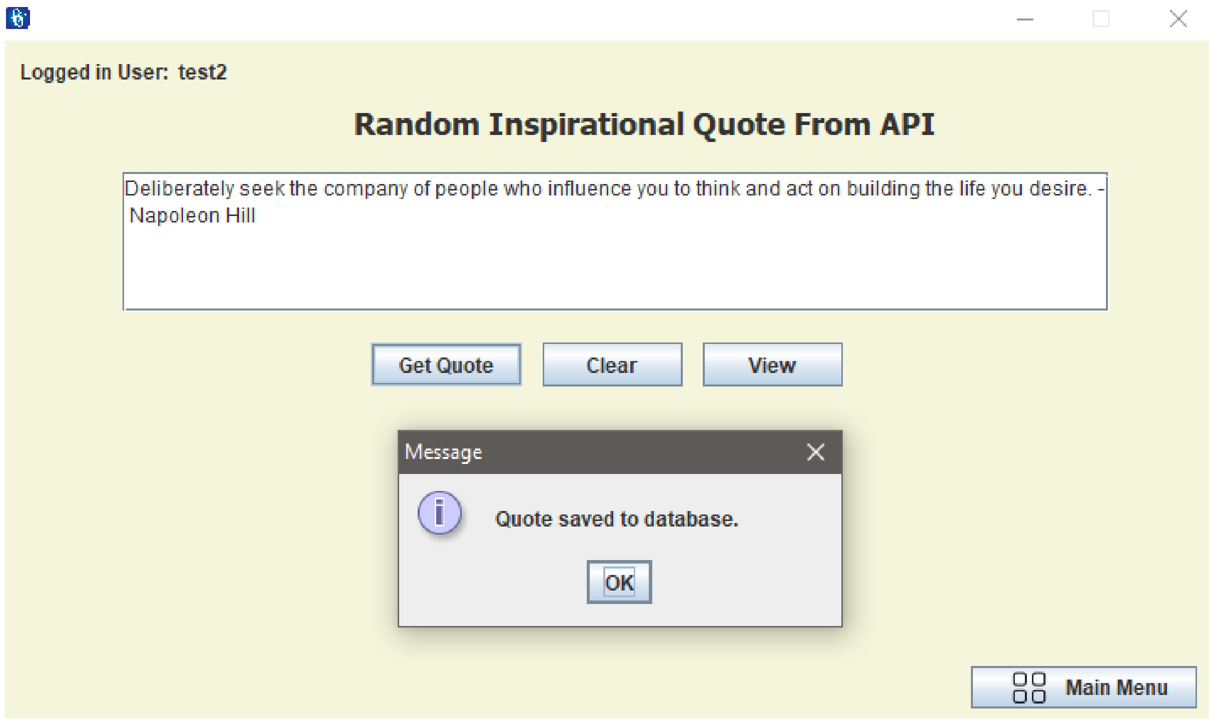
****

Name of Test Case*:* User presses get quote in the random quote feature.

Description of Test Case: User presses get quote in Random Quote.

Expected Result of Test Case: Random Quote Generated.

Actual Result of Test Case: Random Quote Generated.


Logged in User: test2

Random Inspirational Quote From API

Deliberately seek the company of people who influence you to think and act on building the life you desire. - Napoleon Hill

Quote saved to database.

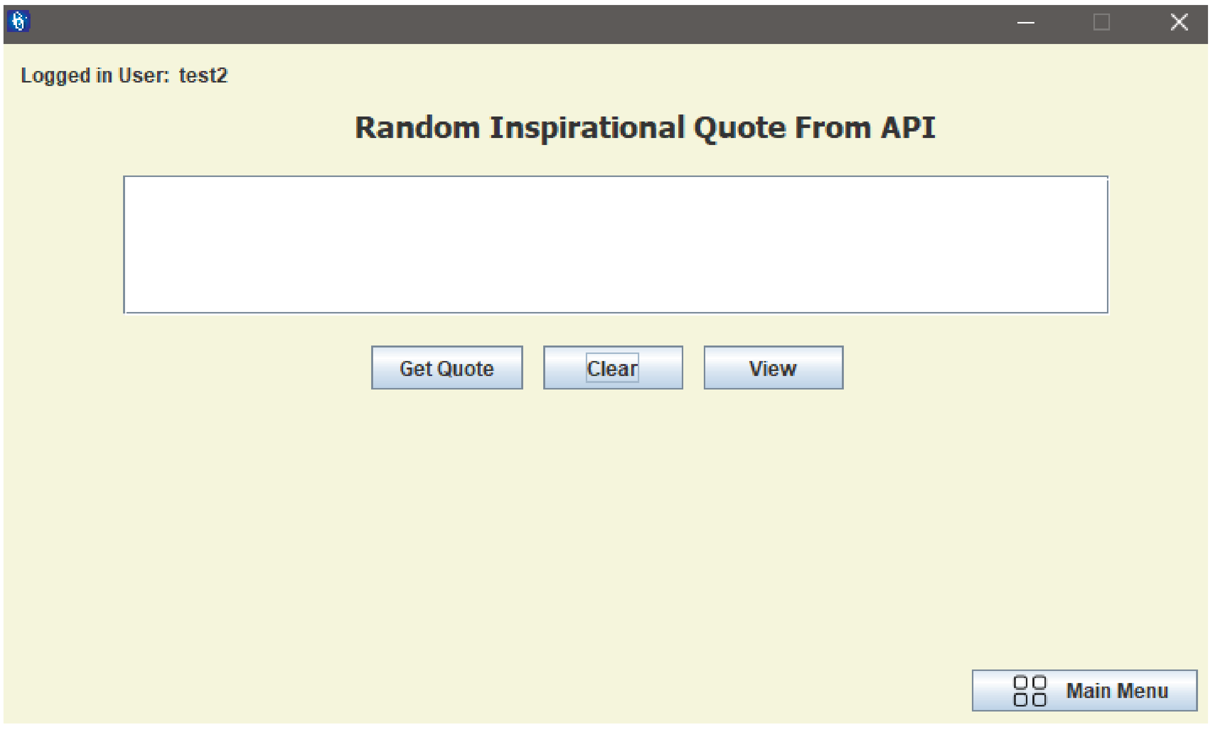
****

Name of Test Case*:* User presses clear in the random quote feature.

Description of Test Case: User presses clear in Random Quote.

Expected Result of Test Case: Random Quote cleared from text field.

Actual Result of Test Case: Random Quote cleared from text field.



Logged in User: test2

Random Inspirational Quote From API

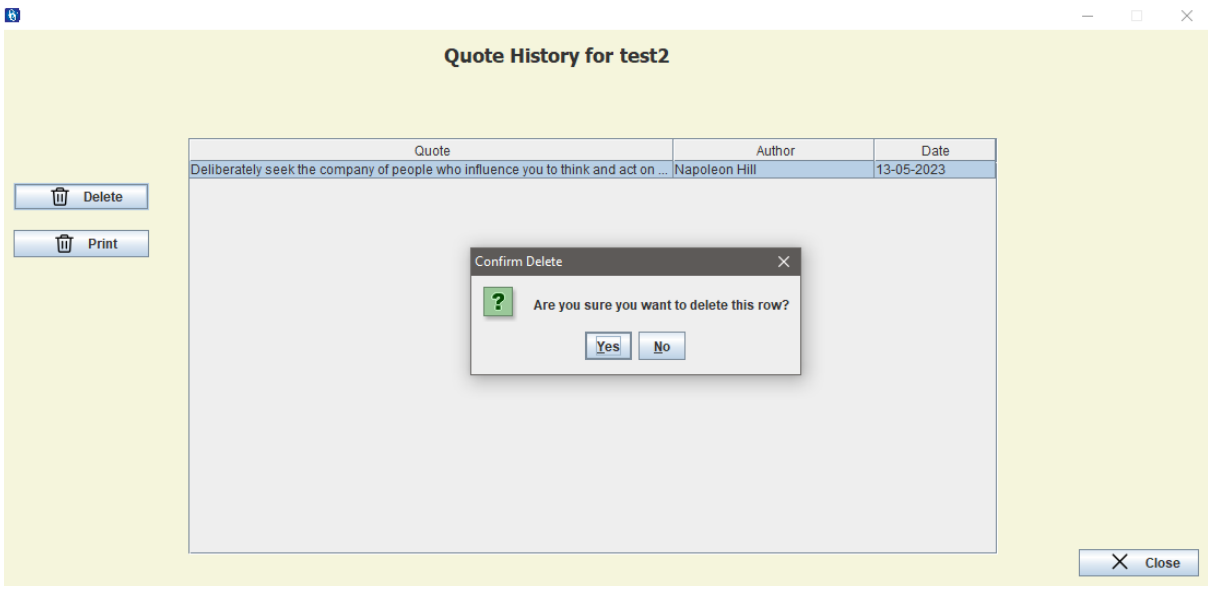
****

Name of Test Case: User deletes entry from view quotes.

