

John Brown University
CS-3233-01
Roberto Agüero

TABLE OF CONTENTS

Abstract.....	3
1 Introduction (Java).....	4
1.1 Overview.....	4
1.2 Object-oriented Programming.....	4
1.3 The Basic GUI (Graphical User Interface) Application.....	5
1.4 Overview of the Java 2D, 3D & API Concepts.....	6
1.5 Overview of Databases.....	6
2 Modules Description.....	7
2.1 Project Overview.....	7
2.2 User Module.....	8
2.3 LogIn Module.....	8
2.4 Register Module.....	9
2.5 MainFrame Module.....	10
2.6 FoodLog Module.....	11
2.7 Workout Module.....	12
2.8 Profile Module.....	14
3 Design and Development of Project.....	15
3.1 Software Requirements.....	15
3.2 Hardware Requirements.....	15
3.3 Sample Code and Results.....	16
4 Conclusion.....	41
References.....	42

CS-3233-01 Final Project

Design and Implementation of Calorie Tracking Application with Graphical User Interface of Java Swing and Database Connection

Abstract:

A calorie tracker is a simple and intuitive app that helps users monitor their daily caloric intake while providing additional features that help manage their weight. Calorie++ is a Java-based calorie tracking application which meets these key features while maintaining a user-friendly approach. The application uses a MySQL database to store user information, allowing them to track and manage their caloric intake over time. The user interface of Calorie++ is built using the Java Swing framework, which provides a clean and organized experience for the user to navigate. The application allows user to input information about the foods they have eaten, as well as the ability to change their daily caloric goal. One of the key features of Calorie++ is the workout tab which allows users to register their exercise and receive a score based off the variables that affect a workout's effectiveness. This allows user to know if they are making progress and helps them identify areas where they might need to make changes in order to achieve their goals. The purpose of Calorie++ is to make working out and eating healthy accessible for people of all knowledge. By providing users with a convenient and user-friendly way to track their caloric intake, Calorie++ helps users make healthier food choices and maintain a healthy weight.

1 INTRODUCTION TO JAVA

1.1 Overview of Java

Java is a popular programming language that was first implemented in 1995 by James Gosling as a language that “had a familiar C-like notation with greater uniformity and simplicity than C/C++” [Source 1]. In modern day, Java is used for developing a wide range of applications which include web, mobile and desktop applications. It is a high-level language, which means that humans can easily understand it and read/write it. Java is designed to be compiled into machine language that can be executed on various platforms such as Windows, Mac and Linux [Source 2]. This concept is called platform independence, and it is a key characteristic of Java. Platform independence means that “a program written in Java language must run similarly on diverse hardware” [Source 1]. This is obtained by compiling code written in Java into bytecode and then running it on a virtual machine which provides portability to Java [Source 1].

The popularity of Java is attributed to its reliability and security, which is why various enterprise-level applications such as banking systems are developed with it [Source 2]. What gives Java security is found in its design, since Java bytecodes as well as JVM specifications allow for built in mechanisms to verify the security of code written in Java [Source 2]. In addition to these two attributes, Java is also popular because of its extensive library of pre-built classes, which include methods and fields [Source 2]. These can be used rather quickly by developers which allow for a productive coding process for applications because of their built-in functionalities.

1.2 Object Oriented Programming

Object-oriented programming (OOP) is a programming concept that is based on the concept of "objects", which are elements of a class and can contain data and code that manipulates that data [Source 1]. A class works as a blueprint that represents variables and methods that the user defines respective to the purpose of it. The concept of Object-Oriented Programming is also accompanied by its respective pillars, which are abstraction, encapsulation, inheritance and polymorphism [Source 3]. These class pillars provide a baseline for using objects across different projects because they allow objects to have characteristics that connect different ideas through their similarities. Objects are then created from these

classes and consist of different instances, or examples, of the blueprint's important features.

The purpose of object-oriented programming is to make large software projects easier to manage, furthermore improving their quality and the number of time/resources needed for its completion. As previously stated, reusability/adaptability is a strong suit of Java, and OOP demonstrates this. For example, if certain projects were to have similar content/requirements, let's say, in the same company, there can be concepts used such as inheritance and abstract classes to simplify the reusability of the Java code. In this type of situation, objects in Java are seen as "pluggable components" which help the industry take their projects to larger scale without a need to start from the ground up [Source 1].

1.3 The Basic GUI (Graphical User Interface) Application

A GUI, or graphical user interface, is a type of user interface that allows the user of the application to interact with the program through graphical visual indicators or icons instead of text. Usually, they provide an enhancement to programs in order to make the user experience less complex and more appealing to the eye. In Java, a GUI typically consists of components such as windows and buttons which can have different functionalities depending on the purpose of the program. They are made up of visual elements and are used to display information, change the screen and/or receive input from the user. These components can also be arranged depending on the layout chosen by the developer.

To create a simple GUI in Java, developers can implement the Swing library which consists of components such as JFrames and JLabels or the Abstract Window Toolkit (AWT) which also provides ActionEvents and ActionListeners that interact with said components. The AWT API also provides layouts for said GUI such as GridLayout or GridBagLayout as previously mentioned [Source 4]. With combination of other libraries in Java's deep pool, these tools can provide a set up for a solid application.

Apart from Swing and AWT, there is another library called JavaFX which was developed by Chris Oliver as a way to replace Swing [Source 5]. The purpose of JavaFX is to expand on Swing's components while providing additional functionality and speed of GUI application development. Some of these new features that cannot be found in Swing are CSS styling to components allowing developers to furthermore customize their application as well as the Canvas API which allows

users to draw on the JavaFX application [Source 5]. While these tools allow for the development of simple GUI applications, they are not meant to be used for large scale apps.

1.4 Overview of the Java 2D, 3D & API Concepts

An API, or an application programming interface, provides “a set of protocols and definitions for building and integrating application software” [Source 8]. Java has two APIs for graphics, which are Java 2D and Java 3D. The Java 2D API contains components of two-dimensional fields, and provides the user with “imaging capabilities” that can be used through the AWT library [Source 6]. Some of the main functionalities provided by this API are tools for color enhancement which can be used for customization of applications as well as render models for changing the display of an application depending on the device. All these Java 2D objects are based off a coordinate plane called “user space” in order to meet the constraints of the application [Source 6].

Java 3D is an API similar to Java 2D, but instead it enables the developer to create three-dimensional graphics inside an application. One of the most important features of Java 3D is that it has a high performance due to its high speed of rendering for the visible geometry [Source 9]. It is not as commonly used when building GUI applications as Java 2D because of the limitations of a GUI, but it can enhance any application by providing a more visual approach. One of the main goals of Java 3D was to provide programmers with the opportunity to develop a high-level OOP program consistently with the use of its modern components that provide high quality graphics.

1.5 Overview of Databases

In programming, databases are an organized collection of data/information that allows for an easy storage. This data can be stored either electronically via a computer system, or in a program through data structures that the developer defines [Source 7]. The information stored in databases is made accessible through a program that calls queries depending on said information trying to be obtained [Source 7]. There are various kinds of databases, but the most popular of these when referring to programming are relational databases.

A majority of modern databases are run off of SQL, which stands for Structured Query Language. SQL provides data manipulation statements as well as ways to either provide access control or define said data [Source 7]. Some current

problems faced by databases are that there are various security issues whenever the access to these databases are provided to a larger number of users. Data breaches are very common these days and with such important information being stored companies cannot run the risk of getting their data leaked (Source 7). Additionally, the maintenance and management of these databases can get difficult because the software is constantly being updated as well as the data volume being constantly growing.

2 MODULES DESCRIPTION

2.1 Project Overview

Calorie++ is a GUI-based calorie tracker that implements Swing and AWT for its full design. The application consists of a log in system, a main menu, and three sub menus for each intended activity. These can be divided into the User, login, register, mainframe, FoodLog, workout and profile module. A majority of the components used in this project are JLabels, JTextFields, JFrames, JPanels and JButtons. The theme colors of this application are white and green, which a few shades of gray and yellow to add contrast. There was going to be an attempt of a remodel with JavaFX as the main library for the project, but the decision was made to proceed with Swing and AWT for comfort.

