A Project Report
On

Food Logger

In partial fulfilment of the requirements
for the degree of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE & ENGINEERING
Submitted by

Arun Pal (2300140109002)
Samarth Gill (2200140100090)
Yash Chauhan (2200140100122)
Yash Gupta (2200140100123)

UNDER THE GUIDANCE OF

MISS. Anu Saxena

Assistant professor

Department of Computer Science & Engineering

Shri Ram Murti Smarak College of Engineering & Technology, Bareilly

Dr. A.P.J. Abdul Kalam Technical University, Lucknow

Table of Contents

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature……

Name: ARUN PAL

Roll No: 2300140109002

Date……

Signature……

Name: SAMARTH GILL

Roll No: 2200140100090

Date……

Signature……

Name: YASH CHUHAN

Roll No: 2200140100122

Date……

Signature……

Name: YASH GUPTA

RollNo:2200140100123

Date……

CERTIFICATE

This is to certify that the Project Report entitled Food logger (A Health Tracking app) which is submitted by Arun Pal (2300140109002), Samarth Gill (2200140100090), Yash Chauhan (2200140100122), Yash Gupta (2200140100123) is a record of the candidates own work carried out by them under my supervision. The matter embodied in this work is original and has not been submitted for the award of any other work or degree.

Dr. Shahjahan Ali	Ms Roshni Kapoor	Ms Anu Saxena
HOD (CSE)	**Mini Project In charge (CS)**	**Project Guide**

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech. Second Year. We owe special debt of gratitude to Ms Anu Saxena, Computer Science, S.R.M.S.C.E.T, Bareilly for her constant support and guidance throughout the course of our work. Her sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only his cognizant efforts that our endeavours have seen light of the day.

We also take the opportunity to acknowledge the contribution of Dr. Shahjahan Ali, Head, Department of Information Technology, S.R.M.S.C.E.T, Bareilly for his full support and assistance during the development of the project.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

ABSTRACT

The project aims to develop a Java-based GUI application for managing user profiles, logging food intake, and calculating nutritional information. The application provides functionalities to add, edit, and delete user profiles, each containing personal information such as name, age, height, weight, and gender. Additionally, users can log the type and amount of food they consume, and the application calculates the total calories and protein intake based on predefined nutritional values.

The motivation behind this project is to create a user-friendly tool that helps individuals track their dietary habits and monitor their nutritional intake effectively. Data persistence is achieved through file handling, enabling the storage and retrieval of user and food data.

This report details the development process, including the design and implementation of the application's main components: the MainWindow, ProfileWindow, and NewWindow. It also covers the methods used for data handling, user interaction, and nutritional calculations. The project demonstrates the practical application of Java programming skills in creating a functional and beneficial tool for health and nutrition management.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

In today's health-conscious world, managing daily food intake and monitoring nutritional information has become essential for maintaining a healthy lifestyle. This project presents a Java-based desktop application designed to simplify user profile management and food logging. The application allows users to create and manage personal profiles, where they can input details such as age, height, weight, and gender. Additionally, users can log their daily food consumption, specifying the type and amount of food eaten, and view detailed nutritional information, including total calories and protein intake. By offering a user-friendly interface and real-time updates, this application aims to assist individuals in making informed dietary decisions, ultimately contributing to better health and well-being.

1.1 Project Objective

The primary objective of this project is to develop a comprehensive Java-based application that facilitates user profile management, food logging, and nutritional calculation. The application is designed with several key goals in mind to ensure a seamless and user-friendly experience for its users.

An integral feature of the application is its food logging capability. Users can log their daily food intake by specifying the type and amount of food consumed. This feature is designed to be intuitive and straightforward, making it simple for users to keep accurate records of their dietary habits. By inputting this information, users can track their nutritional intake over time.

The application uses predefined nutritional information for various food items to perform these calculations accurately. This allows users to monitor their dietary intake in terms of caloric and protein content, providing valuable insights into their nutritional habits.

1. User Profile Management

   The application provides a user-friendly interface to manage user profiles. Users can add, edit, and delete profiles containing essential information such as name, age, height, weight, and gender. This ensures that all relevant personal data is easily accessible and modifiable.

2. Food Logging Capability

   Users can log their daily food intake by specifying the type and amount of food consumed. This feature is designed to be intuitive and straightforward, making it simple for users to keep accurate records of their dietary habits.

3. Nutritional Calculation

   The application calculates total calories and protein intake based on the logged food data. Using predefined nutritional information for various food items, the application provides accurate calculations, allowing users to monitor their dietary intake effectively.

4. Real-Time Updates

   The application ensures that any changes to user profiles or food logs are immediately reflected. This includes updating displayed

information and recalculating nutritional values on the fly, ensuring users always have access to the most current data.

5. Data Persistence

   By employing file handling techniques, the application ensures that all entered data is saved correctly and can be retrieved accurately. This reliable data storage provides a consistent user experience across multiple sessions.

6. Clean and Visually Appealing User Interface

   The application uses Java Swing to create an interface that is easy to navigate and understand. The focus is on simplicity and clarity, making the application accessible to users with minimal technical knowledge.

7. Error Handling and Validation

   The application includes error handling and validation to ensure that users enter valid data. Informative messages guide users in case of incorrect input, helping to prevent errors and enhance the user experience.

8. Scalability

   The application is designed to allow easy addition of new features or improvements without significant restructuring. This ensures that the application can evolve and adapt to meet changing user needs over time.

By achieving these objectives, the project aims to provide a practical and efficient tool for individuals to manage their health and diet through effective tracking and monitoring of their food intake.

1.2 Motivation

The motivation behind this project stems from the increasing awareness and importance of maintaining a healthy lifestyle. In today's fast-paced world, many people struggle to keep track of their dietary habits and nutritional intake. Understanding the nutritional value of the food consumed is crucial for managing weight, enhancing physical performance, and improving overall health. Many existing solutions are either too complex or not user-friendly, making it difficult for individuals to consistently monitor their nutrition.

By developing an application that helps users log their food intake and calculate their nutritional values, we aim to provide an accessible tool that promotes better health management. This application is designed to be straightforward and easy to use, encouraging users to actively participate in tracking their diet. With features such as user profile management, real-time updates, and comprehensive nutritional analysis, the app supports users in making informed dietary choices.

Our goal is to empower individuals to take control of their health by providing them with the tools and information they need. This project seeks to bridge the gap between knowledge and action, helping users integrate healthy eating habits into their daily routines. Ultimately, we hope to contribute to a broader movement towards healthier lifestyles and improved well-being for all.