Description of Test Case: User deletes past quote.

Expected Result of Test Case: Database entry deleted.

Actual Result of Test Case: Database entry deleted.

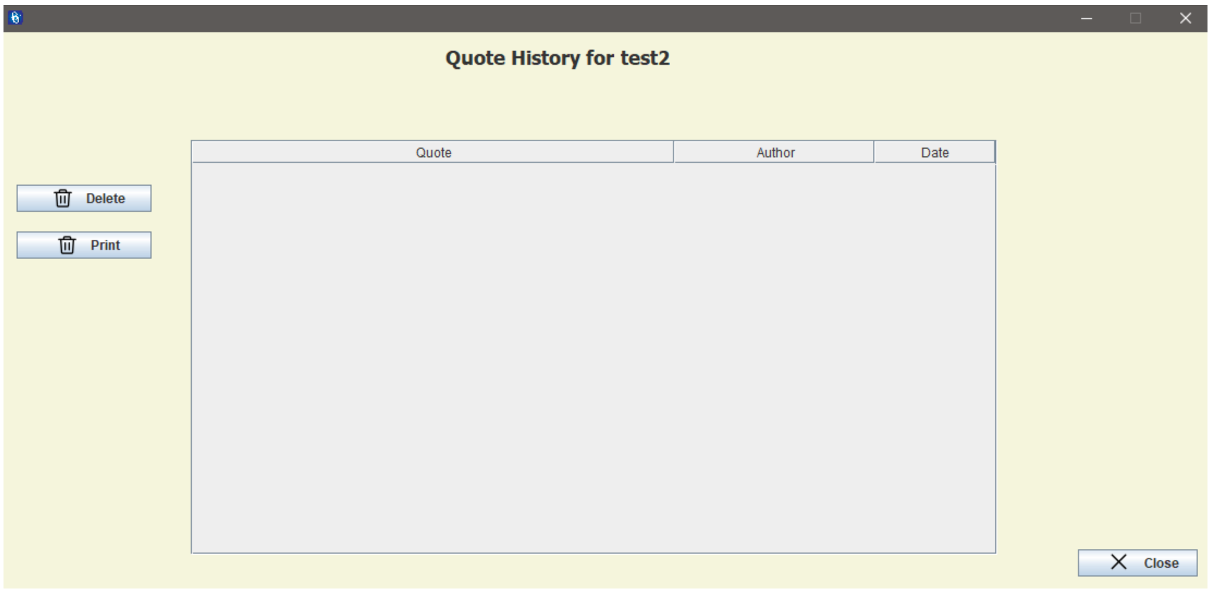


Quote History for test2

Quote	Author	Date
Deliberately seek the company of people who influence you to think and act on ...	Napoleon Hill	13-05-2023

Confirm Delete

Are you sure you want to delete this row?



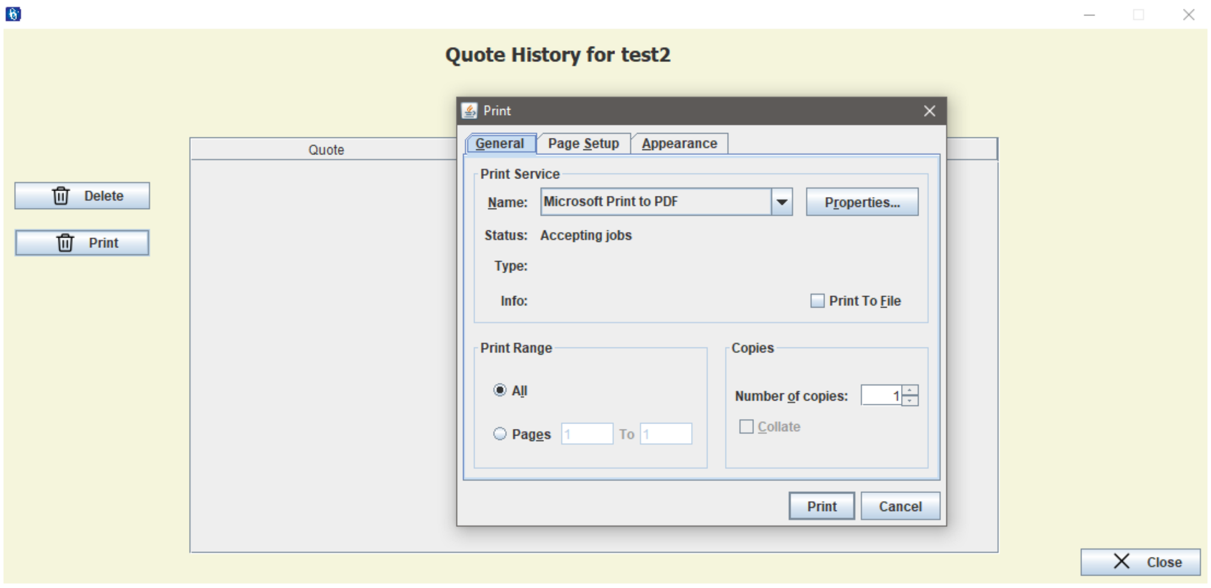
****

Name of Test Case: User presses print in Random Quote.

Description of Test Case: User presses the print button in Random Quote.

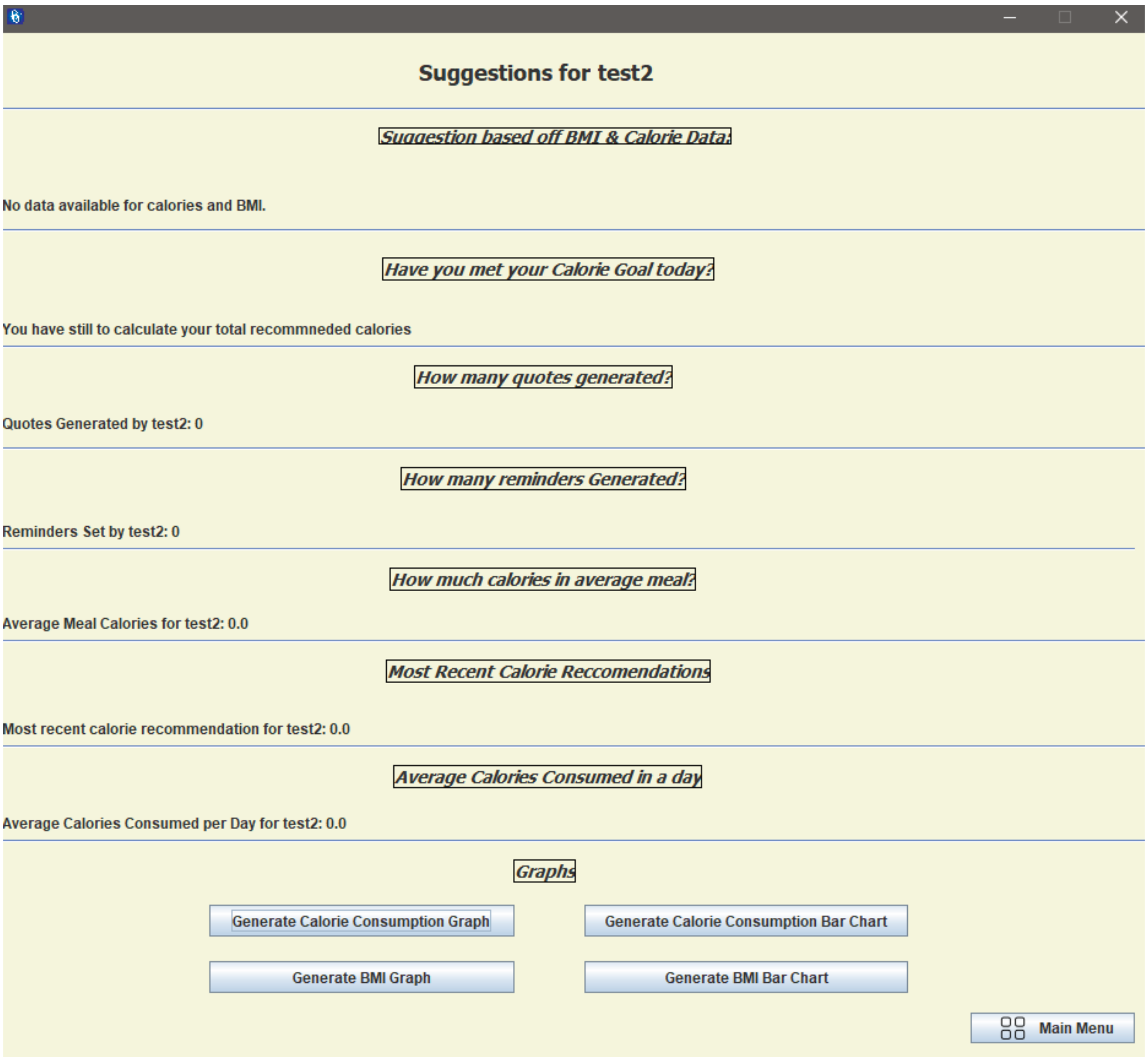
Expected Result of Test Case: Print dialogue opened.