The purpose of this project is to provide exercise beginners with a platform to learn about their health. Many of the modern calorie trackers such as MyFitnessPal have complicated user interfaces and new people might get lost because they don't know terms. With Calorie++, this was kept in mind when designing the interface which is why majority of the modules are so simple. The whole idea with this application is to introduce people that want to eat well/exercise through a simple app which they can just log in and use for a short time each day. While it might be very simple, the complexity of this application could not be expanded on because of time limitations as well as limitations from the libraries being used. While Swing and AWT are consistent and reliable, they are a little updated and do not provide as many modern features as tools such as .NET.

2.2 User Module

The User module is a very simple class which provides the variables that each user will have whenever their account is created. It consists of string variables that represent the user's name, email, phone number, password, and address. These

variables are made to replicate those present in the database, so that each user can be identified properly and routed to their login without any issues. This module serves as the way to connect both the LogIn module as well as the Register module with the database. This module consists of no methods because the JDBC kit provides a way to modify the class variables without a necessity for user-defined functions.

2.3 LogIn Module

The first functional module of Calorie++ is the login form that the user will meet when the application is first run. This module imports all the Java AWT and Swing elements as well as the JDBC kit in order to connect user information with the database. This module consists of a very simple window with two JTextFields for the user to enter their email address and password with an option to both log in or register, which are both JButtons. The log in JButton has an ActionListener function so that whenever the user enters their information and clicks to log in, it will automatically create a User object and check if they are in the database. If the user is registered in the database, it will take them to the main frame. If the user is not registered in the database, it will display a message to try again as the login was invalid.

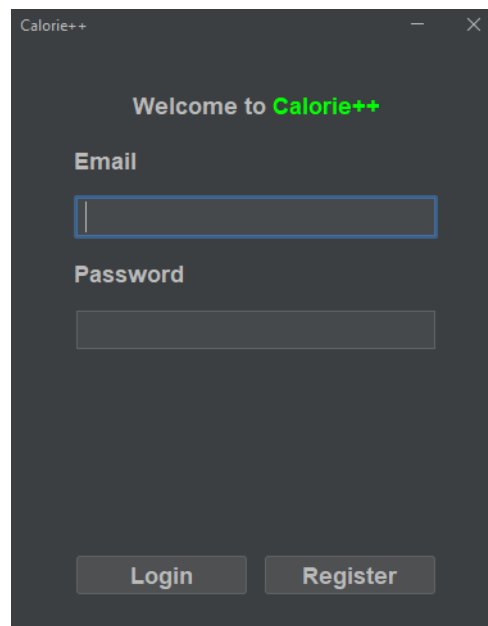


Figure 1 - LogIn Module

The function for user authentication is called `getAuthenticatedUser()`, which creates a connection to the database and then combines a prepared statement with

the query **"SELECT * FROM users WHERE email=? AND password=?"** to then create a ResultSet that gets the values from the database and assigns it to the private variables of the User class (such as name, email, etc.). If the connection fails, the function will return and tell the user that a connection to the database was unsuccessful. Apart from this function, the registration button will lead the user to its own window, which is part of the Register module.

2.4 Register Module

Similar to the LogIn module, the Register module imports the SQL library in order to use the JDBC kit for the registration of users. The user can access this module without being logged in, just by the simple click of the “register” button. Whenever the window for this module loads up, there will be a collection of JLabels and JTextFields that asks the user their information. It will ask for all of the User module variables, which range from name to email to password. The layout of this module is determined by the GridLayout which uses parameters of 0 rows and 1 column in order to give the window the shape of a typical registration form.

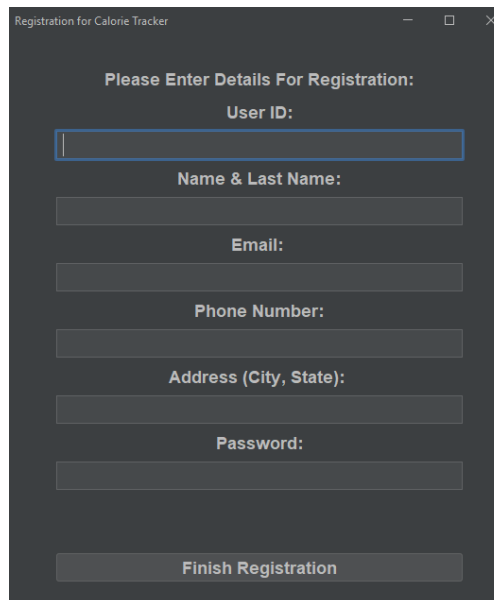


Figure 2 - Register Module

What makes this module complex is that once the user has entered all their details and they click the “finish registration” button, a new user in the database must be added to the table. To achieve this, a function called `getRegistered()` is used and resembles the `getAuthenticatedUser()` from the LogIn module. A connection to the database is created at the beginning of the function, and the query **"INSERT INTO users (id, name, email, phone, address, password) VALUES (?, ?, ?, ?, ?, ?)"** is

used in combination with a PreparedStatement to set these values exactly as the ones that were entered into the JTextFields by the user. The PreparedStatement then executes an update to the values in the table, and the user is registered in the database. After the user registers themselves, they are immediately sent back to the login page where they can use their newly obtained credentials to access the application.

2.5 MainFrame Module

The MainFrame module is the “home page” or lobby of the application. It consists of three different panels organized in a column format to provide the user a clear view of which feature of the application that they want to access. Panel 1 provides the “Log Food” feature, which leads the user to the FoodLog module and serves as the main feature of Calorie++. Panel 2 gives the user access to the “Log Workout” section, which will point them in the direction of the Workout module. The final panel will lead the user to their profile/settings page, which is part of the Profile module. All these panels have a JButton that leads to their respective module.

ImageIcons and JLabels are used in conjunction to display a small logo or image that represents each of the sections available in the user. This provides a simple experience for users that maybe cannot read, or simply do not understand certain terminology. This module is fairly simple since it is the “center point” of the whole GUI application. Each of the modules that they can access through the mainframe provides a way to come back in case the user wants to check out a different feature, the same way that a full-fledged application would.

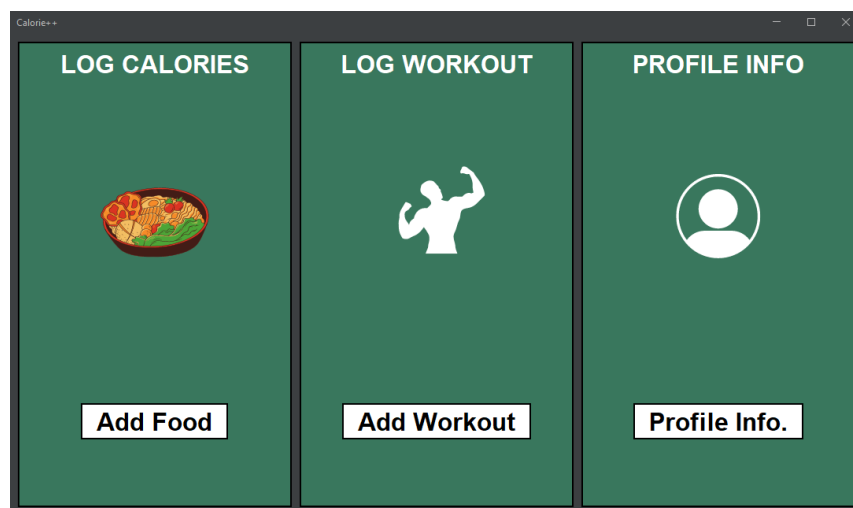


Figure 3 - MainFrame Module

2.6 FoodLog Module

The FoodLog module consists of a main panel split into two halves vertically in order for ease of access for the user. The left side of the window is used for the user to enter their calories consumed in breakfast, lunch, dinner and snacks. JTextFields are used in each line to retrieve this information from the user. There is also a “log” button for the user to click whenever they are done entering their total calories. The total will be increased no matter how many times the user has clicked the button to prevent any limitations. The right panel consists of an upper and lower section to divide the functionality. The upper panel displays the total calories consumed by the user, with an initial value of 0/2000. It also displays a message which tells the user how many calories they are missing depending on their amount entered. The bottom panel has a JTextField and 3 JButtons that all alter the upper panel. The first JButton, which says “Change Calories Goal” is combined with the JTextField as a way for the user to change their calorie goal. Once clicked, the number will immediately change. The second button acts as a reset for the total calories consumed in the case of a mistake made by the user. The final JButton provides the user a return button to “home” or the mainframe module.

