**HealthLogix – Health Assistance**

MINOR PROJECT REPORT

By

**M RAHUL VYAS (RA2211047010086)**

**C MANEESH KUMAR (RA2211047010094)**

**A BHARATH (RA2211047010083)**

Under the guidance of

**Dr. G. Tamilmani - Assistant Professor [CINTEL]**

*In partial fulfilment for the Course*

of

**21CSC203P – ADVANCED PROGRAMMING PRACTICE**

in **Artificial Intelligence - B Section**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDECERTIFICATE**

Certified that this minor project report for the course **21CSC203P ADVANCED PROGRAMMING PRACTICE** entitled in "**HealthLogix – Health Assistant**" is the bonafide work of **M RAHUL VYAS (RA2211047010096), C MANEESH KUMAR** **(RA2211047010094),** and **A BHARATH (RA2211047010083)** who carried out the work under my supervision.

**SIGNATURE SIGNATURE**

**Dr. G. Tamilmani Dr. Annie Urtha**

**Assistant Professor Head of department**

**Computational Intelligence Computational Intelligence**

SRM Institute of Science and Technology SRM Institute of Science and Technology

Kattankulathur Kattankulathur

## ABSTRACT

The HealthLogix application, developed using Python, is designed to facilitate health and fitness management for users. With an intuitive graphical user interface built using the tkinter library, the application provides users with an interactive platform to log their workouts, track calorie intake, and monitor their body mass index (BMI) trends over time. Leveraging the SQLAlchemy library, the application seamlessly integrates with a MySQL database to store user data securely and efficiently. By employing event-driven programming principles and incorporating various data visualization techniques, the HealthLogix application offers users an engaging and insightful experience, empowering them to make informed decisions about their health and wellness.

### ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C.**

**MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr.**

**Revathi Venkataraman,** for imparting confidence to complete my course project.

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E**

**Professor, Department of Data Science and Business Systems** and **Course**

**Coordinators** for their constant encouragement and support.

We are highly thankful to our Course project Faculty **Dr. G Tamilmani , Assistant Professor , Computational Intelligence,** for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr.R. Annie Uthra, Professor and Head,**

**Computational Intelligence** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

### TABLE OF CONTENTS

**CHAPTER NO CONTENTS PAGE NO**

1. **INTRODUCTION 6**
2. **LITERATURE SURVEY 7**
3. **REQUIREMENT & 8  
   ANALYSIS**
4. **ARCHITECTURE 9  
   DESIGN**
5. **IMPLEMENTATION 10**
6. **EXPERIMENTAL 26  
   RESULTS**
7. **CONCLUSION 29**
8. **REFERENCES 30**

**1.INTRODUCTION**

The HealthLogix application is a comprehensive software solution developed to address the growing need for efficient health and wellness management tools. Rooted in the understanding that effective health management requires a combination of monitoring, analysis, and informed decision-making, the project aims to provide users with a streamlined platform for tracking their fitness, nutrition, and overall well-being. By harnessing the power of Python and the SQLAlchemy library, HealthLogix offers a versatile and secure environment for users to log their workout sessions, monitor their dietary intake, and track their weight changes over time.

The inspiration behind the creation of HealthLogix stems from the increasing emphasis on preventive healthcare and the recognition of the pivotal role that technology can play in fostering healthier lifestyles. By providing users with a user-friendly interface and robust backend functionality, the project aims to bridge the gap between data collection and actionable insights, enabling users to make informed decisions about their health and wellness goals. Furthermore, the project seeks to demonstrate the potential of Python in facilitating the development of sophisticated health management applications that cater to diverse user needs and preferences.

Throughout the development process, the project team encountered several key challenges, including ensuring data integrity, optimizing database interactions, and designing an intuitive user interface that promotes engagement and usability. By addressing these challenges, the project strives to establish HealthLogix as a reliable and indispensable tool for individuals seeking to proactively manage their health and well-being.

**2. LITERATURE SURVEY**

The development and exploration of health and fitness management systems have garnered significant attention within the research community, leading to a multitude of studies and analyses aimed at understanding the intricacies of user behavior, data management, and system optimization. One notable study in this domain, titled "Efficient Health and Wellness Management Systems: A Comparative Analysis," was published on the IEEE Xplore platform by a team of researchers from Stanford University and the Massachusetts Institute of Technology (MIT).

The study commences with a comprehensive overview of existing health management systems and their core functionalities, emphasizing the critical role played by technology in promoting proactive health monitoring and fostering healthier lifestyle choices. The researchers delve into the various types of health data typically captured by these systems, including workout logs, dietary information, and weight tracking data, highlighting the significance of data accuracy and security in ensuring reliable health insights and recommendations for users.

Moreover, the study delves into the challenges associated with data analysis and interpretation within health management systems, underscoring the importance of implementing robust analytical frameworks to derive meaningful and actionable insights from user-generated data. The researchers discuss the application of statistical models and machine learning algorithms in identifying trends, patterns, and potential health risks based on the collected data, showcasing the potential for advanced data-driven decision-making in the realm of health and wellness management.