Actual Result of Test Case: Print dialogue opened.



****

For this next part of the testing, I will have to populate the system with some information. As I have previously deleted all the entries added ill add some more and then come back to the stats and suggestion page. For now, the stats and suggestions page looks like this.


Suggestions for test2

Suggestion based off BMI & Calorie Data:

No data available for calories and BMI.

Have you met your Calorie Goal today?

You have still to calculate your total recommneded calories

How many quotes generated?

Quotes Generated by test2: 0

How many reminders Generated?

Reminders Set by test2: 0

How much calories in average meal?

Average Meal Calories for test2: 0.0

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test2: 0.0

Average Calories Consumed in a day

Average Calories Consumed per Day for test2: 0.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

There is not enough information for the stats and suggestions to give any stats or suggestion or even be able to generate any of the graph.

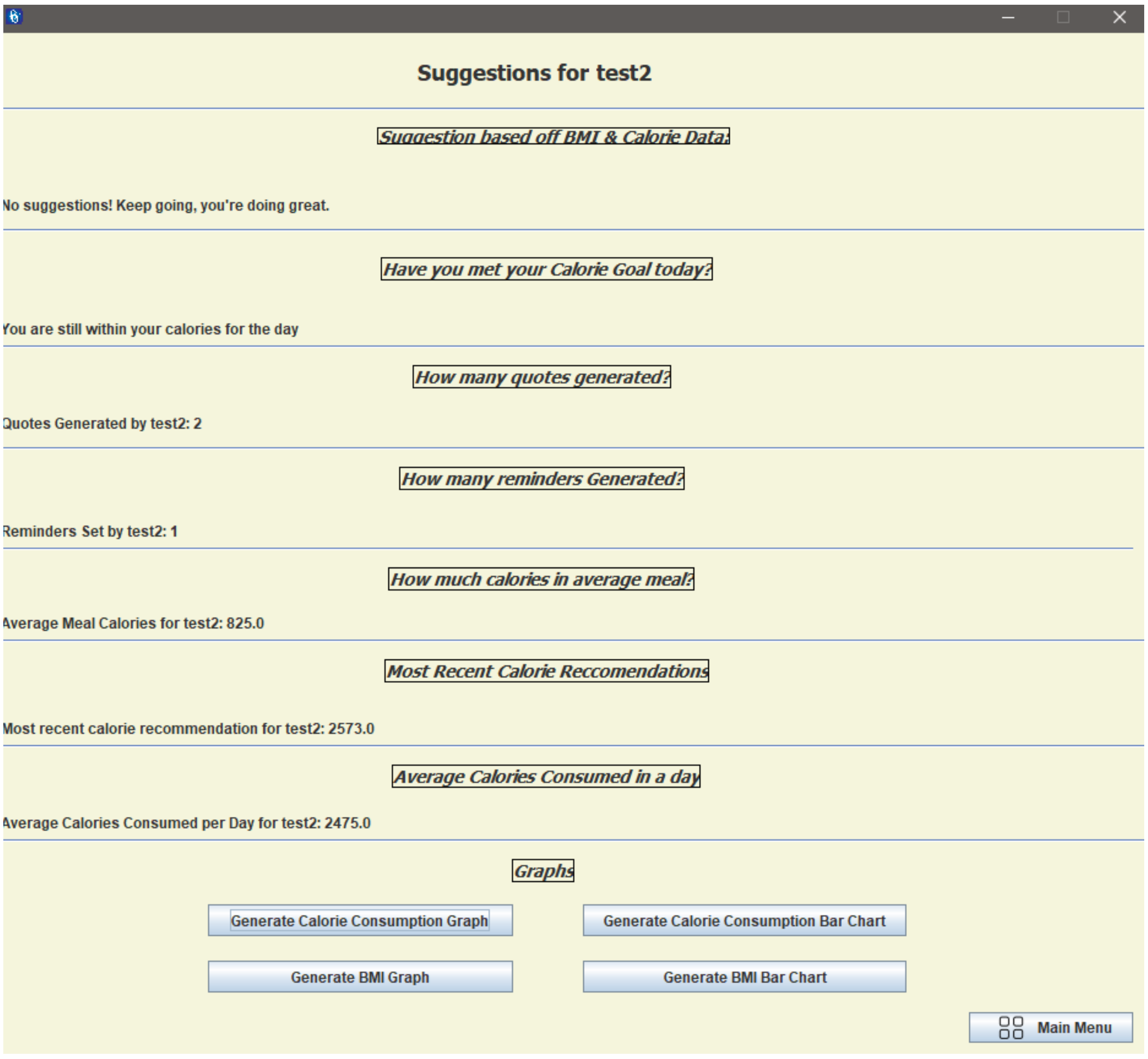
****

Name of Test Case: Stats and Suggestions Generates Stats and Suggestions.

Description of Test Case: Stats and Suggestions being generated.

Expected Result of Test Case: Some stats and suggestions created.

Actual Result of Test Case: Some stats and suggestions were created.


Suggestions for test2

Suggestion based off BMI & Calorie Data:

No suggestions! Keep going, you're doing great.

Have you met your Calorie Goal today?

You are still within your calories for the day

How many quotes generated?

Quotes Generated by test2: 2

How many reminders Generated?

Reminders Set by test2: 1

How much calories in average meal?

Average Meal Calories for test2: 825.0

Most Recent Calorie Reccomendations

Most recent calorie recommendation for test2: 2573.0

Average Calories Consumed in a day

Average Calories Consumed per Day for test2: 2475.0

Graphs

Generate Calorie Consumption Graph
Generate Calorie Consumption Bar Chart
Generate BMI Graph
Generate BMI Bar Chart
Main Menu