Figure 4 - FoodLog Module

The way that the total calories are estimated are result of a function called `calcTotCals()`, which takes the input from the user in the JTextFields and adds it up to a global variable, `total`, to then display in the right panel. The formula for this function is as follows:

$$totalCals += (breakfastCals + lunchCals + dinnerCals + snackCals)$$

Equation 1 - Total Calories Equation

The `calcTotCals()` function is called whenever the “log” button is clicked by the user. This button uses an `ActionListener` as well as the previous function. Depending on the number of calories entered by the user, the information label in the right panel will be updated. If the total calories are above the calorie goal, the label will turn red to show that the user is going over their limit and that it is a bad idea. If the total calories are above the halfway goal, the label will turn orange to show the user that they are almost to their goal. Otherwise, the label will remain green to give the user a sense of safety and motivation to the calorie goal.

2.7 Workout Module

The purpose of the Workout Module is to allow users to enter the time, intensity and category of their workout in order to produce a score that is generated by a formula taking all these factors into account. The layout of the module is produced by a table in a `GridBag` layout setting, which can be expanded by the click of a button. Three `JTextFields` gather the user information and they are given the option to choose up to 6 different workouts in the same day.

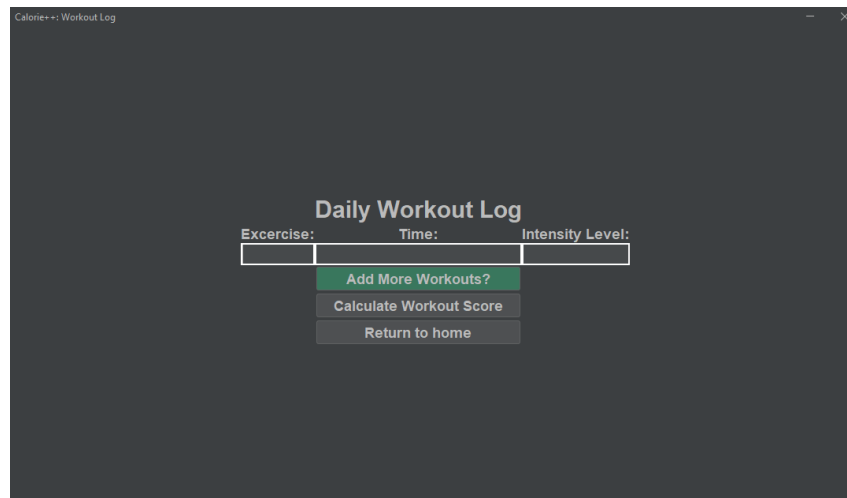


Figure 5 - Workout Module

In order to expand the table, a `createRow()` function is bound to the click of a button. Every time the button is clicked, three exact same copies of the `JTextFields` are created. Then, the constraint functions `gridx()` and `gridy()` are used to position the new instances of these `JTextFields` right below the current ones. Since each of these are bounded to a button, the measurements must be exact so that the layout is not destroyed by an excessive amount of button clicks. The exact syntax and functionality of this function will be expanded on in the design section of this report.

The JButton labeled “Calculate Workout Score” is clicked by the user after they have entered the number of workouts they want to, and it calls the displayScore() function. This extensive function creates a new window to display a JLabel that contains the workout score. The workout score is calculated by the following equation:

$$WOScore = ((exerciseScore + intensityScore + timeScore)/3) * 10$$

Equation 2 - Workout Score Calculator

The way that this equation calculates the score is that it will take all 3 scores from their respective text field, and assign it a percentage based on the deserving score. For example, a Run is given an exercise score of 8, because it is usually a high demanding exercise. The intensity score will remain the same as the user enters it, but it will be capped at 10. Finally, the time score will be relative to the user, so if they worked out for more than an hour, they would have a score of 10. When these variables are plugged into the equation, it would produce a total workout score of 93, which indicates that the workout was good for the user. More of the code will be expanded on in the design section, but this equation allows for an accurate representation of improvement/decline based on the user’s choice for exercise.

2.8 Profile Module

The profile module is the last of the three accessible modules from the mainframe. It consists of two different panels that both use a GridLayout to portray the information in a neat manner. This module contains the user information, all depending on the account that they use to log in. It shows their full name as well as their account level, which increases depending on the amount of time they have used the application for. This module gives the user the option to add a profile picture to their account, which uses a JFileChooser. Additionally, through this module the user can change the theme of the whole application, but due to time limitations this feature is very simple and only works with the mainframe. The user can also then change the font of the entire program as well as an option to “return home” or go back to the mainframe. Finally, the sign out button allows the user to go back to the login module in case they want to exit the application or enter with another account.

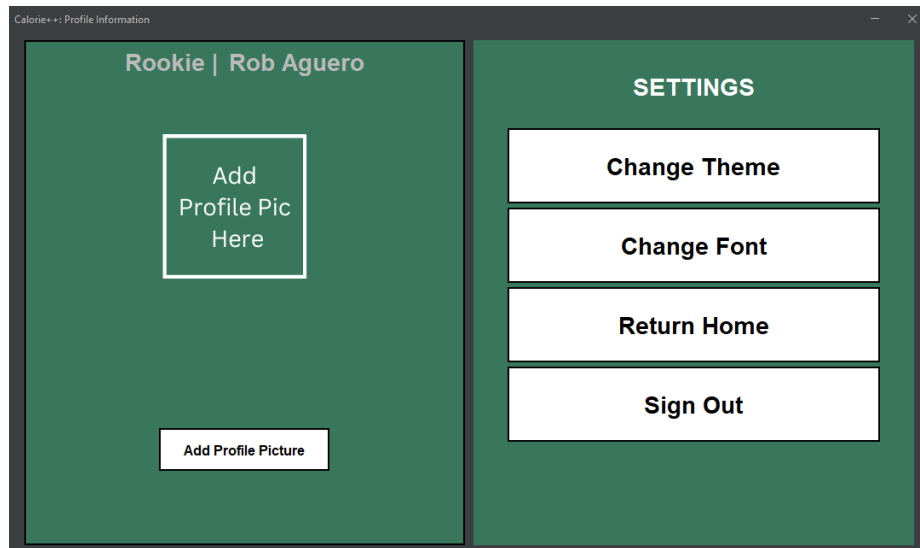


Figure 6 - Profile Module

In order for the JLabel with the image to be changed, it requires the use of the JFileChooser element. The full code for this function will be found in the design section, but the basic process is an ActionListener on the “Add Profile Picture” JButton. Whenever the button is clicked, a file choosing option will be prompted and the ImageIcon of the current label will be replaced by whichever image the user chooses. It only takes in .PNG formatted images due to constraints of the JLabel’s size. Another important function in this module is the code to obtain the name of the user. A function getUsername() is used and creates a connection to the database, from which the simple query of **"SELECT name FROM users"** is used in conjunction with a PreparedStatement to get the information. The code for this function can also be found in the design section.

3 DESIGN AND DEVELOPMENT OF PROJECT

3.1 Software Requirements

Calorie++ was developed with OpenJDK version 11.0.12. The initial versions of this project were started in Eclipse IDE for Java Developers (2022-12), but because of problems with the accessibility of .jar files, it was moved to Visual Studio Code (Version 1.74). With the use of an Extension Pack for Java (v0.25.7) the switch of IDEs was made accessible and also provided certain tools that help a developer when creating programs with Java. Additionally, the use of MySQL connector .jar files are needed for full functionality of application.