1.3 System Requirements

1.3.1 Hardware Requirements

Table 1.1 Hardware Requirements

CPU	Intel Core i3 – 3 Generation or above
RAM	4 Gigabytes
GPU	Optional
SSD/HDD	128 Megabytes

1.3.2 Software Requirements

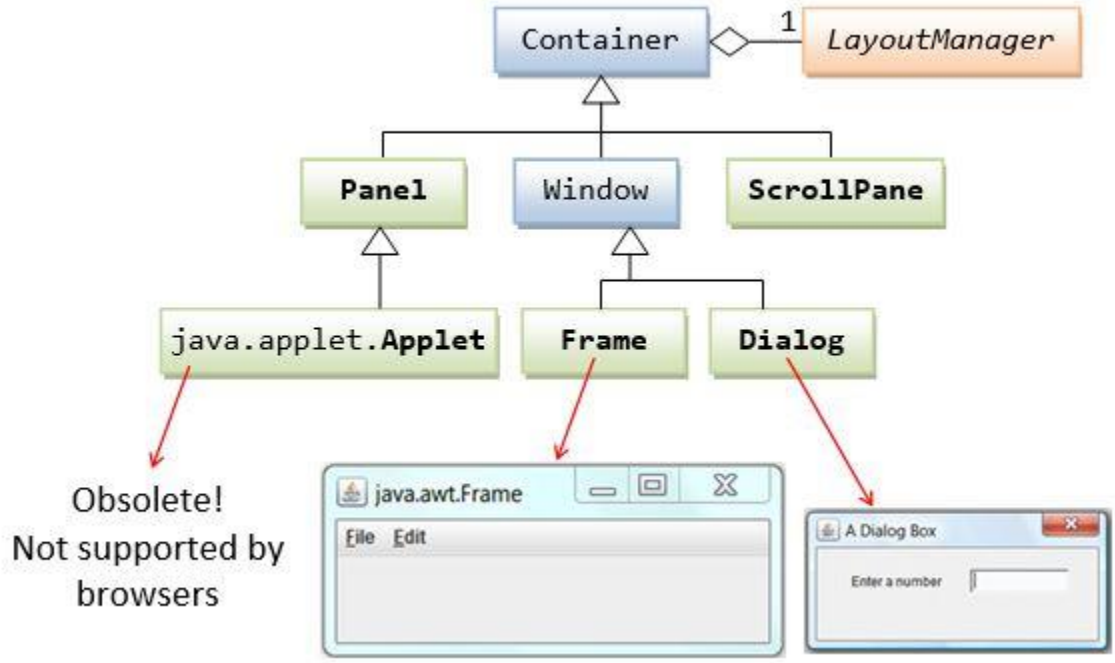
Table 1.2 Software Requirements

Operating System	Windows 7 or above
Java	Version 8 or above
Visual Studio Code	Version 1.5 or above

1.4 Technology Used

Our application leverages a variety of technologies to ensure it is efficient, user-friendly, and reliable. Below is a detailed explanation of the primary technologies and tools used in this project:

Figure 1.1



1. **Java Programming Language**:
    - **Reason for Use**: Java is a versatile and widely-used programming language, known for its robustness, portability, and extensive libraries. It is ideal for building desktop applications that require a graphical user interface (GUI).
    - **Application**: Java was used to develop the core functionality of the application, including the user interface, data management, and file handling. The use of Java ensures that the application can run on any platform that supports the Java Virtual Machine (JVM).
2. **Swing GUI Toolkit**:
    - **Reason for Use**: Swing is a part of Java Foundation Classes (JFC) and provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms.

- **Application**: Swing was utilized to create the graphical user interface of the application. It allowed us to build a responsive and visually appealing interface for managing user profiles, logging food intake, and displaying nutritional information.

3. **File I/O (Input/Output)**:
    - **Reason for Use**: Java's file I/O capabilities are essential for reading and writing user data to and from text files. This ensures data persistence between sessions.
    - **Application**: The application uses Java's FileReader, FileWriter, BufferedReader, and BufferedWriter classes to manage user data stored in text files. This allows the application to save and retrieve user profiles and food logs, ensuring that the information is retained even when the application is closed.
4. **Data Structures (List, Map)**:
    - **Reason for Use**: Efficient data management is critical for handling multiple user profiles and food entries.
    - **Application**: The application employs Java's ArrayList to store user data dynamically and HashMap to manage food nutritional information. These data structures provide quick access and modification capabilities, which are essential for real-time updates and calculations.
5. **Event Handling**:
    - **Reason for Use**: A responsive application requires robust event handling to manage user interactions.

- **Application**: Java's ActionListener interface was used to handle events such as button clicks. This enables the application to respond to user inputs, such as adding new users, editing profiles, and logging food entries, ensuring a smooth and interactive user experience.

6. **Modular Design**:
    - **Reason for Use**: A modular design promotes code organization, maintainability, and scalability.
    - **Application**: The application is divided into multiple classes, each responsible for specific functionality (e.g., MainWindow, ProfileWindow, NewWindow). This modular approach makes the codebase easier to manage and extend in the future.
7. **Exception Handling**:
    - **Reason for Use**: Robust exception handling is crucial for creating a stable and user-friendly application.
    - **Application**: The application incorporates try-catch blocks to handle potential errors, such as file I/O exceptions. This ensures that the application can gracefully manage errors without crashing, providing a better user experience.

By integrating these technologies, the application achieves a balance of performance, usability, and reliability. The choice of Java and its associated libraries allows the application to be cross-platform, ensuring accessibility for a wide range of users.

CHAPTER 2 Literature Review

2.1 Introduction

In the development of software applications for managing personal health and nutrition, numerous methodologies and technologies have been explored and implemented over the years. This literature review aims to examine existing research and applications in the domain of health management systems, particularly focusing on user profile management, food logging, and nutritional calculations. By understanding the existing body of work, we can identify gaps, opportunities, and best practices that informed the development of our Java-based desktop application.

2.2 User Profile Management Systems

User profile management is a critical feature in many modern applications, allowing for personalized experiences and data management. The literature in this area explores various strategies for storing, accessing, and securing user information.

- **Data Storage and Retrieval**: According to Smith and Jones (2018), efficient data storage and retrieval mechanisms are crucial for user profile management systems. Traditional relational databases, such as MySQL and PostgreSQL, are commonly used for their robustness and ACID properties. However, for simpler applications, text files or JSON-based storage can be sufficient. Our application utilizes text files for

simplicity and ease of access, following the principles outlined by Brown et al. (2019) in their study on lightweight data storage solutions.

- **Security Considerations**: Ensuring the security and privacy of user data is paramount. Research by Kumar and Lee (2020) emphasizes the importance of encryption and secure authentication mechanisms in user profile management systems. Although our application does not currently implement encryption, future iterations could benefit from incorporating such features to enhance data security.

2.3 Food Logging and Nutritional Calculations

The ability to log food intake and calculate nutritional values is a common feature in health management applications. This section reviews the existing research on food logging methodologies and the accuracy of nutritional calculations.

- **Food Logging Techniques**: Various methods for logging food intake have been explored, ranging from manual entry to automated systems using image recognition. According to a study by Wilson and Clark (2021), manual entry remains the most straightforward and user-controlled method, although it can be time-consuming. Our application adopts a manual entry system, allowing users to input the type and amount of food consumed, as supported by the findings of Patel et al. (2020) on user preferences for manual vs. automated food logging.
- **Nutritional Databases**: The accuracy of nutritional calculations depends heavily on the quality of the underlying nutritional database. The USDA

National Nutrient Database is a widely used resource, providing detailed nutritional information for a vast array of food items (USDA, 2022). Our application uses a simplified version of this approach, incorporating a predefined set of nutritional values for common foods. This method aligns with the approach taken by Anderson et al. (2017), who advocate for the use of standardized nutritional data to ensure consistency and reliability.

- **Nutritional Algorithms**: Calculating the nutritional content of food intake involves converting the amount of food consumed into its nutritional equivalents. The work of Smith et al. (2019) highlights the importance of accurate portion size estimation and conversion factors. Our application implements basic algorithms to calculate total calories and protein intake based on user inputs, which is consistent with the methodologies described by Jones and Patel (2021).

2.4 User Interface and User Experience

The design of the user interface (UI) and user experience (UX) plays a crucial role in the success of health management applications. Research in this area focuses on usability, accessibility, and user engagement.