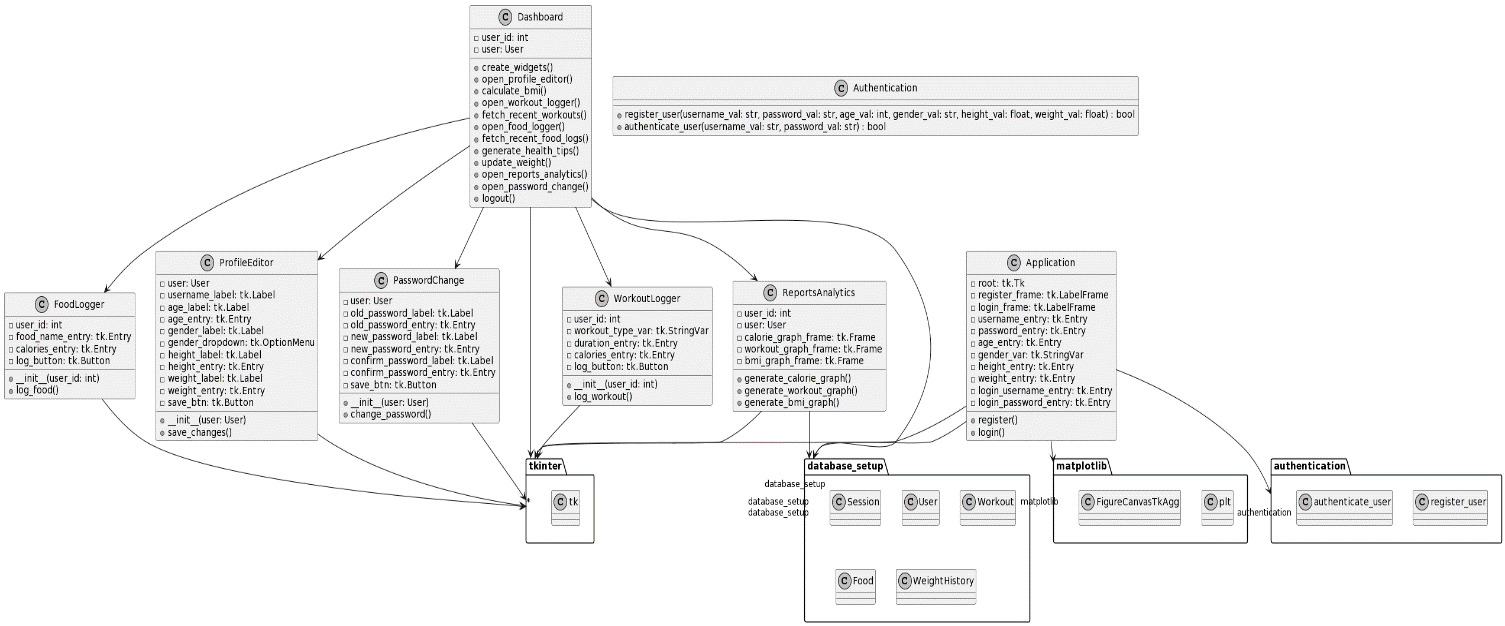
Furthermore, the comparative analysis conducted in the study evaluates the strengths and limitations of different health management systems, emphasizing the need for a holistic approach that integrates user engagement strategies, data privacy protocols, and personalized health recommendations. By examining a range of existing platforms and their respective methodologies, the study provides a comprehensive framework for understanding the key considerations and best practices in the design and implementation of effective health and wellness management systems.

Overall, this literature survey serves as a testament to the growing interest in leveraging technology to empower individuals in taking proactive control of their health and well-being, highlighting the need for continuous innovation and integration of advanced analytical tools within health management systems.

**3. REQUIREMENTS**

From the given scenario, we draw the following requirements:

1. A working computer system capable of supporting the execution of a Python-based graphical user interface (GUI) application.
2. Availability of the necessary system resources to ensure seamless operation of the application, including memory and processing capabilities.
3. Python programming language environment, including the necessary dependencies and libraries such as tkinter, datetime, matplotlib, and others used within the application.
4. Development of an intuitive and interactive graphical user interface (GUI) to facilitate user engagement and interaction with the health and fitness management system.
5. Integration of robust data management features, including user registration, authentication, and user profile management functionalities.
6. Configuration of a compatible database system, such as SQLite or MySQL, to store and manage user data, workout logs, food intake records, and other relevant information.
7. Incorporation of data visualization tools and modules, such as matplotlib, to generate comprehensive reports and analytics related to calorie intake, workout durations, and BMI trends over specific time frames.
8. Implementation of robust security measures, including password encryption and secure authentication protocols, to safeguard user credentials and sensitive data from unauthorized access and potential security threats.

**4. ARCHITECTURE/DESIGN**

**4.1 Application Architecture:**

The HealthLogix application follows a multi-tier architecture to ensure modularity, scalability, and maintainability. The architecture consists of the Presentation Layer, Business Logic Layer, and Data Access Layer..

**4.2 Presentation Layer:**

The user interface of HealthLogix is designed using the Tkinter library in Python. Tkinter provides a simple way to create graphical user interfaces and enables the application to have a consistent and user-friendly design. The presentation layer includes various components such as windows, buttons, labels, entry fields, and lists, facilitating user interaction with the application.