3.2 Hardware Requirements

For this project, there are no hardware requirements since the GUI application can run in 99.9% of devices. Java Swing and AWT libraries are really old, and the code is pretty lightweight so even on old devices, the run speed of the application is not affected. The only requirement really would be a computer whether it be a laptop or a desktop because that is the only capable device of running this program.

3.3 Sample Code

Since this full-fledged application uses a handful of modules, a lot of code such as JLabels and JTextFields are fairly similar, thus it would be illogical to include every single line of code in the report. Due to the size of each file, only certain significant snippets or common blocks of code will be included. Most functions that either connect to the database or provide a feature other than just changing windows will have its own block of code, as previously mentioned in Section 2. Additionally, images of the database connected to this application will be provided to show how the login/registration process works and saves the user information.

Functions/Samples of code in this section:

- LogInButton + getAuthenticatedUser()
- getRegistered() + example of registration
- MainFrame images + button color customization
- calcTotCals() + food log button + calorie reset + calorie goal overwriting
- createRow() + displayScore()
- Profile picture changer + user level/name getter + change theme

LogIn Button Code + Function: This button is what the user clicks whenever they finish entering their information. It has an ActionListener so that whenever it gets clicked it checks with the getAuthenticatedUser() function as well as a MouseListener for visual clarity. More explanation on the getAuthenticatedUser() function can be found below.

```

JButton logInButton = new JButton("Login");
    logInButton.setFont(programFont);
    logInButton.addMouseListener(new java.awt.event.MouseAdapter()
{
    public void mouseEntered(java.awt.event.MouseEvent evt) {

```

```

        logInButton.setBackground(myColor1);
        logInButton.setForeground(Color.white);
    }

    public void mouseExited(java.awt.event.MouseEvent evt) {
        logInButton.setBackground(new
JButton().getBackground());
    }
});
logInButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //Gets email and password from user input
        String email = emailTF.getText();
        String password =
String.valueOf(passwPF.getPassword());
        User user = getAuthenticatedUser(email, password);
//checks if the user is registered
        if (user != null) //If user is registered in database,
go to application
        {
            MainFrame mainFrame = new MainFrame();
            mainFrame.initialize(user);
            dispose();
        }
        else //If not registered, display error mesage
        {
            JOptionPane.showMessageDialog(GUI.this,
                "Email or Password Invalid",

```



```

        "Please Try Again",
        JOptionPane.ERROR_MESSAGE);
    }
}
});

```

getAuthenticatedUser(): Function for verification that user is found in database before logging in. This function takes in two parameters, which are the user's email and password obtained from JTextFields. If they are found, the user will then be routed to the mainframe and proceed from there.

```

private User getAuthenticatedUser(String email, String password) {
    User user = null;
    //Connect to Database
    final String DB_URL =
"jdbc:mysql://localhost/calorieTracker?serverTimezone=UTC";
    final String USERNAME = "root";
    final String PASSWORD = "";

    try{
        Connection conn = DriverManager.getConnection(DB_URL,
USERNAME, PASSWORD); //Creates connection between program and database
        // Connected to database successfully...

        String sql = "SELECT * FROM users WHERE email=? AND
password=?";

        PreparedStatement preparedStatement =
conn.prepareStatement(sql);
        preparedStatement.setString(1, email);
        preparedStatement.setString(2, password);

        ResultSet resultSet = preparedStatement.executeQuery();
    }
}

```

```

        if (resultSet.next()) { //Here the values for the Module
User.JAVA are being taken from the Database

            user = new User();
            user.name = resultSet.getString("name");
            user.email = resultSet.getString("email");
            user.phone = resultSet.getString("phone");
            user.address = resultSet.getString("address");
            user.password = resultSet.getString("password");
        }
        preparedStatement.close();
        conn.close();
    }catch(Exception e){
        System.out.println("Database connection failed!");
    }

    return user;
}

```

getRegistered(): This function is found in the Register module, and it allows the user to enter all their information to be entered into the table in the database. Screenshots providing outcomes are shown below.

```

private User getRegistered(String userID, String name, String email,
String phone, String address, String password){

    //Use all information to put the user into database.

    //After registration, if the user is in the database, go back
to login page.

    User user = null;

    //Connect to Database

    final String DB_URL =
"jdbc:mysql://localhost/calorieTracker?serverTimezone=UTC";

    final String USERNAME = "root";

```

```

final String PASSWORD = "";

try{
    Connection conn = DriverManager.getConnection(DB_URL,
    USERNAME, PASSWORD); //Creates connection between program and database
    // Connected to database successfully...
    //System.out.println("Connected.");

    String sql = "INSERT INTO users (id, name, email, phone,
    address, password) VALUES (?, ?, ?, ?, ?, ?)";

    PreparedStatement preparedStatement =
    conn.prepareStatement(sql);

    preparedStatement.setString(1, userID);
    preparedStatement.setString(2, name);
    preparedStatement.setString(3, email);
    preparedStatement.setString(4, phone);
    preparedStatement.setString(5, address);
    preparedStatement.setString(6, password);

    preparedStatement.executeUpdate();

    /*

    ResultSet resultSet = preparedStatement.executeQuery();

    if (resultSet.next()) { //Here the values for the Module
    User.JAVA are being taken from the Database

        user = new User();
        user.name = resultSet.getString("name");
        user.email = resultSet.getString("email");
        user.phone = resultSet.getString("phone");
        user.address = resultSet.getString("address");
        user.password = resultSet.getString("password");
    }
    }

```

```

    }
    */
    preparedStatement.close();
    conn.close();
}
catch(Exception e){
    System.out.println("Database connexion failed!");
}
return user;
}

```

Example of Registration:

Registration for Calorie Tracker

Please Enter Details For Registration:

User ID:
3

Name & Last Name:
Roberto Agüero

Email:
rcaguero@outlook.com

Phone Number:
+1 479-220-0680

Address (City, State):
Siloam Springs, AR

Password:
.....

Finish Registration

Figure 7 - Registration Form Filling

After the user clicks “finish registration” button, information is added into database.

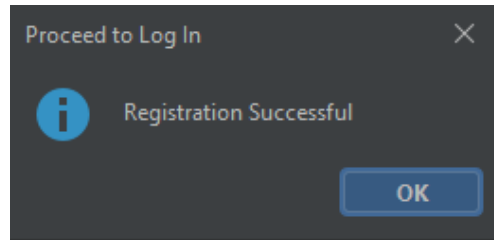


Figure 8 - Message Post-Button Click

☐ Show all

Number of rows: 25

Filter rows: Search this table

Sort by key: None

Extra options

	id	name	email	phone	address	password
<input type="checkbox"/> Edit Copy Delete	1	Rob Aguero	raguero@gmail.com	+14792200680	Siloam Springs, AR	Blue251525
<input type="checkbox"/> Edit Copy Delete	3	Roberto Aguero	rcaguero@outlook.com	+1 479-220-0680	Siloam Springs, AR	Blue251525

Figure 9 - Database Table post-Registration

MainFrame Image Icons: Adding images into a GUI application is not the simplest process. To achieve this, JLabels with ImageIcon were used as well as source images in another folder. The screenshots of the output of the mainframe can be found in Figure 3.

```

JLabel img1 = new JLabel();

ImageIcon imgThisImg = new
ImageIcon("C:/Users/rcagu/Desktop/Java
Projects/LogInForm/src/Images/food2.png");

img1.setIcon(imgThisImg);

img1.setPreferredSize(new Dimension(500,400));

JLabel img2 = new JLabel();

ImageIcon imgThisImg2 = new
ImageIcon("C:/Users/rcagu/Desktop/Java
Projects/LogInForm/src/Images/workout12.png");

img2.setIcon(imgThisImg2);

img2.setPreferredSize(new Dimension(500,400));