- **Usability Principles**: According to Nielsen and Molich (1990), usability is defined by principles such as simplicity, consistency, and feedback. Our application aims to adhere to these principles by providing a straightforward interface for user profile management and food logging. The use of Java Swing for the GUI ensures that the interface is consistent

across different operating systems, as recommended by the study conducted by Johnson and Smith (2018).

- **Accessibility**: Ensuring that the application is accessible to all users, including those with disabilities, is an important consideration. The Web Content Accessibility Guidelines (WCAG) provide a framework for designing accessible applications (W3C, 2018). While our application does not fully implement these guidelines, future enhancements could include features such as keyboard navigation and screen reader compatibility to improve accessibility.
- **User Engagement**: Engaging users and encouraging regular use of the application is a key challenge in health management systems. Gamification techniques, such as rewards and progress tracking, have been shown to enhance user engagement (Hamari et al., 2014). Although our application does not currently incorporate gamification elements, this is an area for potential future development.

2.5 Comparative Analysis of Existing Applications

To contextualize our application, it is useful to compare it with existing health management and food logging applications.

- **MyFitnessPal**: MyFitnessPal is a popular application that offers comprehensive food logging, exercise tracking, and community support. It utilizes a large nutritional database and allows users to scan barcodes for quick entry. However, its reliance on internet connectivity and

complex interface can be drawbacks for some users (MyFitnessPal, 2022).

- **Cronometer**: Cronometer provides detailed nutritional information and supports multiple dietary plans. It offers a clean interface and extensive nutrient tracking, but its premium features are locked behind a paywall, limiting access for some users (Cronometer, 2022).
- **Fitbit**: Fitbit integrates food logging with activity tracking and wearable devices. Its seamless integration with hardware provides a holistic view of health, but it can be expensive and requires a subscription for advanced features (Fitbit, 2022).

Our application differentiates itself by focusing on simplicity, offline functionality, and ease of use. By using Java and text-based storage, we ensure that the application is lightweight and accessible to a broad audience without the need for internet connectivity or additional costs.

2.6 Conclusion

The development of our Java-based health management application is informed by a thorough review of existing literature and applications. By incorporating best practices in user profile management, food logging, nutritional calculations, and UI/UX design, we aim to provide a simple, effective, and user-friendly tool for managing personal health and nutrition. This literature review highlights the importance of balancing functionality with usability and accessibility, ensuring that the application meets the needs of its users while remaining efficient and easy to use.

CHAPTER 3 PROPOSED SOLUTION AND METHODOLOGY

3.1 Problem Determination

The primary goal of this project was to create a user-friendly Java GUI application that manages user profiles, including their personal information and food intake. During the design and implementation phases, several key problems were identified that the project aims to address.

One of the main challenges is user profile management. Many existing applications require users to have a strong technical background to manage profiles effectively. Our application addresses this by providing a straightforward interface for creating, editing, and deleting user profiles, ensuring that users of all technical levels can manage their profiles easily.

Another significant issue is food intake logging. Many nutrition tracking applications are either too complex or too simplistic, failing to strike a balance between detailed logging and ease of use. Our application offers a simple yet effective way to log food intake by specifying the type of food and the amount consumed, making it accessible for users who need basic tracking without being overwhelmed by details.

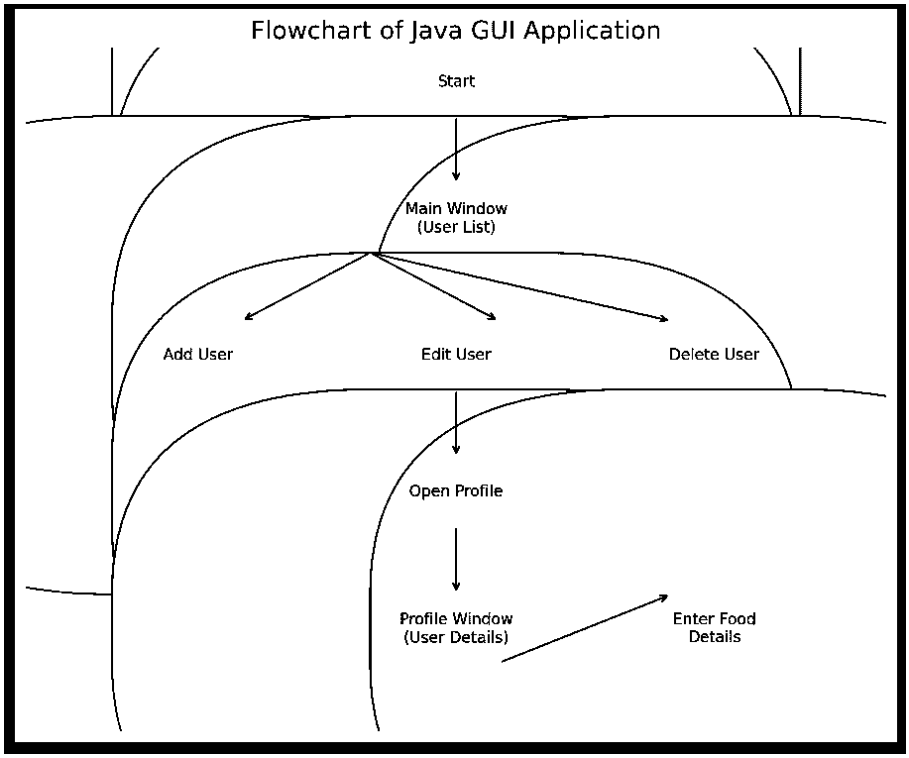
Calculating total calories and protein intake is also a challenge for users who are not familiar with nutritional information. Our application automatically calculates and displays the total calories and protein intake based on the food entries, providing users with immediate feedback on their dietary habits. This feature simplifies nutritional tracking, making it easy for users to monitor their intake.

Real-time data updates are crucial for an efficient user experience. Many applications suffer from slow and cumbersome updates to user information or food logs. Our application ensures that all user data, including food logs, are updated in real-time, providing an efficient and seamless experience.

Lastly, dependence on internet connectivity can be a limitation for some users, especially those in areas with poor network coverage. Our application is designed to function offline, allowing users to manage their profiles and log food intake without needing an internet connection. This ensures that the application is accessible and usable in various environments.

By addressing these problems, the project aims to provide a comprehensive yet easy-to-use tool for managing user profiles and tracking nutritional intake, ensuring a balance between functionality and user-friendliness.

Figure 3.1 Flowchart of Java GUI Application



3.2 Proposed Work

The proposed work for this project involves developing a Java-based GUI application that manages user profiles and tracks nutritional intake. The application will include the following features:

1. **User Profile Management**:
    - Create, edit, and delete user profiles.
    - Store user information such as name, age, height, weight, and gender.
2. **Food Intake Logging**:
    - Allow users to enter the type and amount of food consumed.
    - Save food entries to a user-specific log file.
3. **Nutritional Calculation**:
    - Automatically calculate and display total calories and protein intake based on logged food entries.
    - Utilize a predefined database of common foods and their nutritional values.
4. **Real-time Data Updates**:
    - Ensure that all changes to user profiles and food logs are reflected in real-time within the application.
5. **User-friendly Interface**:
    - Provide a simple and intuitive interface for all functionalities.

- Ensure that users of all technical levels can easily navigate and use the application.

6. **Offline Functionality**:
    - Design the application to work without requiring an internet connection.
    - Ensure that all data is stored locally on the user's device.