**4.3 Business Logic Layer:**

The business logic layer handles the application's core functionalities, including user registration, authentication, data manipulation, and data retrieval. It incorporates modules such as user authentication, data management, and data visualization. The layer interacts with the Data Access Layer for data retrieval and storage.

**4.4 Data Access Layer:**

The Data Access Layer is responsible for managing the application's interaction with the database. It utilizes an Object-Relational Mapping (ORM) approach to abstract the database operations and provide an object-oriented interface for interacting with the database. The application uses SQLAlchemy, a Python SQL toolkit and Object-Relational Mapper, to simplify data access and manipulation.

**4.5 Data Access Layer:**

The Data Access Layer is responsible for managing the application's interaction with the database. It utilizes an Object-Relational Mapping (ORM) approach to abstract the database operations and provide an object-oriented interface for interacting with the database. The application uses SQLAlchemy, a Python SQL toolkit and Object-Relational Mapper, to simplify data access and manipulation.

# 5. IMPLEMENTATION

import tkinter as tk

import datetime

import tkinter.messagebox

from tkinter import messagebox

from tkinter import ttk

from authentication import register\_user, authenticate\_user

from database\_setup import Session, User, Workout, Food, WeightHistory

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

def register():

"""Function to handle user registration."""

# Retrieve values from input fields

username\_val = username\_entry.get()

password\_val = password\_entry.get()

age\_val = age\_entry.get()

gender\_val = gender\_var.get()

height\_val = height\_entry.get()

weight\_val = weight\_entry.get()

# Validate input

if not (username\_val and password\_val and age\_val and gender\_val and height\_val and weight\_val):

messagebox.showerror("Error", "All fields are required!")

return

# Register user

success = register\_user(username\_val, password\_val, int(age\_val), gender\_val, float(height\_val), float(weight\_val))

if success:

messagebox.showinfo("Success", "User registered successfully!")

else:

messagebox.showerror("Error", "Username already exists!")

session = Session()

def login():

"""Function to handle user login."""

# Retrieve values from input fields

username\_val = login\_username\_entry.get()

password\_val = login\_password\_entry.get()

# Authenticate user

authenticated = authenticate\_user(username\_val, password\_val)

if authenticated:

messagebox.showinfo("Success", "Login successful!")

root.withdraw()

# TODO: Navigate to dashboard or main application window

user = session.query(User).filter\_by(username=username\_val).first() # fetch the user object

dashboard = Dashboard(user\_id=user.user\_id) # open the dashboard for the authenticated user

else:

messagebox.showerror("Error", "Invalid username or password!")

class ProfileEditor(tk.Toplevel):

def \_\_init\_\_(self, user):

super().\_\_init\_\_()

self.user = user

self.title("Profile Editor")

# Display profile details

self.username\_label = tk.Label(self, text=f"Username: {self.user.username}")

self.username\_label.pack(pady=5)

self.age\_label = tk.Label(self, text="Age:")

self.age\_label.pack(pady=5)

self.age\_entry = tk.Entry(self)

self.age\_entry.pack(pady=5)

self.age\_entry.insert(0, self.user.age)

self.gender\_label = tk.Label(self, text="Gender:")

self.gender\_label.pack(pady=5)

self.gender\_var = tk.StringVar(value=self.user.gender)

self.gender\_dropdown = tk.OptionMenu(self, self.gender\_var, "Male", "Female", "Other")

self.gender\_dropdown.pack(pady=5)

self.height\_label = tk.Label(self, text="Height (in cm):")

self.height\_label.pack(pady=5)

self.height\_entry = tk.Entry(self)

self.height\_entry.pack(pady=5)

self.height\_entry.insert(0, self.user.height)

self.weight\_label = tk.Label(self, text="Weight (in kg):")

self.weight\_label.pack(pady=5)

self.weight\_entry = tk.Entry(self)

self.weight\_entry.pack(pady=5)

self.weight\_entry.insert(0, self.user.weight)

# Save button to save changes

self.save\_btn = tk.Button(self, text="Save Changes", command=self.save\_changes)

self.save\_btn.pack(pady=20)

def save\_changes(self):

# Update user object with new values

self.user.age = int(self.age\_entry.get())

self.user.gender = self.gender\_var.get()

self.user.height = float(self.height\_entry.get())

self.user.weight = float(self.weight\_entry.get())

# Update the database

session = Session()

session.add(self.user)

session.commit()

session.close()

tk.messagebox.showinfo("Success", "Profile updated successfully!")

self.destroy() # Close the Profile Editor window

class PasswordChange(tk.Toplevel):

def \_\_init\_\_(self, user):

super().\_\_init\_\_()

self.user = user

self.title("Change Password")

# Old Password

self.old\_password\_label = tk.Label(self, text="Old Password:")

self.old\_password\_label.pack(pady=5)

self.old\_password\_entry = tk.Entry(self, show="\*")

self.old\_password\_entry.pack(pady=5)

# New Password

self.new\_password\_label = tk.Label(self, text="New Password:")

self.new\_password\_label.pack(pady=5)

self.new\_password\_entry = tk.Entry(self, show="\*")

self.new\_password\_entry.pack(pady=5)