```

```

JLabel img3 = new JLabel();

ImageIcon imgThisImg3 = new
ImageIcon("C:/Users/rcagu/Desktop/Java
Projects/LogInForm/src/Images/profile7.png");

img3.setIcon(imgThisImg3);

img3.setPreferredSize(new Dimension(500,400));

```

MainFrame Button Color Animation: This is a simple animation created by the use of ActionListeners (more specifically a MouseListener) so that every time the user hovers over a button with the mouse, it will change color for better accessibility.

```

PFB.addMouseListener(new java.awt.event.MouseAdapter() {

    public void mouseEntered(java.awt.event.MouseEvent evt) {

        PFB.setBackground(myColor3);

        PFB.setForeground(Color.BLACK);

    }

    public void mouseExited(java.awt.event.MouseEvent evt) {

        PFB.setBackground(Color.white);

        PFB.setForeground(Color.black);

    }

});

```

Example of MouseListener:

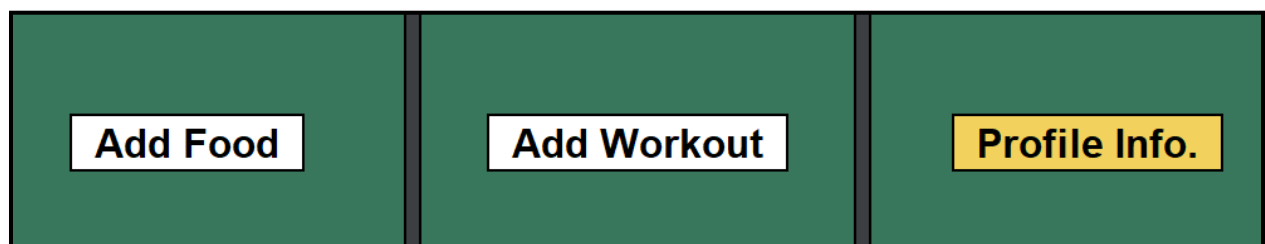


Figure 10 - Highlighted Button with Color Change

calcTotCals(): This function calculates the total number of calories that the user has confused whenever it is clicked. It uses Equation 1, as previously listed in the modules section.

```
public int calcTotCals(String breakfast, String lunch, String dinner,
String snacks){
    int total = 0;
    int bk,lc,dn,sn;
    bk = Integer.parseInt(breakfast);
    lc = Integer.parseInt(lunch);
    dn = Integer.parseInt(dinner);
    sn = Integer.parseInt(snacks);
    total += (bk + lc + dn + sn);
    return total;
}
```

Log Food Button: Whenever this button is clicked, it invokes the calcTotCals() function as well as changes the JLabels to match the information entered by the users.

```
JButton log = new JButton("Log Food");
log.setFont(programFont);
log.setBackground(myColor1);
log.setForeground(Color.white);
sl1.add(log);
log.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //This button will log all values from textfields into
a function that calculates total calories.
        String brkCal = bkf.getText();
        String lcal = lnf.getText();
```

```

String dcal = dnf.getText();
String scal = snf.getText();
totalCals += calcTotCals(brkCal, lcal, dcal, scal);
if(totalCals > calGoal){
    calInf.setText(totalCals + "/" + calGoal);
    calInf.setForeground(Color.red);
    calMot.setText("You are " + (calGoal - totalCals)
+ " calories above from your daily goal.");
}
else if(totalCals < calGoal && totalCals > calGoal/2){
    calInf.setText(totalCals + "/" + calGoal);
    calInf.setForeground(Color.orange);
    calMot.setText("You are " + (calGoal - totalCals)
+ " calories away from your daily goal.");
}
else{
    calInf.setText(totalCals + "/" + calGoal);
    calMot.setText("You are " + (calGoal - totalCals)
+ " calories away from your daily goal.");
}
bkf.setText("");
lfn.setText("");
dnf.setText("");
snf.setText("");
}
});

```

Example of Button functionality:

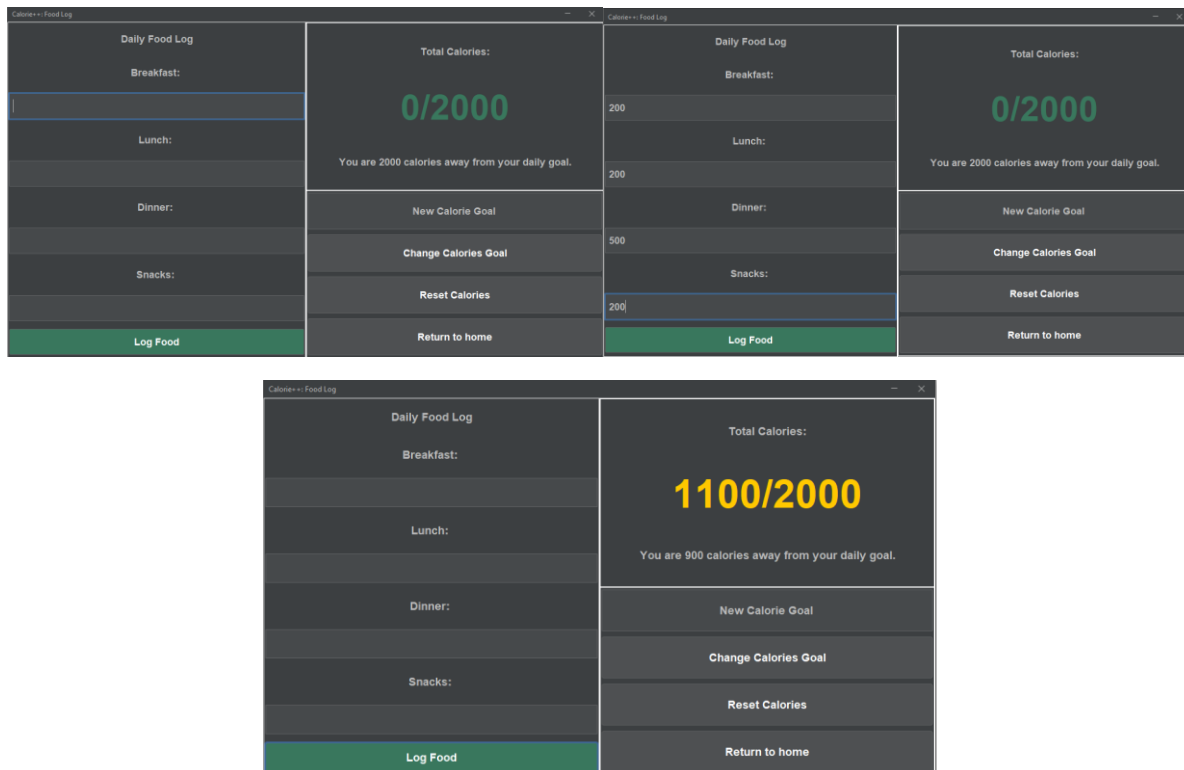


Figure 11 - After logging information

Calorie Reset: This button resets the total calories entered by the user. It uses a simple ActionListener to change the value as the button is clicked.

```

JButton reset = new JButton(" Reset Calories ");
reset.setFont(programFont);
//reset.setBackground(myColor1);
//reset.setForeground(Color.white);
reset.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        totalCals = 0;
        calInf.setText(totalCals + "/" + calGoal);
        calInf.setForeground(myColor1);
        calMot.setText("You are " + (calGoal - totalCals) + "
calories away from your daily goal.");
    }
}

```

```

    });

    reset.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseEntered(java.awt.event.MouseEvent evt) {
            reset.setBackground(myColor1);
            reset.setForeground(Color.white);
        }

        public void mouseExited(java.awt.event.MouseEvent evt) {
            reset.setBackground(new JButton().getBackground());
            reset.setForeground(Color.WHITE);
        }
    });