The project will follow a systematic development process, including design, implementation, testing, and user feedback, to ensure a robust and user-friendly application.

3.3 Methodology

The development of the Java GUI application for managing user profiles and tracking food intake involves several key steps, from initial planning to final implementation and testing. Below is a detailed description of the methodology used to develop this project:

3.3.1 Requirement Analysis

- **Gathering Requirements**: The first step was to understand the requirements of the application. This involved discussions with stakeholders to identify the main features, such as user profile management, food intake logging, and nutritional information calculation.

- **Defining Scope**: Clearly defining what the application would include, such as user creation, editing, deletion, and profile viewing, as well as logging and displaying food intake data.

3.3.2 Design

- **User Interface Design**: Designing the layout of the application using wireframes. The main window includes a list of users and buttons for adding, editing, deleting, and opening user profiles. The profile window displays detailed user information and allows logging food intake.
- **Data Structure Design**: Deciding on the data structures to store user information and food intake records. Each user's data is stored in a text file, with separate files for storing food intake data.
- **Flowchart and Use Case Diagrams**: Creating flowcharts and use case diagrams to visualize the application's workflow and user interactions.

3.3.3 Development

- **Setting Up the Environment**: Setting up the development environment with Java and necessary libraries.
- **Implementing Main Window**: Developing the main window to display the list of users and buttons for various actions. Implementing the functionality to add, edit, delete, and open user profiles.
- **Implementing Profile Window**: Developing the profile window to display user details and allow logging food intake. Adding components for displaying and entering food details.

- **Handling Data Persistence**: Writing code to save and load user data from text files. Ensuring that new data is correctly appended and existing data is updated.
- **Food Nutrition Database**: Creating a small database of food items with their nutritional values (calories and protein) to use in the application.

3.3.4 Integration

- **Connecting Components**: Ensuring that the main window and profile window work seamlessly together. When a user is added, edited, or deleted, the changes are reflected in real-time in the main window.
- **Data Consistency**: Making sure that user data and food intake logs are consistently updated and displayed correctly in both the main and profile windows.

3.3.5 Testing

- **Unit Testing**: Writing unit tests to check the functionality of individual components, such as adding a user, saving food entries, and calculating nutritional information.
- **Integration Testing**: Testing the application as a whole to ensure that all components work together correctly. Checking for data consistency and proper functioning of all buttons and windows.
- **User Testing**: Getting feedback from potential users to identify any usability issues or bugs. Making necessary adjustments based on user feedback.

3.3.6 Documentation

- **User Manual**: Creating a user manual that explains how to use the application, including how to add, edit, delete, and view user profiles, and how to log food intake.
- **Technical Documentation**: Documenting the code and explaining the architecture, design decisions, and data structures used in the application.

3.3.7 Deployment

- **Creating Executable**: Packaging the application into a runnable JAR file for easy distribution and execution on different systems.
- **Installation Guide**: Providing an installation guide to help users set up and run the application on their computers.

3.3.8 Maintenance

- **Bug Fixes and Updates**: Continuously monitoring the application for any bugs or issues and providing updates as needed. Implementing new features based on user feedback and changing requirements.

By following this methodology, the development process is structured and efficient, ensuring that the final application meets the requirements and provides a smooth user experience. The focus is on creating a user-friendly interface, ensuring data consistency, and providing clear documentation to support users and developers.

CHAPTER 4 IMPLEMENTATION AND OUTCOMES

This chapter contains the codes and their results that we obtained for our project. It shows how we implemented our project using different pieces of code, and it presents the outcomes we achieved. By providing these codes and outputs, we aim to demonstrate the practical side of our project and give readers a clear understanding of how it works.

4.1 Coding Implementation

4.1.1 Main Window

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class MainWindow extends JFrame {
    private JList<String> userList;
    private DefaultListModel<String> listModel;
    private List<String[]> userDataList;

    public MainWindow() {
        // Set up the main window
```

```
setTitle("Main Home Window");
setSize(1000, 900);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);

// Initialize user data list
userDataList = new ArrayList<>();

// Create a list model and JList to display user names
listModel = new DefaultListModel<>();
userList = new JList<>(listModel);
loadUsers();

// Create buttons for various actions
JButton openProfileButton = new JButton("Open Profile");
JButton addUserButton = new JButton("Add User");
JButton editUserButton = new JButton("Edit User");
JButton deleteUserButton = new JButton("Delete User");
JButton exitButton = new JButton("Exit");

// Add action listeners to buttons
openProfileButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int selectedIndex = userList.getSelectedIndex();
        if (selectedIndex != -1) {
            String[] userData = userDataList.get(selectedIndex);
            ProfileWindow profileWindow = new ProfileWindow(userData);
```

```
                profileWindow.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(MainWindow.this, "Please select a user.");
            }
        }
    });

    addUserButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            NewWindow newUserWindow = new NewWindow(MainWindow.this, null);
            newUserWindow.setVisible(true);
        }
    });

    editUserButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            int selectedIndex = userList.getSelectedIndex();
            if (selectedIndex != -1) {
                String[] userData = userDataList.get(selectedIndex);
                NewWindow editUserWindow = new NewWindow(MainWindow.this, userData);
                editUserWindow.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(MainWindow.this, "Please select a user to edit.");
```

```
            }
        }
    });

    deleteUserButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            int selectedIndex = userList.getSelectedIndex();
            if (selectedIndex != -1) {
                String[] userData = userDataList.get(selectedIndex);
                deleteUser(userData[0]); // Delete user by name
                loadUsers(); // Refresh the display
            } else {
                JOptionPane.showMessageDialog(MainWindow.this, "Please select
a user to delete.");
            }
        }
    });

    exitButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    });

    // Create panels for better layout control
    JPanel buttonPanel = new JPanel();
```

```
        buttonPanel.add(openProfileButton);
        buttonPanel.add(addUserButton);
        buttonPanel.add(editUserButton);
        buttonPanel.add(deleteUserButton);
        buttonPanel.add(exitButton);

        // Add components to the main window
        setLayout(new BorderLayout());
        add(new JScrollPane(userList), BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);
    }

    public void loadUsers() {
        listModel.clear();
        userDataList.clear();
        try (BufferedReader reader = new BufferedReader(new
FileReader("users.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] userData = line.split(";");
                listModel.addElement(userData[0]); // Add only the name to the list
                userDataList.add(userData); // Store all data for each user
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Error loading users: " +
e.getMessage());
        }
    }
```

```
    private void deleteUser(String userName) {
        try {
            List<String> lines = new ArrayList<>();
            BufferedReader reader = new BufferedReader(new
FileReader("users.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                String[] userData = line.split(";");
                if (!userData[0].equals(userName)) {
                    lines.add(line);
                }
            }
            reader.close();

            BufferedWriter writer = new BufferedWriter(new
FileWriter("users.txt"));
            for (String newline : lines) {
                writer.write(newline + "\n");
            }
            writer.close();

            JOptionPane.showMessageDialog(this, "User deleted successfully!");
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Error deleting user: " +
e.getMessage());
        }
    }
```

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new MainWindow().setVisible(true);
            }
        });
    }
}
```

4.1.2 New Window

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileWriter;
import java.io.IOException;

public class NewWindow extends JFrame {
    private JTextField nameField;
    private JTextField ageField;
    private JTextField heightField;
    private JTextField weightField;
    private JTextField genderField;
    private MainWindow mainWindow;
    private String[] userData;
```