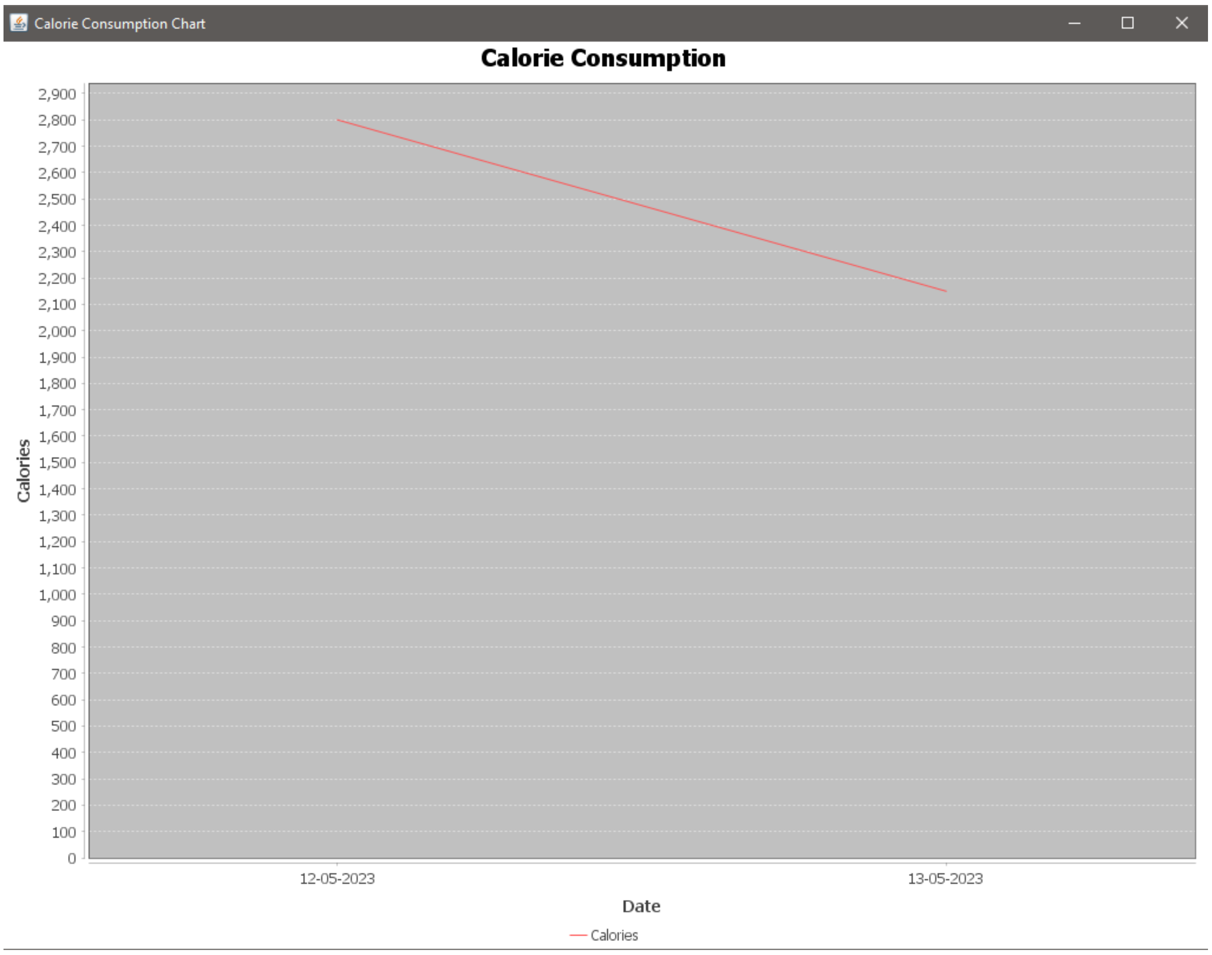
****

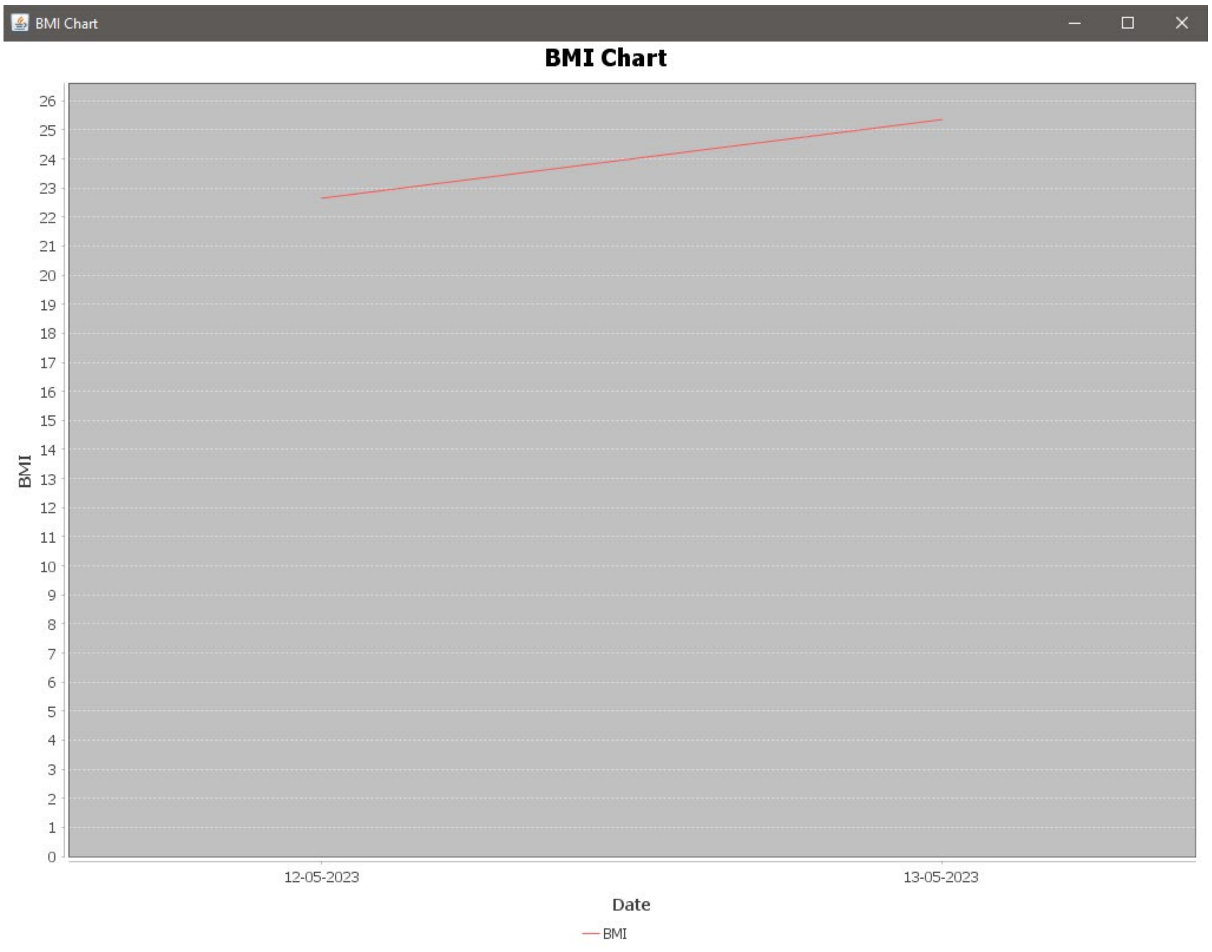
Name of Test Case: Stats and Suggestions Generates graphs.

Description of Test Case: Graphs being generated.

Expected Result of Test Case: Some Graphs created.

Actual Result of Test Case: Some Graphs were created.


Calorie Consumption Chart
Calorie Consumption
Calories
2,900
2,800
2,700
2,600
2,500
2,400
2,300
2,200
2,100
2,000
1,900
1,800
1,700
1,600
1,500
1,400
1,300
1,200
1,100
1,000
900
800
700
600
500
400
300
200
100
0
12-05-2023
13-05-2023
Date
Calories