```

Calorie Goal change: The initial calorie goal is 2000. In case the user wants to change this value, a JTextField and a JButton are linked. Whenever the button is clicked, the value in the JTextField will be displayed as the new caloric goal.

```

JTextField newCals = new JTextField("New Calorie Goal");
newCals.setFont(programFont);
newCals.setHorizontalAlignment(JTextField.CENTER);
lower.add(newCals);
newCals.addFocusListener(new FocusListener() {
    public void focusGained(FocusEvent e) {
        newCals.setText("");
    }

    public void focusLost(FocusEvent e) {
        // nothing
    }
});

```

```

        JButton changeCals = new JButton(" Change Calories Goal ");
        changeCals.setFont(programFont);
        //changeCals.setBackground(Color.white);
        changeCals.setForeground(Color.WHITE);
        changeCals.addMouseListener(new java.awt.event.MouseAdapter()
{
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                changeCals.setBackground(myColor1);
                changeCals.setForeground(Color.white);
            }

            public void mouseExited(java.awt.event.MouseEvent evt) {
                changeCals.setBackground(new
JButton().getBackground());
                changeCals.setForeground(Color.WHITE);
            }
        });
        changeCals.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int CalNumsNew = Integer.parseInt(newCals.getText());
                calGoal = CalNumsNew;
                calInf.setText(totalCals + "/" + calGoal);
            }
        });
    });

```

Example of Calorie Change:

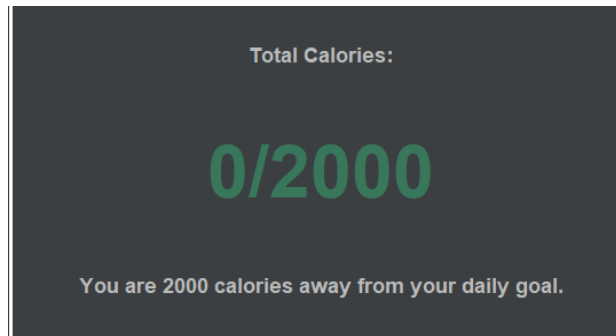


Figure 12 - Before change

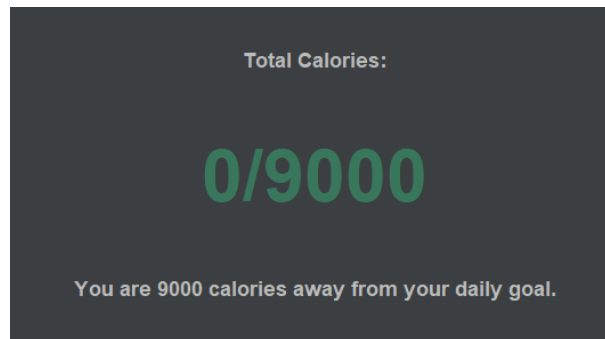


Figure 13 - After change

createRow(): This function creates a new row in the Workout module table each time the button is clicked. It uses `gridx()` and `gridy()` functions with accurate parameters to place each new row of the table right below the previous one. It creates the three new textfields every time in order to not run into any issues.

```
public void createRow(GridBagConstraints c){
    JLabel ex = new JLabel("Excercise:", SwingConstants.LEFT);
    ex.setFont(programFont);
    JTextField extf = new JTextField();
    JTextField ttft = new JTextField();
    JTextField ltft = new JTextField();
    extf.setBorder(new LineBorder(Color.WHITE, 2));
    extf.setFont(programFont);

    JLabel time = new JLabel("Time:", SwingConstants.CENTER);
    time.setFont(programFont);
```

```

    ttf.setBorder(new LineBorder(Color.WHITE, 2));
    ttf.setFont(programFont);

    JLabel level = new JLabel("Intensity Level:",
SwingConstants.CENTER);
    level.setFont(programFont);
    ltf.setBorder(new LineBorder(Color.WHITE, 2));
    ltf.setFont(programFont);

    c.gridx = lbX;
    c.gridy = lbY;
    main.add(ex, c);
    c.gridx = lbX;
    c.gridy = tfY;

    lbX++;

    main.add(extf, c);
    c.gridx = lbX;
    c.gridy = lbY;
    main.add(time, c);
    c.gridx = lbX;
    c.gridy = tfY;

    lbX++;

    main.add(ttf, c);
    c.gridx = lbX;

```

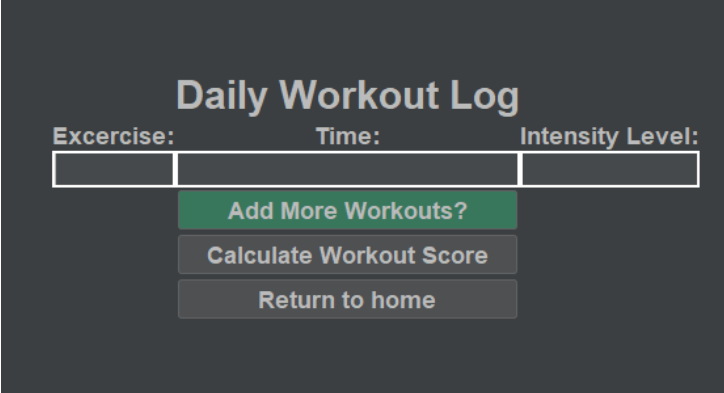
```

c.gridy = lbY;
main.add(level, c);
c.gridx = lbX;
c.gridy = tfY;
main.add(ltf, c);

// 1 2 3 (0, 0, 0)
// 1 2 3 (1, 1, 1)
lbY++;
tfY++;
System.out.println("Added row.");
setVisible(true);
}

```

Example of new rows:



The screenshot shows a dark-themed application window titled "Daily Workout Log". At the top, there are three labels: "Exercise:", "Time:", and "Intensity Level:". Below each label is a white rectangular input field. Underneath the input fields, there are three buttons stacked vertically. The top button is green and labeled "Add More Workouts?". The middle button is grey and labeled "Calculate Workout Score". The bottom button is grey and labeled "Return to home".

Figure 14 - Module before button click

Daily Workout Log

Exercise:	Time:	Intensity Level:

Add More Workouts?

Calculate Workout Score

Return to home

Figure 15 - Module after 6 button clicks

displayScore(): This function creates a completely new panel where the score is calculated based off variables such as intensity and exercise type. A lot of if statements are used to determine the user's input as well as Equation 2 to get the final score.

```
public void displayScore(JTextField extf, JTextField ltf, JTextField ttf){
    double score;
    main.setVisible(false);
    JPanel newPan = new JPanel();
    newPan.setLayout(new GridBagLayout());

    GridBagConstraints gbc = new GridBagConstraints();
    gbc.gridwidth = GridBagConstraints.REMAINDER;
    gbc.anchor = GridBagConstraints.NORTH;

    JLabel title2 = new JLabel("Based of your workouts, your Total
    Workout Score is... ", SwingConstants.CENTER);
    title2.setFont(programFont);
    newPan.add(title2 ,gbc);
```

```

gbc.anchor = GridBagConstraints.CENTER;
gbc.fill = GridBagConstraints.HORIZONTAL;

//Score formula = (intensity * 0.5) + (excercise * 0.3) +
(time * 0.2)

String excercise = extf.getText().toLowerCase();
int exScore = 0;

if(excercise.equals("cardio") || excercise.equals("run") ||
excercise.equals("running")){
    exScore = 8;
}

else if(excercise.equals("lift") ||
excercise.equals("lifting") || excercise.equals("weights")){
    exScore = 4;
}

else if(excercise.equals("walk") || excercise.equals("jog") ||
excercise.equals("treadmill")){
    exScore = 5;
}

else if(excercise.equals("soccer") ||
excercise.equals("basketball") || excercise.equals("weights")){
    exScore = 9;
}

else if(excercise.equals("hiking") ||
excercise.equals("dancing") || excercise.equals("pilates")){
    exScore = 6;
}

else{
    exScore = 2;
}