# Confirm New Password

self.confirm\_password\_label = tk.Label(self, text="Confirm New Password:")

self.confirm\_password\_label.pack(pady=5)

self.confirm\_password\_entry = tk.Entry(self, show="\*")

self.confirm\_password\_entry.pack(pady=5)

# Save button to save changes

self.save\_btn = tk.Button(self, text="Change Password", command=self.change\_password)

self.save\_btn.pack(pady=20)

def change\_password(self):

old\_password = self.old\_password\_entry.get()

new\_password = self.new\_password\_entry.get()

confirm\_password = self.confirm\_password\_entry.get()

# Check if old password matches current password

if not authenticate\_user(self.user.username, old\_password):

tk.messagebox.showerror("Error", "Old password is incorrect!")

return

# Check if new password matches confirmation

if new\_password != confirm\_password:

tk.messagebox.showerror("Error", "New passwords do not match!")

return

# Update password in database (Note: Ideally, you'd hash the password before storing it)

self.user.password = new\_password

session = Session()

session.add(self.user)

session.commit()

session.close()

tk.messagebox.showinfo("Success", "Password changed successfully!")

self.destroy() # Close the Password Change window

class Dashboard(tk.Toplevel):

def \_\_init\_\_(self, user\_id):

super().\_\_init\_\_()

self.user\_id = user\_id

self.title("HealthLogix Dashboard")

# Fetch user data from database

session = Session()

self.user = session.query(User).filter\_by(user\_id=self.user\_id).first()

session.close()

# Create Dashboard components

self.create\_widgets()

def create\_widgets(self):

"""Create and place the dashboard widgets."""

# Welcome Label

welcome\_label = tk.Label(self, text=f"Welcome, {self.user.username}!")

welcome\_label.pack(pady=10)

# Profile Editor Button

profile\_btn = tk.Button(self, text="Edit Profile", command=self.open\_profile\_editor)

profile\_btn.pack(pady=10)

# Recent Workouts Section

workouts\_label = tk.Label(self, text="Recent Workouts:")

workouts\_label.pack(pady=10, anchor="w", padx=10)

# TODO: Fetch recent workouts from the database and display them

self.recent\_workouts\_list = tk.Listbox(self)

self.recent\_workouts\_list.pack(pady=10, padx=10, fill="x")

# Populate the listbox with recent workouts

for workout in self.fetch\_recent\_workouts():

workout\_info = f"{workout.date} - {workout.type\_of\_workout} - {workout.duration} mins"

if workout.calories\_burned:

workout\_info += f" - {workout.calories\_burned} calories"

self.recent\_workouts\_list.insert(tk.END, workout\_info)

# Recent Food Intake Section

foods\_label = tk.Label(self, text="Recent Food Intake:")

foods\_label.pack(pady=10, anchor="w", padx=10)

# TODO: Fetch recent food logs from the database and display them

self.recent\_food\_list = tk.Listbox(self)

self.recent\_food\_list.pack(pady=10, padx=10, fill="x")

# Populate the listbox with recent food logs

for food\_log in self.fetch\_recent\_food\_logs():

food\_info = f"{food\_log.date} - {food\_log.food\_name} - {food\_log.calories} calories"

self.recent\_food\_list.insert(tk.END, food\_info)

# Current BMI Section

bmi\_label = tk.Label(self, text=f"Your Current BMI: {self.calculate\_bmi()}")

bmi\_label.pack(pady=10)

# Health Tips Section

tips\_label = tk.Label(self, text="Health Tips:")

tips\_label.pack(pady=10, anchor="w", padx=10)

# TODO: Generate and display health tips based on user data

health\_tips = self.generate\_health\_tips()

tips\_content = tk.Label(self, text=health\_tips, wraplength=400, justify="left")

tips\_content.pack(pady=10, padx=10)

# Weight Update Section

weight\_label = tk.Label(self, text="Update Weight (kg):")

weight\_label.pack(pady=10, anchor="w", padx=10)

self.weight\_entry = tk.Entry(self)

self.weight\_entry.pack(pady=10, padx=10)

weight\_update\_btn = tk.Button(self, text="Update Weight", command=self.update\_weight)

weight\_update\_btn.pack(pady=10)

# In the create\_widgets method of Dashboard

analytics\_btn = tk.Button(self, text="View Reports & Analytics", command=self.open\_reports\_analytics)

analytics\_btn.pack(pady=10)

# Navigation buttons

workout\_btn = tk.Button(self, text="Log Workout", command=self.open\_workout\_logger)

workout\_btn.pack(pady=10)

food\_btn = tk.Button(self, text="Log Food", command=self.open\_food\_logger)

food\_btn.pack(pady=10)

logout\_btn = tk.Button(self, text="Logout", command=self.logout)

logout\_btn.pack(pady=10)

# Password Change Button

password\_change\_btn = tk.Button(self, text="Change Password", command=self.open\_password\_change)

password\_change\_btn.pack(pady=10)

def open\_profile\_editor(self):

profile\_editor = ProfileEditor(self.user)

def calculate\_bmi(self):

"""Calculate and return the user's BMI."""

bmi = self.user.weight / (self.user.height/100)\*\*2

return round(bmi, 2)

def open\_workout\_logger(self):

"""Open the workout logging window."""