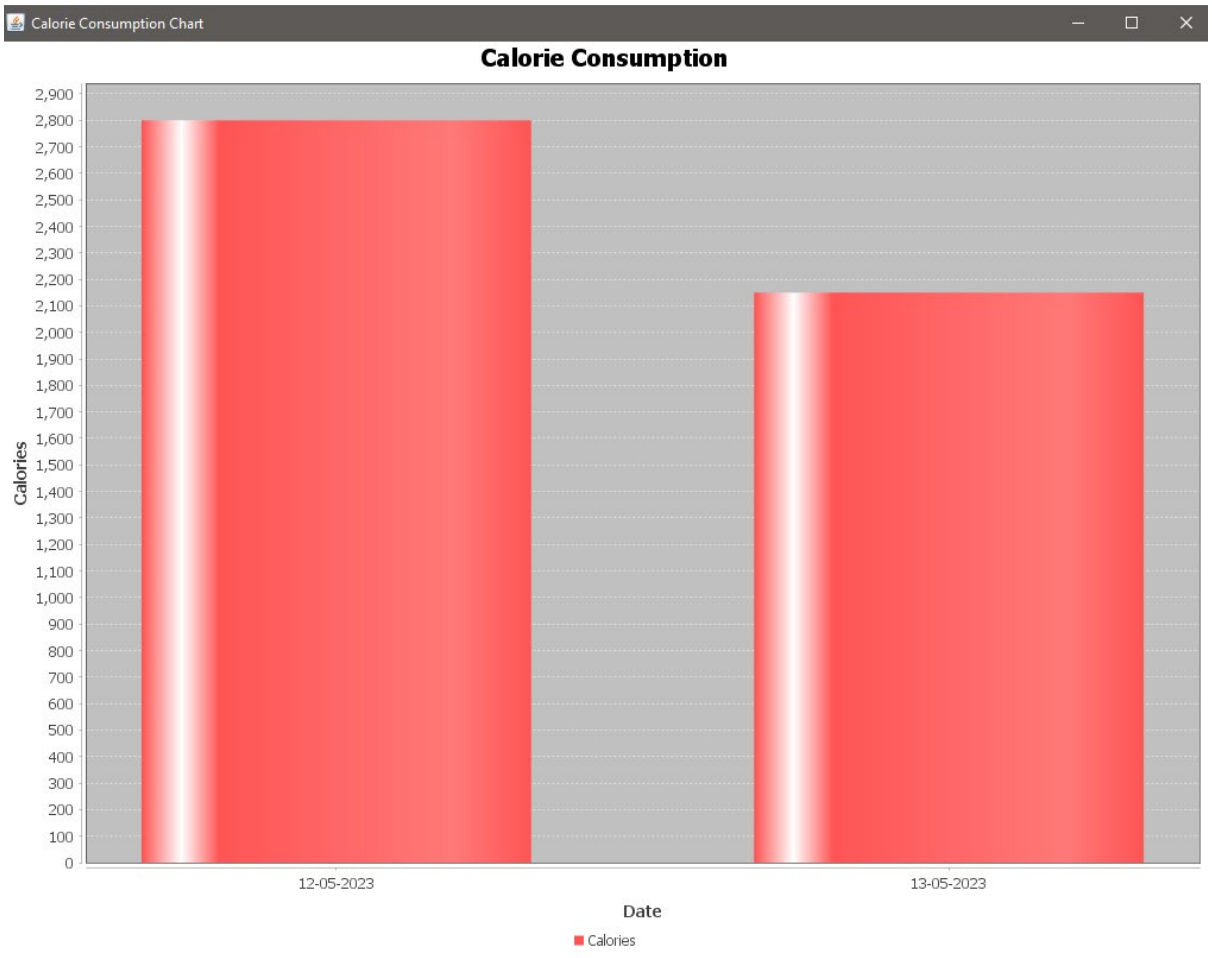
****

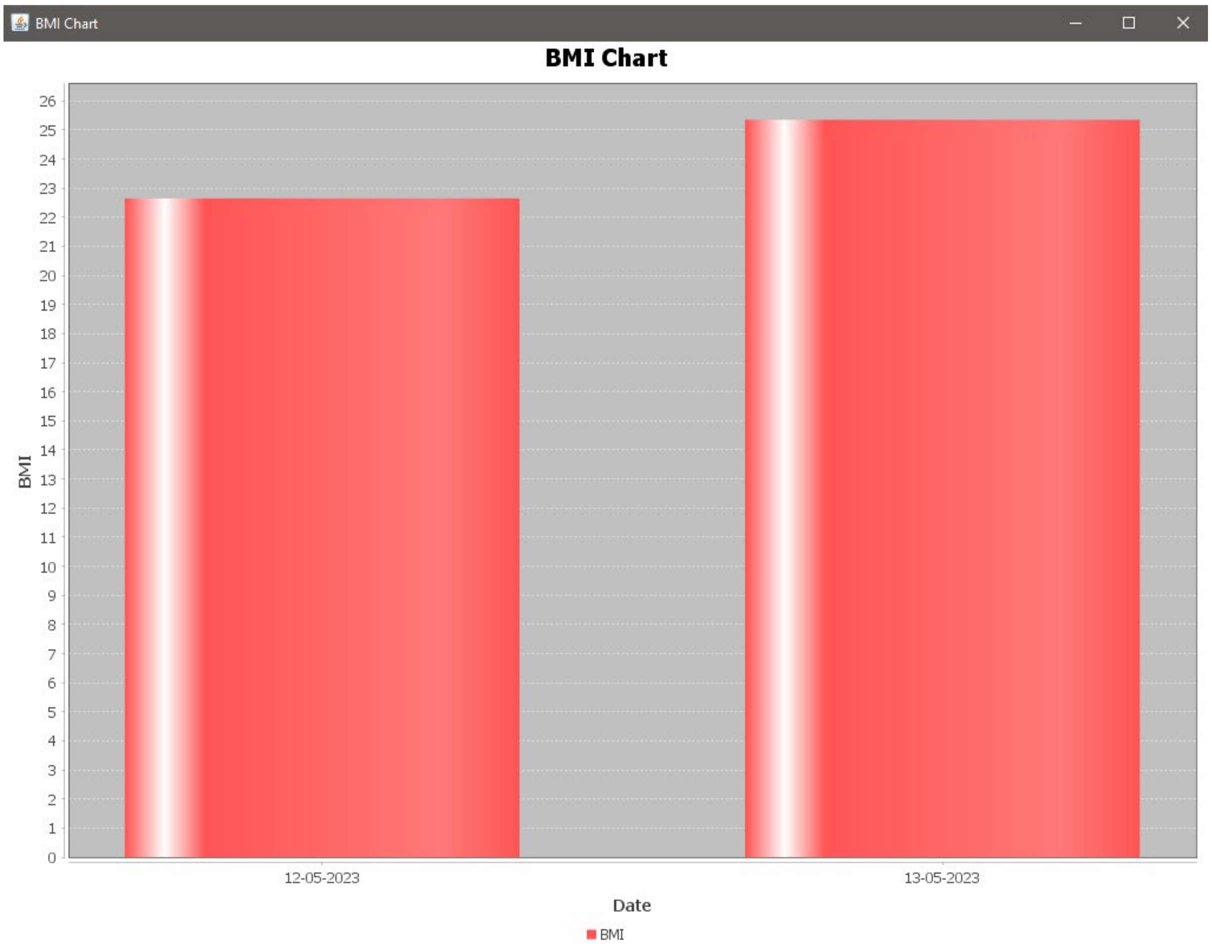
Name of Test Case: Stats and Suggestions Generates Bar Charts.

Description of Test Case: Bar Charts being generated.

Expected Result of Test Case: Some Bar Charts created.

Actual Result of Test Case: Some Bar Charts were created.


Calorie Consumption Chart
Calorie Consumption
Calories
2,900
2,800
2,700
2,600
2,500
2,400
2,300
2,200
2,100
2,000
1,900
1,800
1,700
1,600
1,500
1,400
1,300
1,200
1,100
1,000
900
800
700
600
500
400
300
200
100
0
12-05-2023
13-05-2023
Date
🟥 Calories



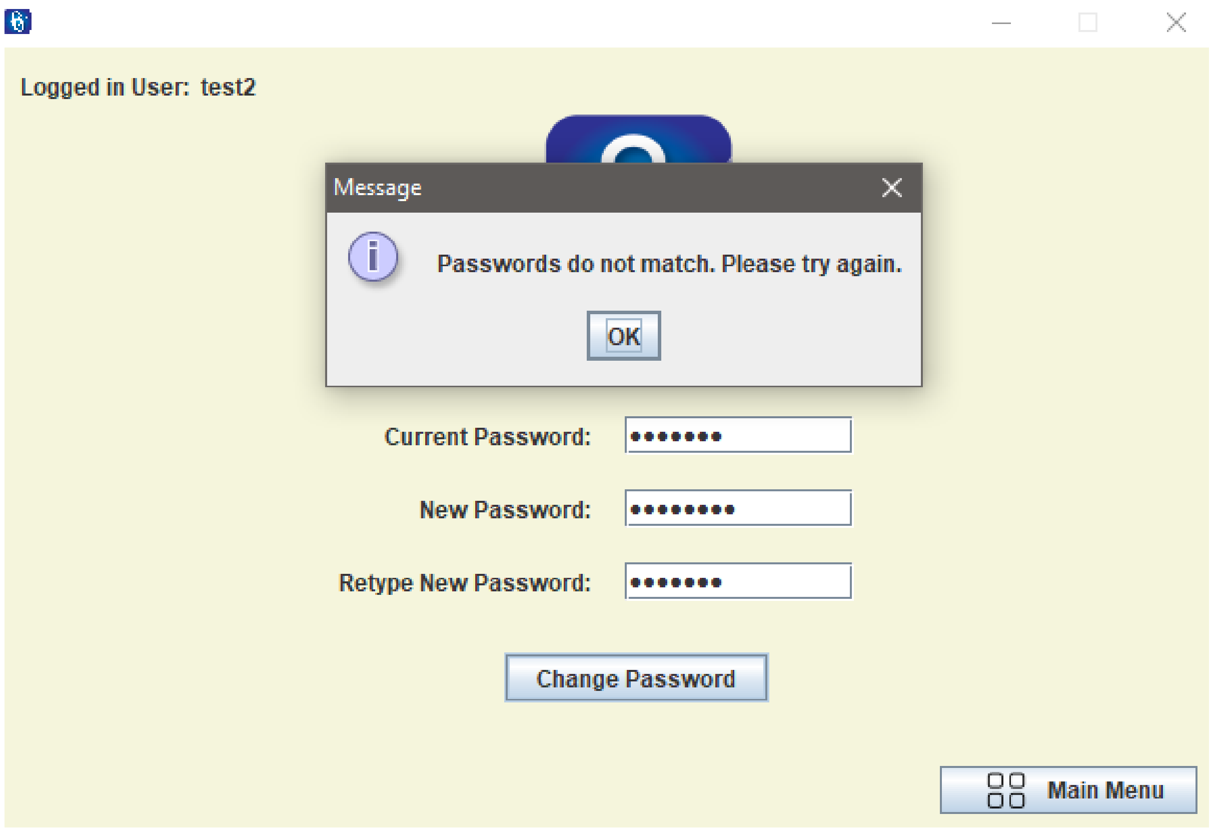
****

Name of Test Case: Change Password with mismatching new passwords.

Description of Test Case: Changing the password for the user with mismatching new passwords entry.

Expected Result of Test Case: Error message pops up.

Actual Result of Test Case: An error message came up.



Logged in User: test2

Message

Passwords do not match. Please try again.