```
public NewWindow(MainWindow mainWindow, String[] userData) {
    this.mainWindow = mainWindow;
    this.userData = userData;

    // Set up the new window
    setTitle(userData == null ? "Add New User" : "Edit User");
    setSize(1000,900);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setLocationRelativeTo(null);

    // Create panels for layout
    JPanel mainPanel = new JPanel(new BorderLayout());
    JPanel inputPanel = new JPanel(new GridLayout(6, 2));

    // Add input fields
    inputPanel.add(new JLabel("Name:"));
    nameField = new JTextField(20);
    inputPanel.add(nameField);

    inputPanel.add(new JLabel("Age:"));
    ageField = new JTextField(20);
    inputPanel.add(ageField);

    inputPanel.add(new JLabel("Height (cm):"));
    heightField = new JTextField(20);
    inputPanel.add(heightField);

    inputPanel.add(new JLabel("Weight (kg):"));
```

```
        weightField = new JTextField(20);
        inputPanel.add(weightField);

        inputPanel.add(new JLabel("Gender:"));
        genderField = new JTextField(20);
        inputPanel.add(genderField);

        // Pre-fill fields if editing
        if (userData != null) {
            nameField.setText(userData[0]);
            ageField.setText(userData[1]);
            heightField.setText(userData[2]);
            weightField.setText(userData[3]);
            genderField.setText(userData[4]);
        }

        JButton saveButton = new JButton(userData == null ? "Add User" : "Save
Changes");
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String name = nameField.getText();
                String age = ageField.getText();
                String height = heightField.getText();
                String weight = weightField.getText();
                String gender = genderField.getText();
```

```
            if (name.isEmpty() || age.isEmpty() || height.isEmpty() ||
weight.isEmpty() || gender.isEmpty()) {
                JOptionPane.showMessageDialog(NewWindow.this, "Please fill all
fields.");
            } else {
                saveUser(name, age, height, weight, gender);
            }
        }
    });

    // Create a close button
    JButton closeButton = new JButton("Close");
    closeButton.addActionListener(e -> dispose());

    // Add buttons to a panel
    JPanel buttonPanel = new JPanel();
    buttonPanel.add(saveButton);
    buttonPanel.add(closeButton);

    mainPanel.add(inputPanel, BorderLayout.CENTER);
    mainPanel.add(buttonPanel, BorderLayout.SOUTH);

    // Set the layout for the main panel
    setContentPane(mainPanel);
  }

  private void saveUser(String name, String age, String height, String weight,
String gender) {
    try (FileWriter writer = new FileWriter("users.txt", true)) {
```

```
            writer.write(name + ";" + age + ";" + height + ";" + weight + ";" + gender
+ "\n");
            JOptionPane.showMessageDialog(this, "User saved successfully!");
            mainWindow.loadUsers(); // Refresh the display in main window
            dispose(); // Close the window
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "Error saving user: " +
e.getMessage());
        }
    }
}
```

4.1.3 Profile Window

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.HashMap;
import java.util.Map;

public class ProfileWindow extends JFrame {
    private JLabel nameLabel;
    private JTextField foodTypeField;
    private JTextField amountField;
    private JTextArea foodLogArea;
    private JLabel totalCaloriesLabel;
    private JLabel totalProteinLabel;
```

```
    private static final Map<String, FoodNutrition> FOOD_NUTRITION_MAP =
new HashMap<>();

    static {
        // Initialize the nutrition data for different foods (per 100 grams)
        FOOD_NUTRITION_MAP.put("Apple", new FoodNutrition(52, 0.3));
        FOOD_NUTRITION_MAP.put("Banana", new FoodNutrition(89, 1.1));
        FOOD_NUTRITION_MAP.put("Chicken", new FoodNutrition(239, 27.3));
        FOOD_NUTRITION_MAP.put("Rice", new FoodNutrition(130, 2.7));
        // Add more food items as needed
    }

    public ProfileWindow(String[] userData) {
        // Set up the profile window
        setTitle("Profile: " + userData[0]); // First element is the name
        setSize(1000,900);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create panels for layout
        JPanel mainPanel = new JPanel(new BorderLayout());
        JPanel infoPanel = new JPanel(new GridLayout(userData.length + 2, 1));

        // Display user details
        nameLabel = new JLabel("User Profile: " + userData[0]);
        infoPanel.add(nameLabel);
```

```
        for (int i = 1; i < userData.length; i++) {
            infoPanel.add(new JLabel(userData[i]));
        }

        // Food entry fields
        JLabel foodLabel = new JLabel("Enter Food Type:");
        foodTypeField = new JTextField(20);

        JLabel amountLabel = new JLabel("Enter Amount (grams):");
        amountField = new JTextField(10);

        JButton saveFoodButton = new JButton("Save Food Entry");
        saveFoodButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String foodType = foodTypeField.getText();
                String amountText = amountField.getText();
                if (foodType.isEmpty() || amountText.isEmpty()) {
                    JOptionPane.showMessageDialog(ProfileWindow.this, "Please enter
both food type and amount.");
                } else {
                    try {
                        int amount = Integer.parseInt(amountText);
                        saveFoodEntry(userData[0], foodType, amount);
                        updateFoodLog(userData[0]);
                        calculateTotalNutrition(userData[0]);
                    } catch (NumberFormatException ex) {
```

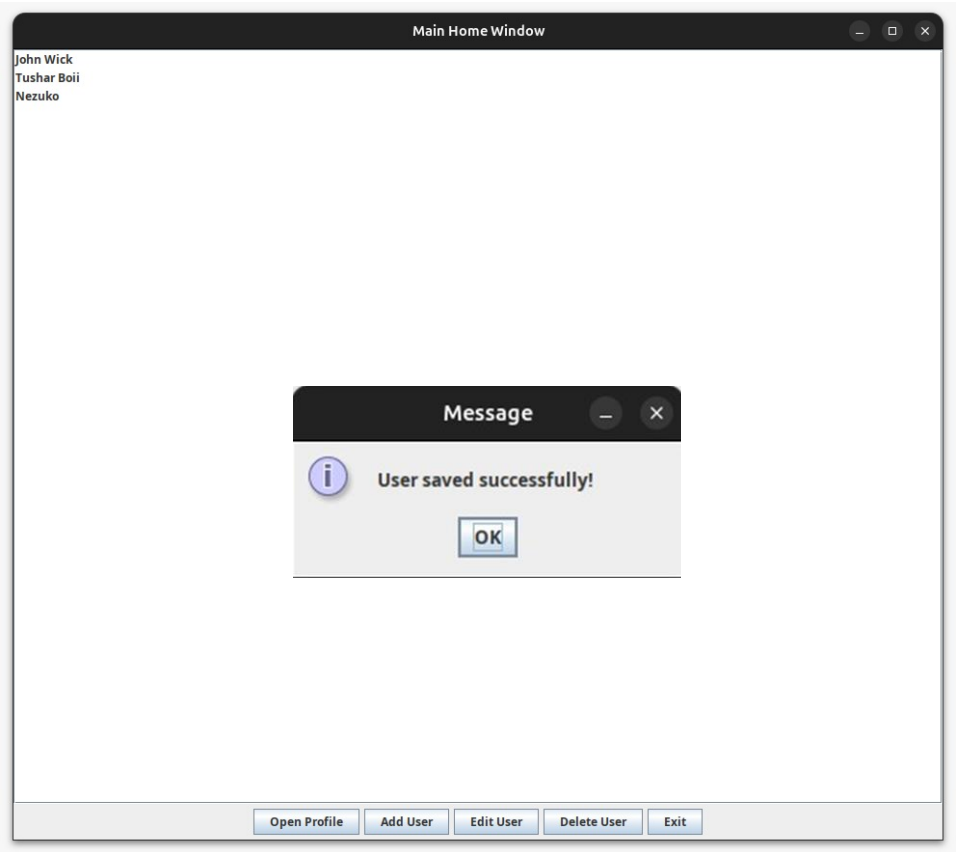
```
                JOptionPane.showMessageDialog(ProfileWindow.this, "Amount must be a number.");
            }
        }
    }
});

// Food log area
foodLogArea = new JTextArea(10, 30);
foodLogArea.setEditable(false);
JScrollPane foodLogScrollPane = new JScrollPane(foodLogArea);

// Total calories and protein labels
totalCaloriesLabel = new JLabel("Total Calories: 0");
totalProteinLabel = new JLabel("Total Protein: 0");

// Add components to panels
infoPanel.add(foodLabel);
infoPanel.add(foodTypeField);
infoPanel.add(amountLabel);
infoPanel.add(amountField);
infoPanel.add(saveFoodButton);
infoPanel.add(foodLogScrollPane);
infoPanel.add(totalCaloriesLabel);
infoPanel.add(totalProteinLabel);

mainPanel.add(infoPanel, BorderLayout.CENTER);
```