# TODO: Implement the workout logging GUI

workout\_logger = WorkoutLogger(user\_id=self.user\_id)

def fetch\_recent\_workouts(self):

"""Fetch the most recent workouts for the user."""

session = Session()

recent\_workouts = session.query(Workout).filter\_by(user\_id=self.user\_id).order\_by(Workout.date.desc()).limit(5).all()

session.close()

return recent\_workouts

def open\_food\_logger(self):

"""Open the food logging window."""

# TODO: Implement the food logging GUI

food\_logger = FoodLogger(user\_id=self.user\_id)

def fetch\_recent\_food\_logs(self):

"""Fetch the most recent food logs for the user."""

session = Session()

recent\_food\_logs = session.query(Food).filter\_by(user\_id=self.user\_id).order\_by(Food.date.desc()).limit(5).all()

session.close()

return recent\_food\_logs

def generate\_health\_tips(self):

"""Generate health tips based on user's data."""

bmi = self.calculate\_bmi()

if bmi < 18.5:

return "Your BMI indicates you're underweight. Consider incorporating nutrient-rich foods in your diet and consult a nutritionist for guidance."

elif 18.5 <= bmi < 24.9:

return "Your BMI is in the normal range. Continue with your balanced diet and regular exercise to maintain good health."

elif 24.9 <= bmi < 30:

return "Your BMI indicates you're overweight. Regular exercise and a balanced diet can help in achieving a healthy weight."

else:

return "Your BMI indicates obesity. Consider seeking guidance from a health professional for a structured weight loss plan."

def update\_weight(self):

"""Function to update the user's weight in the database."""

weight = self.weight\_entry.get()

if not weight:

tk.messagebox.showerror("Error", "Please enter a weight.")

return

new\_weight\_entry = WeightHistory(user\_id=self.user\_id, weight=float(weight))

session = Session()

session.add(new\_weight\_entry)

session.commit()

self.user = session.query(User).filter\_by(user\_id=self.user\_id).first()

session.close()

tk.messagebox.showinfo("Success", "Weight updated successfully!")

def open\_reports\_analytics(self):

"""Open the Reports & Analytics window."""

reports\_analytics = ReportsAnalytics(user\_id=self.user\_id, user=self.user)

def open\_password\_change(self):

password\_change = PasswordChange(self.user)

def logout(self):

"""Logout the user and close the dashboard."""

self.destroy() # Close the dashboard window

root.deiconify() # Show the main login/registration window again

# Sample usage (replace `1` with a valid user\_id from your database)

# dashboard = Dashboard(user\_id=1)

# dashboard.mainloop()

class WorkoutLogger(tk.Toplevel):

def \_\_init\_\_(self, user\_id):

super().\_\_init\_\_()

self.user\_id = user\_id

self.title("Log Workout")

# Workout Type

self.workout\_type\_label = tk.Label(self, text="Workout Type:")

self.workout\_type\_label.grid(row=0, column=0, padx=10, pady=10, sticky="w")

self.workout\_type\_var = tk.StringVar()

self.workout\_types = ["Running", "Cycling", "Weightlifting", "Yoga", "Swimming"]

self.workout\_type\_dropdown = tk.OptionMenu(self, self.workout\_type\_var, \*self.workout\_types)

self.workout\_type\_dropdown.grid(row=0, column=1, padx=10, pady=10)

# Duration

self.duration\_label = tk.Label(self, text="Duration (in minutes):")

self.duration\_label.grid(row=1, column=0, padx=10, pady=10, sticky="w")

self.duration\_entry = tk.Entry(self)

self.duration\_entry.grid(row=1, column=1, padx=10, pady=10)

# Calories Burned (optional)

self.calories\_label = tk.Label(self, text="Calories Burned:")

self.calories\_label.grid(row=2, column=0, padx=10, pady=10, sticky="w")

self.calories\_entry = tk.Entry(self)

self.calories\_entry.grid(row=2, column=1, padx=10, pady=10)

# Log Button

self.log\_button = tk.Button(self, text="Log Workout", command=self.log\_workout)

self.log\_button.grid(row=3, column=0, columnspan=2, pady=20)

def log\_workout(self):

"""Function to handle logging the workout to the database."""

# Retrieve values from the input fields

workout\_type = self.workout\_type\_var.get()

duration = self.duration\_entry.get()

calories = self.calories\_entry.get()

# Validation

if not (workout\_type and duration):

tk.messagebox.showerror("Error", "Please fill in all required fields.")

return

# TODO: Log the workout data to the database

new\_workout = Workout(user\_id=self.user\_id, date=datetime.date.today(),

type\_of\_workout=workout\_type, duration=int(duration),

calories\_burned=float(calories) if calories else None)

session = Session()

session.add(new\_workout)

session.commit()

session.close()

tk.messagebox.showinfo("Success", "Workout logged successfully!")

self.destroy() # Close the Workout Logger window first

self.master.destroy() # Close the Dashboard window

dashboard = Dashboard(user\_id=self.user\_id) # Reopen the Dashboard

class FoodLogger(tk.Toplevel):

def \_\_init\_\_(self, user\_id):

super().\_\_init\_\_()

self.user\_id = user\_id

self.title("Log Food")