```



```
}

int intScore = 0;
int intensity = Integer.parseInt(ltf.getText());
if(intensity >= 10){
    intScore = 10;
}
else if(intensity <= 0){
    intScore = 1;
}
else{
    intScore = intensity;
}

int time = Integer.parseInt(ttf.getText().substring(0,2));
int timeScore = 0;
if(time >= 60){
    timeScore = 10;
}
else if(time > 40 && time < 59){
    timeScore = 8;
}
else if(time > 20 && time < 39){
    timeScore = 5;
}
else{
    timeScore = 3;
}
```

```

score = ((exScore + intScore + timeScore)/3) * 10;

JLabel calScore = new JLabel("Score: " + score,
SwingConstants.CENTER);
calScore.setFont(calFont);
if(score < 30){
    calScore.setForeground(myColor2);
}
else if(score > 30 && score < 60){
    calScore.setForeground(myColor4);
}
else if(score > 60 && score < 90){
    calScore.setForeground(myColor3);
}
else{
    calScore.setForeground(myColor1);
}
newPan.add(calScore, gbc);
JButton but4 = new JButton("Return");
but4.setFont(programFont);
but4.setBackground(myColor1);
but4.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Workout ws1 = new Workout();
        ws1.initialize();
        dispose();
    }
});

```

```

    }

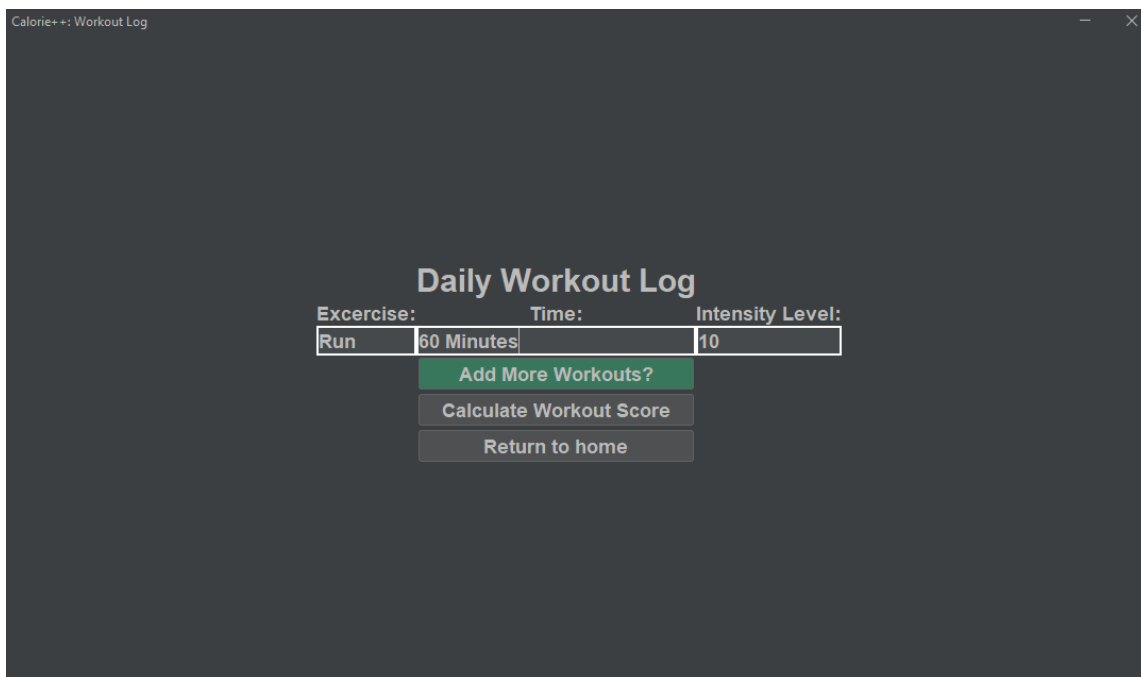
});

newPan.add(but4, gbc);

add(newPan);
}

```

Example of displayScore():

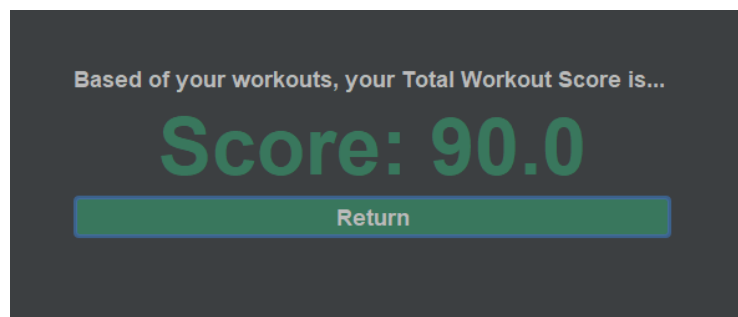


Calorie++ Workout Log

Daily Workout Log

Exercise:	Time:	Intensity Level:
Run	60 Minutes	10

Figure 16 - User details before score calculation



Based of your workouts, your Total Workout Score is...

Score: 90.0

Figure 17 - Portrayed Score

Add Profile Picture: Through this button, the user can change their profile picture using an ActionListener and a JFileChooser. It is fairly simple to replicate this process, but the file must be .png and the scaling of the image is hard for it to be high quality.

```

JButton but1 = new JButton(" Add Profile Picture ");
    but1.setPreferredSize(new Dimension(200, 50));
    but1.setBorder(new LineBorder(Color.BLACK, 2));
    but1.setFont(programFont);
    but1.setBackground(Color.WHITE);
    but1.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseEntered(java.awt.event.MouseEvent evt) {
            but1.setBackground(myColor3);
            but1.setForeground(Color.BLACK);
        }

        public void mouseExited(java.awt.event.MouseEvent evt) {
            but1.setBackground(Color.white);
            but1.setForeground(Color.black);
        }
    });
    but1.setForeground(Color.BLACK);
    but1.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            FileNameExtensionFilter filter = new
FileNameExtensionFilter("JPEG File", "jpg", "jpeg");
            jf1.setFileFilter(filter);
            int response = jf1.showOpenDialog(null);
            try{

```

```

        if (response == JFileChooser.APPROVE_OPTION){
            String name = jf1.getSelectedFile().getPath();
            String message = name + " will be used as the
profile picture.";

            JOptionPane.showMessageDialog(null, message);

            ImageIcon new1 = new ImageIcon(new
ImageIcon(name).getImage().getScaledInstance(300, 300,
Image.SCALE_DEFAULT));

            img1.setIcon(new1);
        }
        else{
            JOptionPane.showMessageDialog(null, "Feel free
to look for a profile picture later! :)");
        }
    } catch(Exception e2){
        e2.printStackTrace();
    }
}

});

```

Example of Profile Picture:

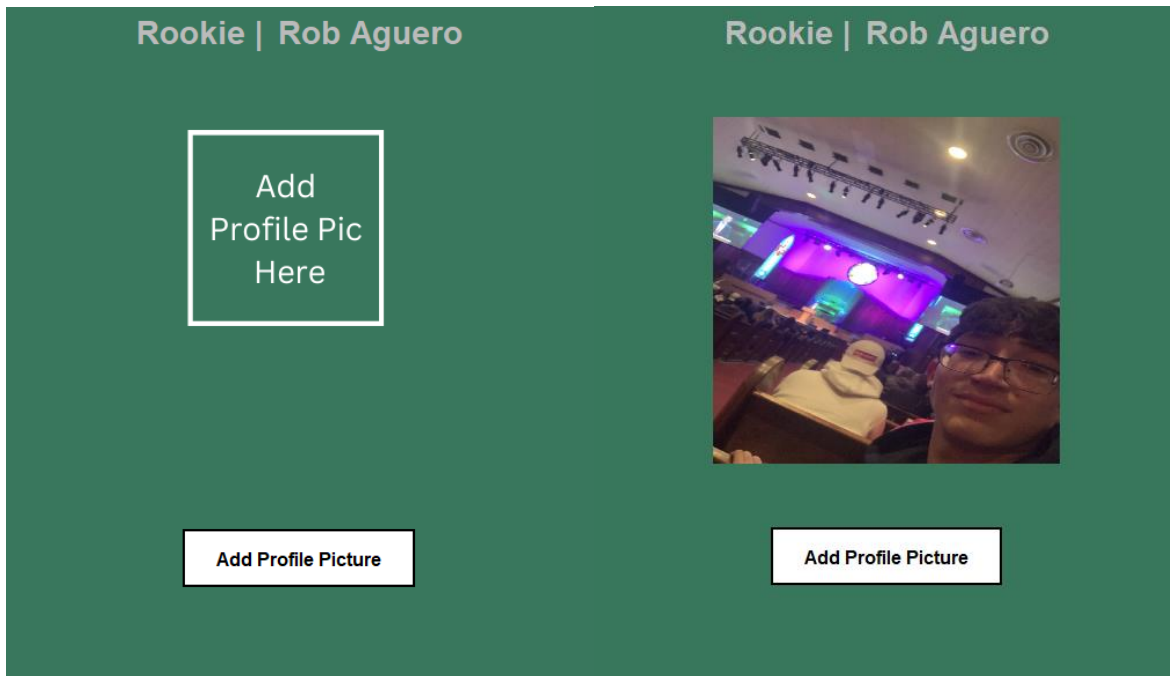


Figure 18 - Profile Picture pre-and post-selection

getUserName(): This function returns the user's name to display on their profile. It creates a connection with the database and uses a simple query as previously listed in the module description to retrieve the information.

```
private String getUserName(){
    User user = null;
    //Connect to Database
    final String DB_URL =
"jdbc:mysql://localhost/calorieTracker?serverTimezone=UTC";
    final String USERNAME = "root";
    final String PASSWORD = "";

    try{
        Connection conn = DriverManager.getConnection(DB_URL,
USERNAME, PASSWORD); //Creates connection between program and database
        // Connected to database successfully...

        String sql = "SELECT name FROM users";
```

```

        PreparedStatement preparedStatement =
conn.prepareStatement(sql);

        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) { //Here the values for the Module
User.JAVA are being taken from the Database
            user = new User();
            user.name = resultSet.getString("name");
            //System.out.println(user.name);
        }
        preparedStatement.close();
        conn.close();
    }catch(Exception e){
        System.out.println("Database connexion failed!");
    }

    return user.name;
}

```

changeTheme: This JButton allows the user to customize the application by changing the entire theme color of the app. It resembles a night mode except that it makes the color pink instead of green. The function does not work for all the application because of Swing restrictions but its functional for the mainframe.

```

Color myColor3 = new Color(243, 214, 106);
JButton changeColor = new JButton(" Change Theme ");
changeColor.setBorder(new LineBorder(Color.BLACK, 2));
changeColor.setFont(programFont2);
changeColor.setBackground(Color.WHITE);
changeColor.setForeground(Color.BLACK);
changeColor.addMouseListener(new java.awt.event.MouseAdapter()
{

```

```

        public void mouseEntered(java.awt.event.MouseEvent evt) {
            changeColor.setBackground(myColor3);
            changeColor.setForeground(Color.BLACK);
        }

        public void mouseExited(java.awt.event.MouseEvent evt) {
            changeColor.setBackground(Color.white);
            changeColor.setForeground(Color.black);
        }
    });

    changeColor.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            MainFrame m1 = new MainFrame();
            Workout w1 = new Workout();
            Profile p1 = new Profile();
            FoodLog f1 = new FoodLog();
            m1.myColor1 = new Color(234, 129, 161); //pink
            w1.myColor1 = new Color(234, 129, 161); //pink
            p1.myColor1 = new Color(234, 129, 161); //pink
            f1.myColor1 = new Color(234, 129, 161); //pink
            User user = new User();
            m1.initialize(user);
            dispose();
        }
    });

```


Example of theme change:

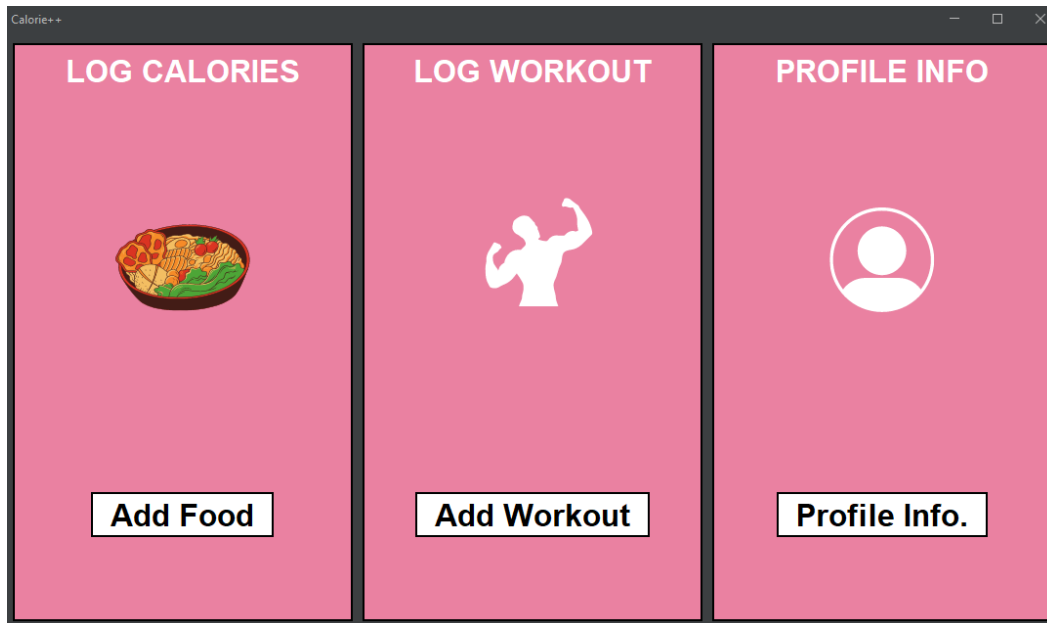


Figure 19 - MainFrame post-theme change

4 CONCLUSION

A calorie tracker built through a Java GUI is a good tool for beginners to start managing their healthy lifestyle. At first, the development was slow because of introduction to new technologies such as Swing and databases, but through development and familiarization this project was able to be completed. There are future implementations that could be added to Calorie++, but the current state of the application is more than sufficient. Each module was developed with a plan and through many errors, they were able to meet their functionality.

Overall, this project was really helpful in learning the possibilities available with Java, and how to make components interact with each other as well as combining libraries. Hopefully with advanced Java programming it is possible to see how an application as simple as this one can be further improved with more modern technologies. The biggest takeaway from this whole process is how much options there are with programming in general, but also Java, as most of these libraries are not really known yet there are a lot of documentation out there for anyone to experiment with.

REFERENCES

- [1] “History of Java Programming Language.” *History of Java Programming Language*, <https://www.freejavaguide.com/history.html>.
- [9] “Introduction to Java3D.” *Introduction to Java 3D*, https://docs.oracle.com/cd/E17802_01/j2se/javase/technologies/desktop/java3d/forDevelopers/j3dguide/Intro.doc.html#47293.
- [4] “Java AWT Tutorial - Javatpoint.” *Www.javatpoint.com*, <https://www.javatpoint.com/java-awt>.
- [2] “Java Developer's Guide.” *Oracle Help Center*, 17 Oct. 2017, <https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/Java-overview.html>.
- [5] “JavaFX Tutorial - Javatpoint.” *Www.javatpoint.com*, <https://www.javatpoint.com/javafx-tutorial>.
- [6] “Lesson: Overview of the Java 2D API Concepts.” *Lesson: Overview of the Java 2D API Concepts (The Java™ Tutorials > 2D Graphics)*, <https://docs.oracle.com/javase/tutorial/2d/overview/index.html>.
- [3] “Object Oriented Programming (OOPS) Concept in Java.” *GeeksforGeeks*, 21 Nov. 2022, <https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/>.
- [7] “What Is a Database?” *Oracle*, <https://www.oracle.com/database/what-is-database/>.
- [8] “What Is an API?” *Red Hat - We Make Open Source Technologies for the Enterprise*, <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>.