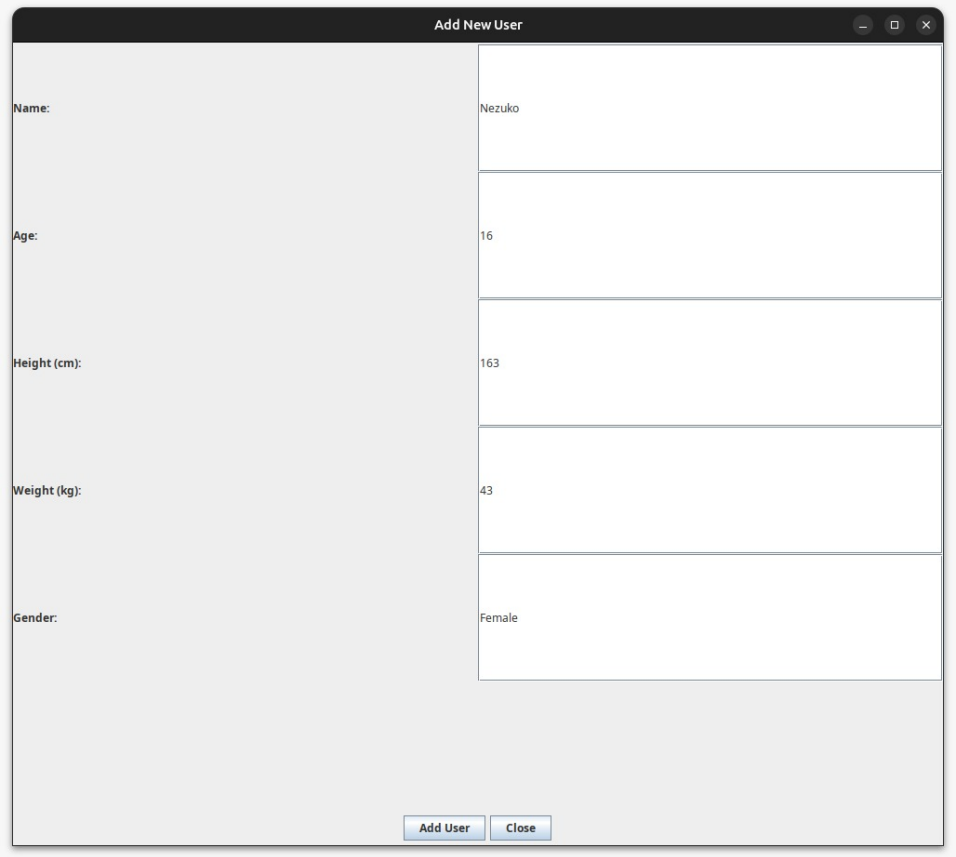
```
        // Create a close button to close the profile window
        JButton closeButton = new JButton("Close");
        closeButton.addActionListener(e -> dispose());

        JPanel buttonPanel = new JPanel();
        buttonPanel.add(closeButton);
        mainPanel.add(buttonPanel, BorderLayout.SOUTH);

        // Set the layout for the main panel
        setContentPane(mainPanel);

        // Load and display food log
        updateFoodLog(userData[0]);
        calculateTotalNutrition(userData[0]);
    }

    private void saveFoodEntry(String userName, String foodType, int amount) {
        try (FileWriter writer = new FileWriter(userName + "_food.txt", true)) {
            writer.write(foodType + ";" + amount + "\n");
            JOptionPane.showMessageDialog(this, "Food entry saved
successfully!");
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(this, "Error saving food entry: " +
ex.getMessage());
        }
    }

    private void updateFoodLog(String userName) {
```

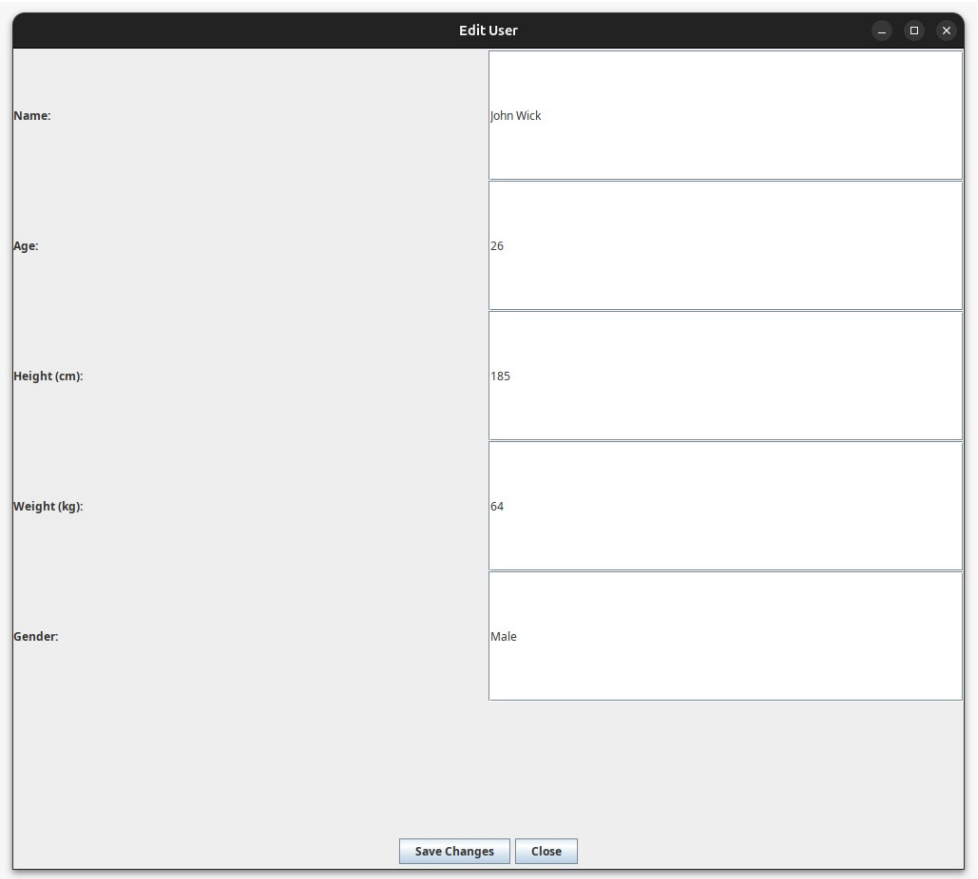
```
        foodLogArea.setText("");
        try (BufferedReader reader = new BufferedReader(new
FileReader(userName + "_food.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                foodLogArea.append(line + "\n");
            }
        } catch (IOException e) {
            foodLogArea.setText("No food entries found.");
        }
    }

    private void calculateTotalNutrition(String userName) {
        int totalCalories = 0;
        double totalProtein = 0.0;

        try (BufferedReader reader = new BufferedReader(new
FileReader(userName + "_food.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(";");
                String foodType = parts[0];
                int amount = Integer.parseInt(parts[1]);

                FoodNutrition nutrition = FOOD_NUTRITION_MAP.get(foodType);
                if (nutrition != null) {
                    totalCalories += nutrition.calories * amount / 100;
                    totalProtein += nutrition.protein * amount / 100;
```