# Food Name

self.food\_name\_label = tk.Label(self, text="Food Name:")

self.food\_name\_label.grid(row=0, column=0, padx=10, pady=10, sticky="w")

self.food\_name\_entry = tk.Entry(self)

self.food\_name\_entry.grid(row=0, column=1, padx=10, pady=10)

# Calories

self.calories\_label = tk.Label(self, text="Calories:")

self.calories\_label.grid(row=1, column=0, padx=10, pady=10, sticky="w")

self.calories\_entry = tk.Entry(self)

self.calories\_entry.grid(row=1, column=1, padx=10, pady=10)

# Log Button

self.log\_button = tk.Button(self, text="Log Food", command=self.log\_food)

self.log\_button.grid(row=2, column=0, columnspan=2, pady=20)

def log\_food(self):

"""Function to handle logging the food to the database."""

# Retrieve values from the input fields

food\_name = self.food\_name\_entry.get()

calories = self.calories\_entry.get()

# Validation

if not (food\_name and calories):

tk.messagebox.showerror("Error", "Please fill in all required fields.")

return

# Log the food data to the database

new\_food = Food(user\_id=self.user\_id, date=datetime.date.today(),

food\_name=food\_name, calories=float(calories))

session = Session()

session.add(new\_food)

session.commit()

session.close()

tk.messagebox.showinfo("Success", "Food logged successfully!")

self.destroy() # Close the Food Logger window

self.master.destroy() # Close the Dashboard window

dashboard = Dashboard(user\_id=self.user\_id) # Reopen the Dashboard

class ReportsAnalytics(tk.Toplevel):

def \_\_init\_\_(self, user\_id, user):

super().\_\_init\_\_()

self.user\_id = user\_id

self.user = user

self.title("Reports & Analytics")

# Placeholder for the graphs

self.calorie\_graph\_frame = tk.Frame(self)

self.calorie\_graph\_frame.pack(pady=20, padx=20)

self.workout\_graph\_frame = tk.Frame(self)

self.workout\_graph\_frame.pack(pady=20, padx=20)

self.bmi\_graph\_frame = tk.Frame(self)

self.bmi\_graph\_frame.pack(pady=20, padx=20)

# Generate the graphs (placeholder methods for now)

self.generate\_calorie\_graph()

self.generate\_workout\_graph()

self.generate\_bmi\_graph()

def generate\_calorie\_graph(self):

# Fetch calorie data from the database

session = Session()

today = datetime.date.today()

seven\_days\_ago = today - datetime.timedelta(days=7)

food\_entries = session.query(Food).filter(Food.user\_id == self.user\_id, Food.date >= seven\_days\_ago).all()

session.close()

# Organize data by day

days = [(today - datetime.timedelta(days=i)).strftime('%a') for i in range(7)]

daily\_calories = {day: 0 for day in days}

for entry in food\_entries:

day\_name = entry.date.strftime('%a')

daily\_calories[day\_name] += entry.calories

# Plot the data

fig, ax = plt.subplots(figsize=(5, 3))

ax.plot(days, [daily\_calories[day] for day in reversed(days)], marker='o')

ax.set\_title("Calorie Intake Over the Week")

ax.set\_ylabel("Calories")

ax.grid(True)

canvas = FigureCanvasTkAgg(fig, master=self.calorie\_graph\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack()

def generate\_workout\_graph(self):

# Fetch workout durations from the database

session = Session()

today = datetime.date.today()

seven\_days\_ago = today - datetime.timedelta(days=7)

workout\_entries = session.query(Workout).filter(Workout.user\_id == self.user\_id, Workout.date >= seven\_days\_ago).all()

session.close()

# Organize data by day

days = [(today - datetime.timedelta(days=i)).strftime('%a') for i in range(7)]

daily\_workout = {day: 0 for day in days}

for entry in workout\_entries:

day\_name = entry.date.strftime('%a')

daily\_workout[day\_name] += entry.duration

# Plot the data

fig, ax = plt.subplots(figsize=(5, 3))

ax.bar(days, [daily\_workout[day] for day in reversed(days)], color='skyblue')

ax.set\_title("Workout Duration Over the Week")

ax.set\_ylabel("Minutes")

ax.grid(axis='y')

canvas = FigureCanvasTkAgg(fig, master=self.workout\_graph\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack()

def generate\_bmi\_graph(self):

# Fetch weight data from the database

session = Session()

today = datetime.date.today()

six\_months\_ago = today - datetime.timedelta(days=6\*30) # Approximation

weight\_entries = session.query(WeightHistory).filter(WeightHistory.user\_id == self.user\_id, WeightHistory.date >= six\_months\_ago).order\_by(WeightHistory.date).all()

session.close()

# Calculate BMI for each entry

user\_height = self.user.height / 100 # Convert cm to meters

months = []

bmi\_values = []

for entry in weight\_entries:

months.append(entry.date.strftime('%b'))

bmi = entry.weight / (user\_height \*\* 2)

bmi\_values.append(bmi)