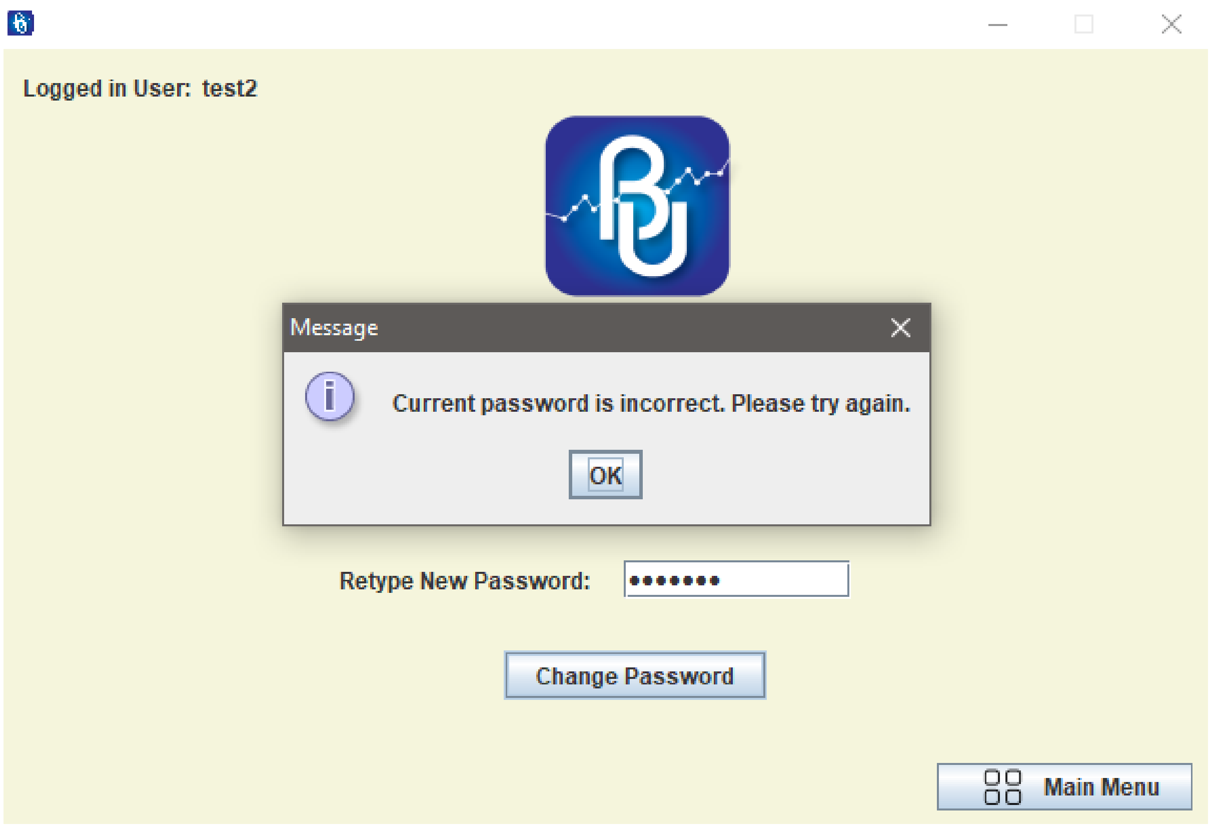
****

Name of Test Case: Change Password with incorrect current password.

Description of Test Case: Changing the password for the user with incorrect current password entry.

Expected Result of Test Case: Error message pops up.

Actual Result of Test Case: An error message came up.


Logged in User: test2

Message

Current password is incorrect. Please try again.

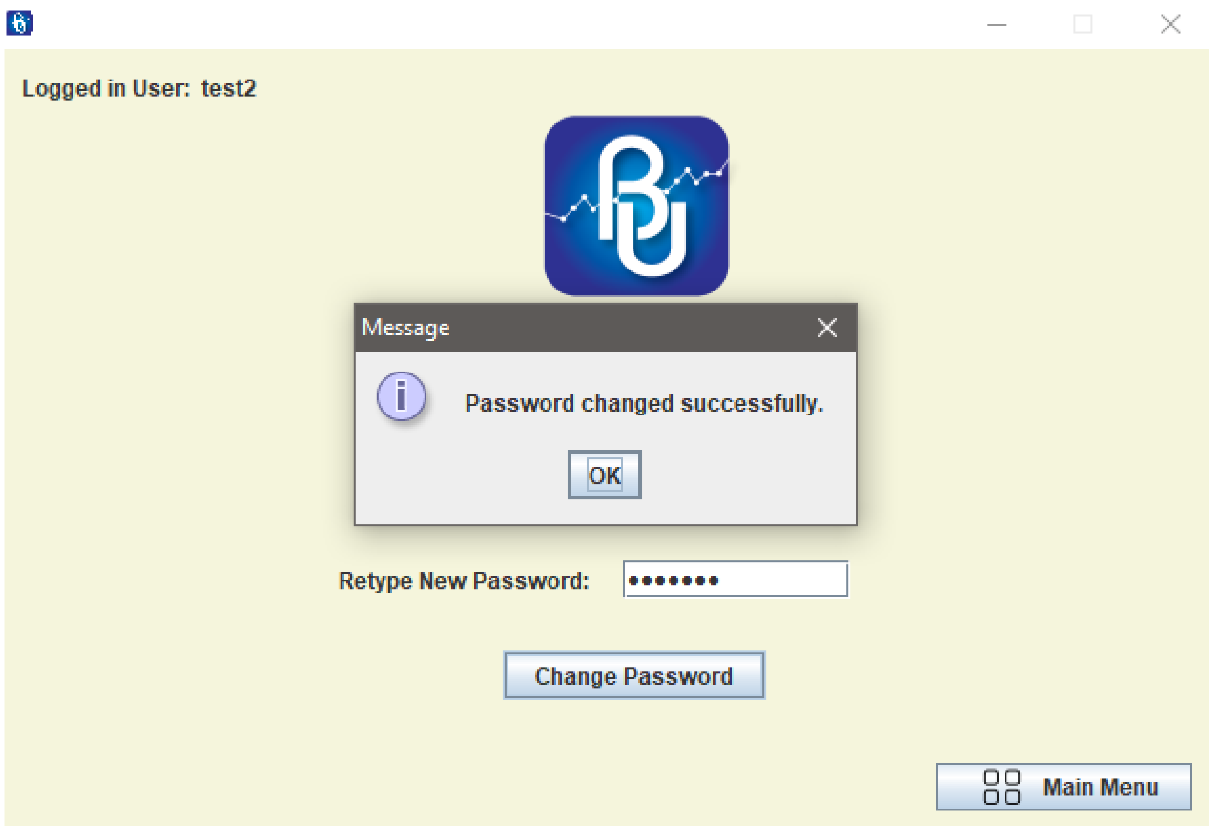
****

Name of Test Case: Change Password.

Description of Test Case: Changing the password for the user.

Expected Result of Test Case: Password for user changed.

Actual Result of Test Case: The Password was changed.


Logged in User: test2

Message

Password changed successfully.

This concludes the unit testing of my system.

2.7. Evaluation

To evaluate the system, I have decided to use Neilsen's Heuristics to evaluate the GUI. Neilsen has a set of 10 useability principals that help be able to be able to evaluate a GUI system. It was made by Jakob Nielsen who was a GUI expert.

1. Visibility of system status:
   I believe that this system has good visibility of status. Once a user is logged in to the system a message saying logged in user and their username is displayed on nearly every window. If it does not have this, it will explicitly mention their username in the title. This way a user knows always what account they are logged in with.
2. Match to world:
   This system matches the world as it has an intuitive interface that many users would have experienced before. The system also uses language that is appropriate and that has been seen in many other systems. This gives the user an air of familiarity with the system even if they have never seen it before.
3. User Control:
   This system is very easy to control and navigate. It is very intuitive. It uses big buttons and has input fields that are well described. For navigation it is all done with buttons that have images, so it is not only visually appealing but easy for the end user to use too.

4. Consistency:
   This system has a very consistent look. Its background and its icons are the same throughout the system. The buttons have the same icons for each feature in every window of the system. There are no windows with any outlandish designs. The system follows a uniform design.
5. Error Management:
   This system handles errors very well it. Upon prompting an error message, the system explains to the user why there has been an error. For example, in the BMI calculator the system checks to make sure there is valid input for the age height and weight of the user and if there isn’t the user is prompted of the issue.
6. Recognition:
   The fields in this system are well labelled to ensure that a user knows what to put in each field and what they represent. All the features in the system are appropriately labelled to ensure that the user knows what feature they are on.
7. Simplicity:
   Simplicity was an important consideration in this system. I did not want a user to feel overwhelmed when using the system, so I had simplicity in mind when designing the various features. I believe I have achieved the goal of simplicity as each window is not overcrowded or messy, instead this system has a clean feel throughout.
8. Error prevention:
   This system is designed in such a way to help mitigate the user encountering an error. Fields are well labelled, and the use of the system is intuitive.
9. Help:
   This system helps the user by using informative buttons to help the user know what they are doing in the system.
10. Flexibility:
    This system supports the use of multiple users and handles their data accordingly. It is designed as a system for use of all ages or background. Multiple accounts can be stored with various data attributed to each account.

I have been evaluating this system throughout the development and the testing processes and overall, I am very happy with the results. I evaluated the system on performance as well as usability and completeness. During the testing phase of my project, I evaluated the system performance. I did this by going through every feature of the program and ensuring it ran quickly and fluidly.

3.0 Conclusions

In conclusion this project has been an interesting project to produce. I had my ups, and I had my downs. Overall, I am happy with the result, and I am proud of the system I produced. I will now begin discussing the advantages and disadvantages of my project.