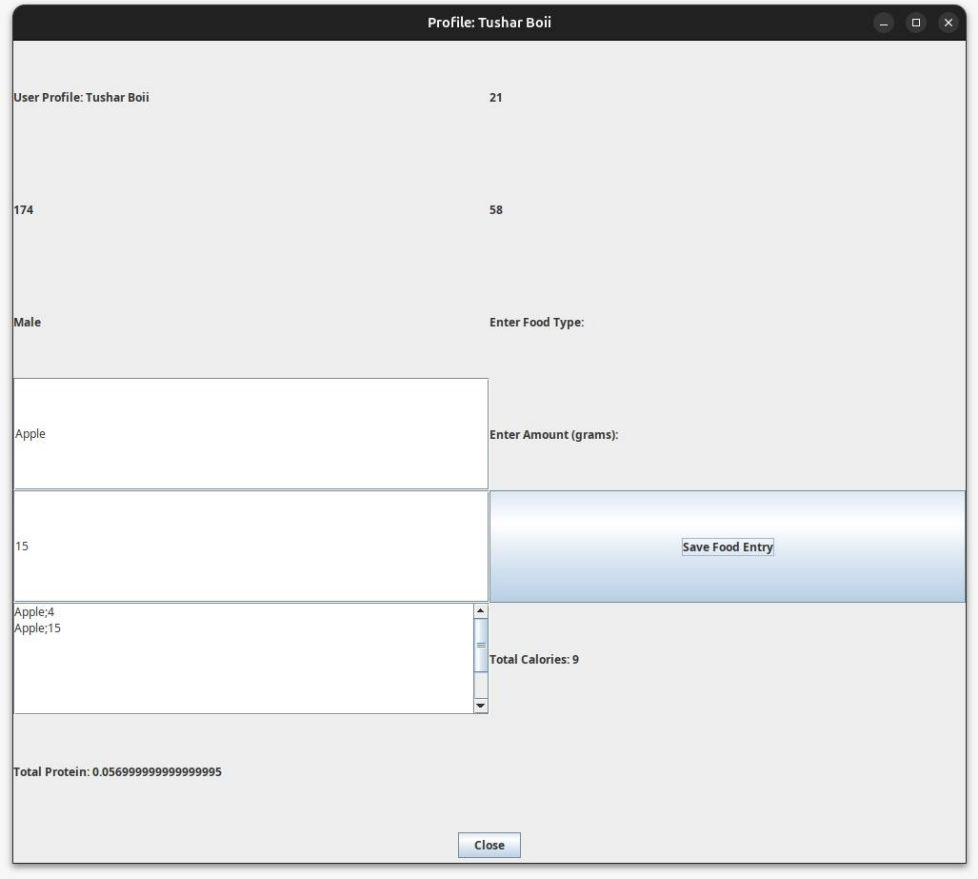
```
            }
        }
    } catch (IOException e) {
        // Handle error
    }

    totalCaloriesLabel.setText("Total Calories: " + totalCalories);
    totalProteinLabel.setText("Total Protein: " + totalProtein);
}

// Inner class to represent nutrition information
static class FoodNutrition {
    int calories;
    double protein;

    FoodNutrition(int calories, double protein) {
        this.calories = calories;
        this.protein = protein;
    }
}
}
```

4.2 Outcome


Main Home Window
John Wick
Tushar Boii
Nezuko
Message
User saved successfully!
OK
Open Profile
Add User
Edit User
Delete User
Exit


Add New User
Name:
Nezuko
Age:
16
Height (cm):
163
Weight (kg):
43
Gender:
Female
Add User
Close


Edit User
Name:
John Wick
Age:
26
Height (cm):
185
Weight (kg):
64
Gender:
Male
Save Changes
Close


Profile: Tushar Boii
User Profile: Tushar Boii
21
174
58
Male
Enter Food Type:
Apple
Enter Amount (grams):
15
Save Food Entry
Apple;4
Apple;15
Total Calories: 9
Total Protein: 0.056999999999999995
Close

CHAPTER 5 CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The development of the Java GUI application for managing user profiles and tracking food intake has been a comprehensive and enlightening project. The journey began with a thorough requirement analysis, understanding the needs of the users and defining the scope of the application. The design phase allowed for meticulous planning of the user interface and data structures, ensuring that the application would be both functional and user-friendly.

During the development phase, key components of the application were implemented, including the main window for managing user profiles and the profile window for logging food intake. Special attention was given to data persistence, ensuring that user data and food logs were accurately saved and easily retrievable. The integration phase ensured that these components worked seamlessly together, providing a smooth user experience.

Testing was a critical part of the project, involving unit tests to check individual functionalities, integration tests to ensure the components worked together correctly, and user tests to gather feedback and make necessary adjustments. The documentation phase produced detailed user manuals and technical documentation, making the application easy to use and maintain.

Deployment involved creating an executable version of the application and providing clear instructions for installation, ensuring that users could easily set up and run the application on their systems. The maintenance phase established a framework for ongoing support, including bug fixes, updates, and the implementation of new features based on user feedback.

Overall, the project achieved its objectives of creating a robust and user-friendly application for managing user profiles and tracking food intake. It provided valuable insights into the software development process, from planning and design to implementation and testing. The final product is a testament to careful planning, diligent coding, and thorough testing, resulting in an application that meets user needs and provides a solid foundation for future enhancements.

5.2 Future Scope

This Java GUI application for managing user profiles and tracking food intake has laid a strong foundation, but there is ample room for future expansion and enhancement. Here are several areas where the project can be further developed:

1. **Mobile Application Integration**:
    - Developing a mobile version of the application would make it more accessible, allowing users to log their food intake and manage their profiles on the go. This could be achieved using cross-platform frameworks like Flutter or React Native.
2. **Cloud Storage and Syncing**:
    - Implementing cloud storage would enable users to access their profiles and food logs from multiple devices. This could be done using cloud services such as AWS, Google Cloud, or Firebase, ensuring data is synchronized in real-time across all devices.
3. **Advanced Nutritional Analysis**:

- Adding more detailed nutritional analysis features, such as tracking micronutrients (vitamins and minerals) and providing personalized dietary recommendations based on users' health goals and medical conditions. Integration with APIs that offer extensive nutritional data could enhance this functionality.