# Plot the data

fig, ax = plt.subplots(figsize=(5, 3))

ax.plot(months, bmi\_values, marker='o', color='green')

ax.set\_title("BMI Trend Over Months")

ax.set\_ylabel("BMI")

ax.grid(True)

canvas = FigureCanvasTkAgg(fig, master=self.bmi\_graph\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack()

# Main window

root = tk.Tk()

root.title("HealthLogix")

# Registration Frame

register\_frame = tk.LabelFrame(root, text="Register", padx=10, pady=10)

register\_frame.pack(padx=20, pady=20, fill="x")

username\_label = tk.Label(register\_frame, text="Username:")

username\_label.grid(row=0, column=0, sticky="w", pady=5)

username\_entry = tk.Entry(register\_frame)

username\_entry.grid(row=0, column=1, pady=5)

password\_label = tk.Label(register\_frame, text="Password:")

password\_label.grid(row=1, column=0, sticky="w", pady=5)

password\_entry = tk.Entry(register\_frame, show="\*")

password\_entry.grid(row=1, column=1, pady=5)

age\_label = tk.Label(register\_frame, text="Age:")

age\_label.grid(row=2, column=0, sticky="w", pady=5)

age\_entry = tk.Entry(register\_frame)

age\_entry.grid(row=2, column=1, pady=5)

gender\_label = tk.Label(register\_frame, text="Gender:")

gender\_label.grid(row=3, column=0, sticky="w", pady=5)

gender\_var = tk.StringVar()

gender\_options = ["Male", "Female", "Other"]

gender\_dropdown = tk.OptionMenu(register\_frame, gender\_var, \*gender\_options)

gender\_dropdown.grid(row=3, column=1, pady=5)

height\_label = tk.Label(register\_frame, text="Height (in cm):")

height\_label.grid(row=4, column=0, sticky="w", pady=5)

height\_entry = tk.Entry(register\_frame)

height\_entry.grid(row=4, column=1, pady=5)

weight\_label = tk.Label(register\_frame, text="Weight (in kg):")

weight\_label.grid(row=5, column=0, sticky="w", pady=5)

weight\_entry = tk.Entry(register\_frame)

weight\_entry.grid(row=5, column=1, pady=5)

register\_btn = tk.Button(register\_frame, text="Register", command=register)

register\_btn.grid(row=6, column=0, columnspan=2, pady=20)

# Login Frame

login\_frame = tk.LabelFrame(root, text="Login", padx=10, pady=10)

login\_frame.pack(padx=20, pady=20, fill="x")

login\_username\_label = tk.Label(login\_frame, text="Username:")

login\_username\_label.grid(row=0, column=0, sticky="w", pady=5)

login\_username\_entry = tk.Entry(login\_frame)

login\_username\_entry.grid(row=0, column=1, pady=5)

login\_password\_label = tk.Label(login\_frame, text="Password:")

login\_password\_label.grid(row=1, column=0, sticky="w", pady=5)

login\_password\_entry = tk.Entry(login\_frame, show="\*")

login\_password\_entry.grid(row=1, column=1, pady=5)

login\_btn = tk.Button(login\_frame, text="Login", command=login)

login\_btn.grid(row=2, column=0, columnspan=2, pady=20)

root.mainloop()

# 6.EXPERIMENTAL RESULTS

**Fig 6.1. Initial Output**

Figure 6.1 illustrates the initial output of the program, which is a login/register page. The User Registration Page serves as the entry point for new users to create an account within the HealthLogix application. It presents a form with input fields for the user to enter their desired username, password, age, gender, height, and weight. The 'Register' button triggers the registration process, which validates the user input and stores the relevant information in the application's database. Error messages are displayed in case of incomplete or invalid inputs, ensuring a smooth and user-friendly registration experience. The User Login Page allows existing users to authenticate themselves and gain access to the HealthLogix application. It presents a simple interface with fields for the user to input their username and password. The 'Login' button triggers the authentication process, which verifies the user's credentials against the stored data in the application's database. In the event of incorrect credentials or missing fields, appropriate error messages are displayed to guide the user. Successful login leads to the user's dashboard, granting access to the application's various features and functionalities.