Advantages:

This program allows a user to be easily able to hold themselves accountable in their journey through self-improvement. It is a system that encompasses many features to facilitate a user eat better, maintain better habits, and keep themselves motivated. BetterU is an all-in-one tool meaning that it has all the tools necessary for improving oneself.

A big advantage of this system is that it is easy to use. The system is designed to be intuitive and easily grasped. There are also helpful prompts for a user if they may have entered incorrect input or if for example on the statistics page, they have entered no input at all.

Disadvantages:

A disadvantage of this system would be that it is a PC based application. I would like to develop this into an IOS and Android application too to be able to increase its user base. The making of the API request can also take a few seconds sometimes. In further versions I would try make this quicker.

4.0 Further Development or Research

If I had additional time and resources, I would add more features to this application. Some features to do with sleep would be nice to add as sleep is a very important part of our lives. If I had more time, I would have liked to get my friends and family to use the system to evaluate and gain some insight from their perspective. A possible feature I would like to add in the future would be a meal plan creator that can tailor the meal plan to your exact needs such as goals and if you have food allergies. With more resources I would keep the project going in the same direction, I would just make it more robust.

5.0 References

References

algo, b., n.d. [Online]
Available at: https://www.youtube.com/watch?v=wrd4NdmUhlI

Anon., n.d. *BMI Calculator » Harris Benedict Equation.* [Online]
Available at: https://www.bmi-calculator.net/bmr-calculator/harris-benedict-equation/

Anon., n.d. *Body Mass Index (BMI) Calculator.* [Online]
Available at: https://www.diabetes.ca/managing-my-diabetes/tools---resources/body-mass-index-(bmi)-calculator

Anon., n.d. *mvnrepository.* [Online]
Available at: https://mvnrepository.com/artifact/org.jfree/jfreechart/1.5.4

https://www.youtube.com/@thecsrevelation, n.d. *Java Project: Login and registration NetBeans with SQLite3 Database.* [Online]
Available at: https://www.youtube.com/watch?v=YZbXZ2yqGWQ

Kacper Pawlik, M. M. M. a. R. R., n.d. *BMR Calculator (Basal Metabolic Rate, Mifflin St Jeor Equation).* [Online]
Available at: https://www.omnicalculator.com/health/bmr

Koidan, K., n.d. *What Is a Nested Query in SQL?.* [Online]
Available at: https://learnsql.com/blog/sql-nested-select/

Life, P. o., n.d. *Change Background Color of JFrame.* [Online]
Available at: https://www.youtube.com/watch?v=fyA0RLYWw3w

Nielsen, J., n.d. *Nielsen Norman Group.* [Online]
Available at: https://www.nngroup.com/articles/ten-usability-heuristics/

Oamen, D., n.d. *How to Create an Employee Database Management Systems using SQLite in Java NetBeans - Full Tutorial.* [Online]
Available at: https://www.youtube.com/watch?v=MtYk5drk3SE&t=544s

toedter, n.d. [Online]
Available at: https://toedter.com/jcalendar/

6.0 Appendices

6.1. Project Proposal

National College of Ireland

Project Proposal

< To Be Named>

<22/12/22>

<Computimg>

<Software Development>

<2022/2023 >

<James Butler>

<x20129211>

<x20129211@student.ncirl.ie>

Contents

7.0 Objectives

(Max half Page)

What does this project set out to achieve?

Self-improvement is something that has seen an increase in popularity in recent years. On Instagram, YouTube and TikTok it is very easy to see within minutes someone promoting self-improvement or else documenting their own journey. More and more people these days are finding themselves lost, not knowing what to do with themselves or where to go next.

Why wouldn't you want to be the best version of yourself?

The goal of this app is to help the user improve their life. This app is a tool to facilitate in the journey of self-improvement. My app will have a variety of tools to enable better habits to be created for who is using the app. As this will be an overall self-improvement and habit tracker app there are functionalities which I believe fulfil this goal.

The first functionality is a calorie tracker. This can aid someone who wants to loose/gain weight. I will also add a journal. This journal is to be encouraged to do daily. Journalling is very good to do for the mind and to track progress. There will also be functionality to add reminders these reminders can be set in the app and will display for what the user has set for. For example, a reminder to do 30 mins light exercise in the morning or even to take a 10-minute mindfulness mediation. I would also like to add a page to display a positive message for the day. This can be looked at while using the app to re-affirm the user and to provide positive reinforcement.

8.0 Background

(Max half Page)

Why did you choose to undertake this project? How will you meet the objectives set out in Section 1.0?

I chose to do this project as I have become interested in going to the gym and with nutrition this year. It began when I went for a blood test in June this year (2022). I found out that I had high cholesterol. I was told I would need to change my diet and to try get this number down. I then started a whole new eating regiment and paid attention to my calories and found a passion in becoming healthier. I have always wanted to get into the gym and now I found a great reason to. As I was getting healthier and stronger, I took interest in self improvement and started applying some of the principals to my life. My whole life got much better. I started to become a lot happier.

I want my app to be able to help another person become a better and happier version of themselves. I like to help people and I think an app that can help achieve this outcome for me and another person is a great idea. I want to inspire people to stop engaging in destructive behaviours and lead an overall better life as this is something I have struggled with in the past. I will meet the objectives that are set out in section 1 by creating an app that will allow have the functionality to help someone eat better and keep on top of their mental and physical health.

9.0 State of the Art

(Max half page)

What similar applications exist already? What makes your project stand out? How does it differ from similar work of others?

There are similar applications to my application. Some of these apps are for mental wellbeing such as Headspace. Headspace helps promote better mental wellbeing by using a range of meditation techniques. The aim of Headspace aim is to make a person more mindful. There are also applications that count calories such as MyFitnessPal. This application is quite useful for managing your calories and macros. There are also many note taking applications that can be used for journalling like google keep but the app I will create will keep progress of the journals and remind you to journal. My application will be a one for all solution to an overall improved life. It will be unique as it is a one stop app for most of the user's self-improvement needs.

This app will differ from similar apps as it will be the only comprehensive app dedicated to overall improving of an individual.

10.0 Technical Approach

(Max 1 page)

What approach will you take to development? How will you identify requirements? How will you break down requirements into project tasks, activities and milestones?

The approach I will take for development is an agile approach. Since there are no stakeholders or investors, I will be outlining the requirements for the app myself. To identify requirements, I will be conducting a requirements analysis. A requirement analysis is a when you identify analyse and manage the requirements of an application to be able to determine the objectives of a project and to make sure that there are no conflicting requirements. To do this an approach I have used in the past and could work for this project is to outline the HLD (high level design) and then the LLD (low level design). The HLD will outline the scope of the project, what the project is and what it stands for. The LLD will have requirements relating to certain functions in the application.

To get a list of requirements I will begin by listing off all the features I want to be in this application. Once the features are listed, I will make sure to delete any duplicates and to order them by priority. It is important to define these requirements fully. For me I will be thinking of requirements in the form of use cases from the point of view of an end user. For example, a user can input text into the journal and save it to a database. The functional and non-functional requirements (HLD and LLD) will be then ensured of feasibility. After this it is then possible to sign off.

As I am using an agile approach, I will use a Kanban board of some sort to be able to break down the project tasks, activities, and milestones. There are many Kanban boards such as Jira or Trello. For me I will use the headings of To-Do, doing and finished. Everything I need to be done will be in the to do section. The stuff that is currently being worked on will be in the doing and then finished stuff will be in the finished. As something goes from to-do to doing it will be moved across the board to correspond with its status.

11.0 Technical Details

(Max 1 page)

Implementation language and principal libraries. What are the important algorithms or approaches under consideration for this work?

This project will be implemented in the Java programming language. As I am a creating an android application, I have two choices which are Kotlin or Java. I decided to go with java as I have some familiarity with the java programming language. To create this application, I will have to ensure it can run on just about any android phone. Luckily there is emulators in android studio that emulate the environment of an android phone and tell you how many phones that the software being created will be compatible with. The emulator I will use will enable the app to be used with over 98 percent of phones.

While creating this application I will be using some third-party libraries to help with the functionality and the useability of my project. A library that takes my interest is a library known as OkHttp. This library can be used for my random quote section in my application. This library allows for HTTP requests to be made. I want to be able to pull a quote in off the internet as apposed to having a prewritten list of quotes.

Another library I would like to use is the crashlithics library. This is a lightweight crash reporter. I believe this could be helpful in the creation of my project and for testing purposes.

Gson could also be a handy library if I am trying to parse some form of JSON into my application. Although I am not certain of if ill need this functionality yet. I believe it is worthwhile to mention.

An important approach to consider for this work would be the stats page where I want to be able to log the users progress as they use the app. I want to be able to pull in information such as how much they used the app. The streak they are on. How many days on target for their calorie goals etc. I want the stats page to be able to provide feedback but in a constructive manner. I want to make sure the end user does not feel bad about themselves. This is very important to me as I want to achieve a positive change in somebody's life and not a bad one.