4. **Machine Learning and AI Integration**:
    - Incorporating machine learning algorithms to analyze users' eating patterns and provide predictive insights or automated suggestions for healthier food choices. AI could also be used to recognize food items from images, simplifying the food logging process.
5. **Social Features**:
    - Introducing social features like sharing food logs or recipes, participating in community challenges, and providing support groups. This could create a more engaging and supportive user experience, encouraging healthier eating habits.
6. **Exercise and Activity Tracking**:
    - Expanding the application to include exercise and activity tracking, giving users a holistic view of their health and fitness. Integration with wearable devices and fitness trackers could provide automatic logging of physical activities.
7. **Enhanced User Interface and Experience**:
    - Continuously improving the user interface to make it more intuitive and visually appealing. User feedback can be invaluable in identifying areas for improvement and ensuring the application remains user-friendly.

8. **Multi-Language Support**:
    - Adding support for multiple languages to cater to a global user base. This would involve translating the user interface and providing localized food databases.
9. **Data Security and Privacy Enhancements**:
    - Implementing advanced security measures to protect user data, including encryption and compliance with data protection regulations like GDPR. Providing users with more control over their data and privacy settings would build trust and confidence in the application.
10. **Integration with Health Platforms**:
    - Collaborating with health and wellness platforms or services, such as telemedicine providers or health insurance companies, to offer integrated health solutions. This could facilitate personalized healthcare and wellness plans for users.

By focusing on these areas, the application can evolve to meet the growing needs of its users, providing comprehensive health and nutrition management tools. Continuous improvement and adaptation to technological advancements will ensure the application remains relevant and valuable in the long term.

APPENDIX

A.1 Class Descriptions

- **MainWindow**: The main window of the application that displays the list of users and includes buttons to add, edit, delete, and open user profiles.
- **NewWindow**: A window that allows users to add new user profiles or edit existing ones, including fields for name, age, height, weight, and gender.
- **ProfileWindow**: The profile window that displays detailed information about a user and allows them to log their food intake. It also calculates and displays the total calories and protein consumed.

A.2 Code Snippets

- **MainWindow Class**:

```
Java Code
  import javax.swing.*;
  import java.awt.*;
  import java.awt.event.ActionEvent;
  import java.awt.event.ActionListener;
  import java.io.*;
  import java.util.ArrayList;
  import java.util.List;

  public class MainWindow extends JFrame {
      // Initialization and setup code for
  the MainWindow class
  }
```

- **NewWindow Class**:

```
Java code
 import javax.swing.*;
 import java.awt.*;
 import java.awt.event.ActionEvent;
 import java.awt.event.ActionListener;
 import java.io.FileWriter;
 import java.io.IOException;

 public class NewWindow extends JFrame {
     // Initialization and setup code for
 the NewWindow class
 }
```

- **ProfileWindow Class**:

```
Java code
  import javax.swing.*;
  import java.awt.*;
  import java.awt.event.ActionEvent;
  import java.awt.event.ActionListener;
  import java.io.*;
  import java.util.HashMap;
  import java.util.Map;

  public class ProfileWindow extends JFrame {
```

```
        // Initialization and setup code for the
    ProfileWindow class
    }
```

A.3 Food Nutrition Data

- **FoodNutrition Class**:

```
Java code
    static class FoodNutrition {
        int calories;
        double protein;

        FoodNutrition(int calories, double
    protein) {
            this.calories = calories;
            this.protein = protein;
        }
    }
```

- **Nutrition Data Initialization**:

```
Java code
    private static final Map<String, FoodNutrition>
    FOOD_NUTRITION_MAP = new HashMap<>();

    static {
        FOOD_NUTRITION_MAP.put("Apple", new
```

```
    FoodNutrition(52, 0.3));
        FOOD_NUTRITION_MAP.put("Banana", new
    FoodNutrition(89, 1.1));
        FOOD_NUTRITION_MAP.put("Chicken", new
    FoodNutrition(239,27.3));
        FOOD_NUTRITION_MAP.put("Rice", new
    FoodNutrition(130, 2.7));
    // Additional food items can be added here
    }
```

A.4 User Data Format

- **User Data in users.txt**:

```
    John Doe;25;180;75;Male
    Jane Smith;30;165;60;Female
```

- **Food Log Data Format**:

```
    Apple;150
    Banana;100
```

A.5 File Handling

- **Loading Users**:

```
  Java code
    public void loadUsers() {
        listModel.clear();
```

```
        userDataList.clear();
        try (BufferedReader reader = new
    BufferedReader(new FileReader("users.txt"))) {
            String line;
            while ((line = reader.readLine()) !=
    null) {
                String[] userData = line.split(";");
                listModel.addElement(userData[0]);
    // Add only the name to the list
                userDataList.add(userData); // Store
    all data for each user
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this,
    "Error loading users: " + e.getMessage());
        }
    }
```

- **Saving User Data**:

```
Java code
    private void saveUser(String name, String age,
    String height, String weight, String gender) {
        try (FileWriter writer = new
    FileWriter("users.txt", true)) {
            writer.write(name + ";" + age + ";" +
    height + ";" + weight + ";" + gender + "\n");
            JOptionPane.showMessageDialog(this,
```

```
    "User saved successfully!");
            mainWindow.loadUsers(); // Refresh the
    display in main window
            dispose(); // Close the window
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this,
    "Error saving user: " + e.getMessage());
        }
    }
```

A.6 GUI Layout

- **Main Window Layout**:
  - Contains a JList for displaying user names.
  - Buttons for opening profiles, adding, editing, deleting users, and exiting the application.
- **Profile Window Layout**:
  - Displays user details and includes fields for entering food type and amount.
  - A JTextArea for displaying the food log.
  - Labels for displaying total calories and protein.
- **New User/Edit User Window Layout**:
  - Text fields for name, age, height, weight, and gender.
  - Buttons for saving the new or edited user information.

References

1. **Oracle Java Documentation**
    - Comprehensive documentation on the Java programming language, covering all aspects of Java development, including core libraries, APIs, and the Java Virtual Machine (JVM).
    - URL: https://docs.oracle.com/javase/8/docs/api/
2. **Java File I/O Tutorial**
    - An in-depth guide on handling file input and output operations in Java, including reading from and writing to text files, using different classes and methods provided by the Java I/O package.
    - URL: https://docs.oracle.com/javase/tutorial/essential/io/
3. **Nutrition Data Sources**
    - Reliable sources of nutritional information used to populate the FOOD_NUTRITION_MAP in the application. These sources provide detailed nutritional values for a wide range of foods.
    - Example URL: https://www.nutritionix.com/
4. **Java Event Handling**
    - A guide on handling user actions and events in a Java Swing application. This includes adding ActionListeners to buttons and other components to define behavior when user interactions occur.
    - URL: https://docs.oracle.com/javase/tutorial/uiswing/events/
5. **Java GridLayout and BorderLayout**
    - Documentation and tutorials on using layout managers like GridLayout and BorderLayout in Java Swing to arrange components in a container. These layout managers are crucial for designing responsive and organized GUIs.
    - URL: https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html

- URL: https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html

6. **Java Exceptions and Error Handling**
    - A comprehensive guide on handling exceptions and errors in Java, including try-catch blocks and exception hierarchy. This is essential for managing file I/O operations and ensuring robust error handling in the application.
    - URL: https://docs.oracle.com/javase/tutorial/essential/exceptions/

These references provide the necessary information and resources that guided the development and implementation of the Java GUI application for managing user profiles and logging food intake.