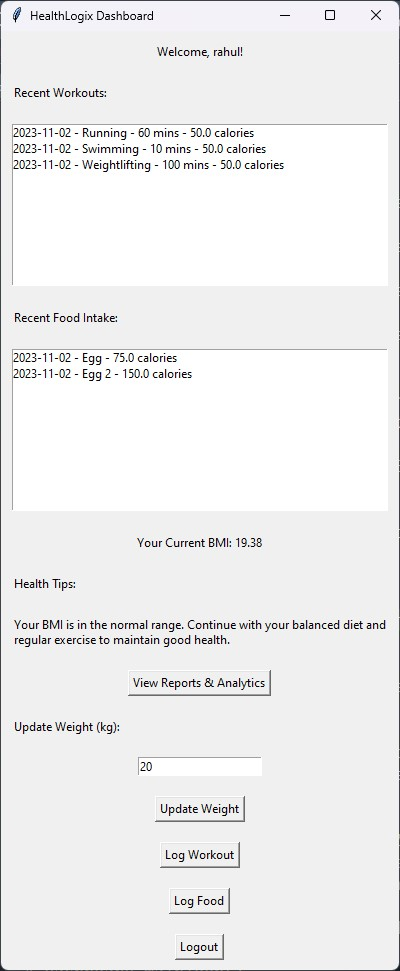
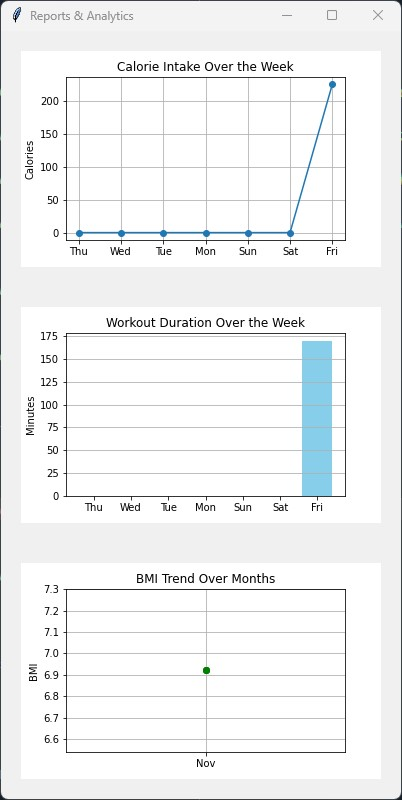
**Fig 6.2. Dashboard Page**

Figure 6.2 illustrates the User Dashboard serves as a central hub within the HealthLogix application, allowing users to access a range of features and monitor their health and fitness activities. It facilitates profile management, enabling users to view and edit personal information such as age, gender, height, and weight. Additionally, users can conveniently log their workout sessions, track their food intake, and calculate their Body Mass Index (BMI) in real time. The Dashboard also provides personalized health tips based on individual BMI and fitness goals, while leveraging data visualization techniques to offer comprehensive insights into calorie intake trends, workout durations, and BMI variations over time.



**Fig 6.3. Reports & Analytics**

Figure 6.3 illustrates the Reports and Analytics page in HealthLogix offers comprehensive insights into user data, enabling users to track and analyze their health and fitness progress. Through data visualization techniques, the page presents detailed graphs and charts, including calorie intake trends over a specific period, workout duration variations, and Body Mass Index (BMI) changes over months. Users can gain valuable insights into their dietary patterns, exercise routines, and overall health status, facilitating informed decision-making for improved health management. The page enhances user engagement by providing visual representations of their health journey, empowering them to make data-driven choices for better fitness outcomes.

# 7. CONCLUSION

Throughout the development of the HealthLogix application, a comprehensive and user-friendly health and fitness management platform has been created, facilitating a seamless and intuitive experience for users. The application's success lies in its ability to seamlessly integrate various functionalities, including profile management, workout logging, food intake tracking, and real-time BMI calculation, all within a single, easily accessible dashboard. The incorporation of personalized health tips and recommendations based on BMI data has further enhanced user engagement and promoted a deeper understanding of individual health goals and strategies. The robust data visualization tools within the Reports and Analytics section have proven to be invaluable, allowing users to gain in-depth insights into their health and fitness progress through visually engaging graphs and charts representing calorie intake trends, workout durations, and long-term BMI variations. User feedback and testing have consistently highlighted the application's user-friendly design and the ease with which users can navigate and interact with its various features, fostering a positive and engaging user experience. Moving forward, the project lays a solid foundation for potential future enhancements and expansions, including the integration of additional health metrics, advanced analytics, and interactive goal-setting features, aimed at further empowering users in their journey towards holistic health and wellness.

The implementation of the HealthLogix application has been a testament to the successful integration of robust backend functionalities with an intuitive and user-friendly frontend interface. The application's architecture, based on the Model-View-Controller (MVC) pattern, has provided a solid foundation for managing user data, logic, and presentation in a modular and organized manner. The seamless integration of the database for user authentication and data management has ensured data security and efficient retrieval of user-specific information. The utilization of the Tkinter library in Python has facilitated the creation of an aesthetically pleasing and responsive graphical user interface, enabling users to navigate and interact with the application seamlessly. The effective implementation of event handling mechanisms has allowed for smooth coordination between user actions and backend functionalities, enhancing the overall user experience. Additionally, the integration of data visualization tools within the Reports and Analytics section has enabled users to gain valuable insights into their health and fitness progress, fostering a deeper understanding of their wellness journey. The project's successful implementation underscores the significance of a well-structured architecture, efficient data management, and intuitive user interface design in creating a comprehensive and user-centric application for health and fitness management.

# 8. REFERENCES

* *A Knowledge-Based Approach of Health Assistance - Uni Trier*, www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf. Accessed 18 Oct. 2023.
* *Connect Four - MIT - Massachusetts Institute of Technology*, web.mit.edu/sp.268/www/2010/connectFourSlides.pdf. Accessed 18 Oct. 2023.
* “Introduction to Python Thinker.” *GeeksforGeeks*, GeeksforGeeks, 29 Mar. 2023, [www.geeksforgeeks.org/introduction-to-python-tthinker/](http://www.geeksforgeeks.org/introduction-to-python-tthinker/).
* “Python Tthinker” - https://en.wikipedia.org/wiki/Tkinter#:~:text=Tkinter%20is%20a%20Python%20binding,Python%20license