12.0 Special Resources Required

(Max half page)

What special resources if any will be required for this work?

There will be no special resources required for this project. The only resource is a laptop with an internet connection for research and development. Android studio for development and an Android phone for proper testing. Other than these there are no other resources needed. All the above resources I already have in my possession.

13.0 Project Plan

(Max 2 pages)

Project plan with details on implementation steps and timelines. This project plan should provide as much detail as possible for now and will be revised with more detail with the mid point documentation.

To ensure proper planning of this project I will use project management tools such as a Trello board and Gantt charts. This will help me to be able to make consistent progress and to be able to stay on task. With the scope and the objective defined it is now time to be scheduling tasks. The tasks on a task-by-task basis will be done through the Kanban method of using a Trello board. The Gantt chart will have a high-level timeframe set out.

The beginning of this project will consist of getting the prerequisites set up and ready to go. Since I have established an idea and have the functional and non-functional requirements done at this point, I can begin by thinking about how the UI will look. While doing this I will also concentrate om making simple applications in android studio to become familiar with the IDE. I will follow my Gantt chart below as this will help me stay on track and give me deadlines to work to throughout the SDLC.

TASK	PROGRESS	START	END
Project Inception			
Project Proposal	100%	22/10/2022	30/10/22
Research Project	100%	20/10/22	28/10/22
Compare the competition	100%	1/11/22	12/30/22
Project Ethics Form	25%	13/11/22	15/11/23
Functional and non-functional requirements	100%	20/11/22	30/11/22

Midpoint presentation	75%	10/12/22	20/12/22
Analysis			
Create Name and Logo	20%	5/1/23	10/1/23
Use case Diagrams	0%	12/1/22	25/1/23
Create Mock-up of UI	100%	15/12/22	20/12/22
Documentation	0%	25/2/23	01/02/23
Development			
Create UI	0%	1/02/23	10/02/23
Create Functionality	0%	10/02/00	1/04/23
Finalise Features and UI	0%	01/04/23	08/04/23
Devise Tests	0%	08/04/23	13/04/23
Documentation	0%	13/02/00	17/04/23
Testing			
Use the created tests	0%	17/04/23	23/04/23
Test on multiple devices	0%	23/04/23	1/05/23
Documentation	0%	1/05/23	16/05/23
Submission	0%	17/05/23	17/05/23

As seen above the development will follow a 4-phase plan. This plan consists of inception, analysis, development, and testing. Since the inception plan at this point would be complete. I will now discuss the analysis part of the plan.

Once feeling comfortable with android studio, I will then create a name and a logo for my application. I want the name and the branding of the application to have a modern and clean feel. Upon finishing the creation of the name and logo I will create the use case diagrams for the program. This will help me map out the flow of the program and it will make development easier. I will then also create mock-ups of the UI using paint.net or MS paint. This will facilitate the creation of the UI in the design phase. Once completed it is time to do the documentation. I will document everything I have done in the design phase.

In the development phase of the project, it will begin by creating the UI in android studio. After this it is time to get down to the nitty and gritty and work on the core functionality of my project. This functionality will be the most time-consuming part of my project. The functionality consists of a calorie counter, Journal, reminders/habit former and a statistics page. Once this functionality is implemented it is then time to finalise the functionality and the UI and do the documentation for the design phase.

It is now time to move onto the final phase of the SDLC which is the testing phase. I will do this by devising certain tests for each of the functionalities to ensure that they are working. Once I have confirmed the features are working, I will then move on to testing the application across multiple devices to ensure compatibility across each android devices. It is then time to document this phase and then submit my project.

14.0 Testing

(Max 1 page)

Describe how you will evaluate the system with real technical data using system tests, integration tests etc. If applicable describe how you will evaluate the system with an **end user. (be careful here re Ethics etc)**

In this phase I will be extensively testing my program to ensure that it works correctly. That means all functionality is to be working and that there are no random crashes. Android studio is good as it gives the option to be able to emulate the application on a device of your choice. This is great as I will be able to then test my application on a range of mobile devices. This whole phase will be dedicated to investigation and to discovery. It will be me ensuring the code and the finished application adheres to the requirements set out.

Devising the test plan is of utmost importance. I will be conducting quality assurance tests along with also integration tests. The quality Esurance tests are to ensure that the finished product meets the requirements set out. The system integration tests are where I will use a whole host of different phone emulators within android studio to ensure that the app works across all android devices. Upon conducting these two tests I will then use the app myself and see how it is for daily use. This app is designed to be used every day, so I want the user experience to be as pleasant as possible.

I as an end user will evaluate this app and will distribute it amongst some of my family to see what they think and if they can see anything about the app that I cannot. I think getting another perspective is always a good idea. Another person might also spot potential bugs that I might not see.

14.1. Reflective Journals

15.0 Supervision & Reflection Template

Student Name	James Butler

Student Number	X20129211
Course	BSHCSD4
Supervisor	Keith Maycock

16.0

17.0 Month: October

What?   This month I spent doing some research on potential Ideas for my project. This involved researching the topic of my project and the technologies I would use to implement it.
So What?   This means that I am starting to develop a vision in my mind of what the project will be. Although I did not submit a proper Idea for the project proposal, I will keep researching to find an appropriate idea.
Now What?   The challenges that I still must address can be worked on by starting to think about my mid-point presentation and doing more research on what to create for my project.

Student Signature	James Butler

Supervision & Reflection Template

Student Name	James Butler
Student Number	X20129211
Course	BSHCSD4
Supervisor	Keith Maycock

Month: November

What?

This month I have decided to change streams so that has been my primary focus. I have not thought much about the project.

So What?

This month I have not done too much on the project. I have decided to change streams from Cyber-Security to Software Development so my focus is to get up to speed on the Software Development stream.

Now What?

Now I will get up to speed on the Software Development stream and then think about my project idea.

Student Signature	James Butler

Supervision & Reflection Template

Student Name	James Butler
Student Number	X20129211
Course	BSHCSD4
Supervisor	Keith Maycock

Month: December

What?

This month I have decided on my project I am going to make a self-Improvement application in Android Studio

I have completed my midpoint presentation video.

So What?

This month I have chosen my project. This means I now have a vision of what I want to produce for my project. I have a goal to work towards. I can now start to envision how the development of this app will unfold. The success was that I am happy with the project I have chosen, and it gives me a clear goal to work towards.

I have also completed my midpoint presentation which is a big relief.

Now What?

I need to decide on what features are to be included in the application.

Student Signature	James Butler

Supervision & Reflection Template

Student Name	James Butler
Student Number	X20129211
Course	BSHCSD4
Supervisor	Keith Maycock

Month: January

What?

This month I have been focusing on exams so not much done regarding the project.

A little research has been done for potential features in the application.

So What?

I now have my exams done which means I can focus better on the project.

Some successes are that I have a clear head now to look at my project.

Now What?

I need to start the development of the system.

Student Signature	James Butler

Supervision & Reflection Template

Student Name	James Butler
Student Number	X20129211
Course	BSHCSD4
Supervisor	Keith Maycock

Month: Feburary

What?

This month I have started to watch tutorials and putting together a plan.

So What?

This month I have made progress in the fact I am starting to watch tutorials and building an image in my head of how I am going to go about creating this system. I am getting the pieces together before I start development.

Now What?

Get developing ASAP. Time to implement some of the Android Studio Tutorials.

Student Signature	James Butler

Supervision & Reflection Template

Student Name	James Butler
Student Number	X20129211
Course	BSHCSD4
Supervisor	Keith Maycock

Month: March

What?

This month I have started trying to use Android studio. It is running very slow and debugging near Impossible. Have found out that I can run what I create on my phone.

So What?

I have managed to get the android studio to run applications on my phone. This is a lot faster than using the unusable built in emulator. Debugging on my phone though has its own issues as it takes about 5 mins every time, I want to run some code. Furthermore, Android studio is very laggy and has been a nightmare to work with.

Android studio is very resource heavy, and I am starting to panic as I am using an old machine and have no means of being able to get a new machine. Android studio is not running well at all.

Now What?

I have decided I am going to talk to my supervisor about this and come up with a solution.

Student Signature	James Butler

Supervision & Reflection Template

Student Name	James Butler
Student Number	X20129211
Course	BSHCSD4
Supervisor	Keith Maycock

Month: April

What?   Talked to supervisor.   Decided to go with NetBeans Application.   Development started on NetBeans java application.
So What?   This month, after talking with my supervisor I have decided to create a java NetBeans application. Android studio just wasn't working for me. I still wanted to use java, so this was a good alternative. The application is a java swing application and progress has been made. A login form and 2 functions have been created.   Development full steam ahead
Now What?   To address outstanding challenges, I need to keep working and working. I cannot stop I must just keep working and power ahead.

Student Signature	James Butler