HEALTH TRACKING SYSTEM

A MINI PROJECT REPORT

Submitted by

C.P.AMIRDHA SUBA (9517202209002)

M.A.M.ATCHAYA DURGA (9517202209007)

M.HARINY (9517202209018)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

ANNA UNIVERSITY: CHENNAI 600 025

DECEMBER 2023

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"HEALTH TRACKING SYSTEM"** is the bonafide work of **"C P AMIRDHA SUBA(9517202209002) M A M ATCHAYA DURGA (9517202209007),M HARINY(9517202209018) "** who carried out the mini project work under my supervision.

SIGNATURE

Mrs.D.Monica Seles B.Tech,M.Tech

Assistant Professor

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi – 626 005

Virudhunagar District.

SIGNATURE

Dr.J.Angela Jennifa Sujana, M.E.,Ph.D

Professor & Head

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College,

Sivakasi – 626 005.

Virudhunagar District.

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"HEALTH TRACKING SYSTEM"** is the bonafide work of **"C P AMIRDHA SUBA(9517202209002) M A M ATCHAYA DURGA (9517202209007),M HARINY(9517202209018) "** who carried out the mini project work under my supervision.

SIGNATURE

Ms.K.Paul Mathi Priyanka B.E M.E

Assistant Professor

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College

Sivakasi – 626 005

Virudhunagar District.

SIGNATURE

Dr.J.Angela Jennifa Sujana, M.E.,Ph.D

Associate Professor(SG) & Head

Artificial Intelligence and Data Science

Mepco Schlenk Engineering College,

Sivakasi – 626 005.

Virudhunagar District.

TABLE OF CONTENTS

ABSTRACT

This is a Health Information System that utilizes the Tkinter library for the graphical user interface and interacts with a MySQL database to store and manage user health information.

The program begins by establishing a connection to a MySQL database named "project" with the root user and a specified password. Two tables, user_details and biological_details, are created in the database to store user information, including credentials, physical details, and health metrics.Two triggers, after_insert_user_details and after_delete_user_details, are created to handle automatic data insertion and deletion between the two tables when a user is added or removed.

Functions for user data manipulation:

- insert_user_details: Inserts hashed user details into the user_details table.
- get_user_details_by_id: Retrieves user details based on the provided user ID.
- delete_user_by_id: Deletes a user and their related records from both tables.

Functions for calculating health metrics:

- calculate_bmi: Calculates BMI based on height and weight.
- suggest_water_intake: Suggests daily water intake based on weight.
- suggest_diet_plan: Suggests a diet plan based on BMI.

Health status check functions are included:

- check_optimum_blood_pressure: Checks if blood pressure is within the normal range.

The graphical user interface is created using Tkinter, featuring entry fields for user input and buttons for actions such as submission, viewing details, and user deletion. Functions associated with button clicks include submit_button_clicked, view_details_button_clicked, delete_user_by_id, and view_my_details_button_clicked.

The script runs the create_tables, create_insert_trigger, and create_delete_trigger functions to initialize the database schema. The Tkinter GUI is then created, and the script enters a main loop until the user interacts with the GUI. Upon closure of the GUI, the database connection is closed.

Overall, the Health Information System allows users to input and manage health-related information, calculates and suggests health metrics, and provides a basic graphical interface for user interaction.

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

First and foremost we **praise and thank “The Almighty”,** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa Sujana M.E.,Ph.D.,** Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **Mrs.D.Monica Seles, B.Tech,M.Tech,** Assistant Professor, **Ms.K.Paul Mathi Priyanka, B.E,M.E**, Assistant Professor Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extremely thank our project coordinator **Dr.L.Prasika B.E,M.E,** Assistant Professor, **Dr.A.Shenbagarajan, M.E,Ph.D** Associate Professor(SG) Department of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We express our sincere gratitude to our esteemed project evaluators and reviewers for their valuable feedback and constructive criticism, which significantly contributed to the improvement of our project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We acknowledge and appreciate the support from our friends and peers who provided encouragement and motivation, creating a positive and collaborative environment throughout the project.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

CHAPTER 1
INTRODUCTION

1.1. OVERVIEW :

In the past, personal health monitoring was a relatively manual and sporadic process, often relying on occasional visits to healthcare professionals or self-reported observations. Individuals had limited access to real-time data about their health metrics, making it challenging to maintain a proactive approach to well-being. Health information was typically gathered during periodic check-ups, and the ability to detect early warning signs or trends was limited. Privacy concerns were also prevalent, as sharing health information for remote monitoring was not commonplace.

Now, with the advent of the HEALTH TRACKING SYSTEM (HTS), there's been a paradigm shift in how individuals engage with their health. Wearable devices equipped with sensors provide continuous monitoring, offering real-time insights into vital health parameters. The mobile application interface allows users to input personal details, set goals, and receive personalized recommendations, transforming health management into an interactive and informed experience. The HTS's advanced algorithms process data instantly, enabling early detection of potential health issues and empowering users to make timely lifestyle adjustments.

Privacy and security measures in the HTS address concerns from the past, assuring users that their sensitive health information is encrypted and confidential. Integration with healthcare professionals for remote monitoring fosters a collaborative approach to healthcare, allowing for personalized interventions based on real-time data. Regular updates and a feedback mechanism contribute to the system's adaptability and user satisfaction.

In essence, the transition from the old situation to the current Health Tracking System reflects a monumental leap in personalized health management. It marks a shift from reactive to proactive health monitoring, providing users with the tools and information needed to actively engage in their well-being and make informed decisions for a healthier and more fulfillinglife.

1.2. PROBLEM STATEMENT :

1.2.1. Limited Real-Time Insights:

•Traditional health monitoring methods often fail to provide users with real-time insights into their vital health metrics, hindering their ability to make timely and informed decisions about their well-being.

1.2.2. Lack of Personalization:

•Current health monitoring approaches lack personalized recommendations, leaving users without tailored insights and guidance that align with their unique health profiles and goals.

1.2.3. Inadequate Collaboration with Healthcare Professionals:

•The existing gap in seamlessly integrating health data from wearable devices with healthcare professionals hampers collaborative efforts, preventing timely interventions and personalized healthcare.

1.2.4. Privacy Concerns:

• Many health tracking systems lack robust encryption measures, raising concerns about the confidentiality and security of sensitive health information, thereby hindering user trust and adoption.

1.2.5. Data Divide Between Wearable Devices and Healthcare Providers:

•The disconnect in data sharing between wearable devices and healthcare providers limits the potential for timely interventions and comprehensive healthcare management, resulting in a fragmented healthcare ecosystem.

1.1. BLOCK DIAGRAM :

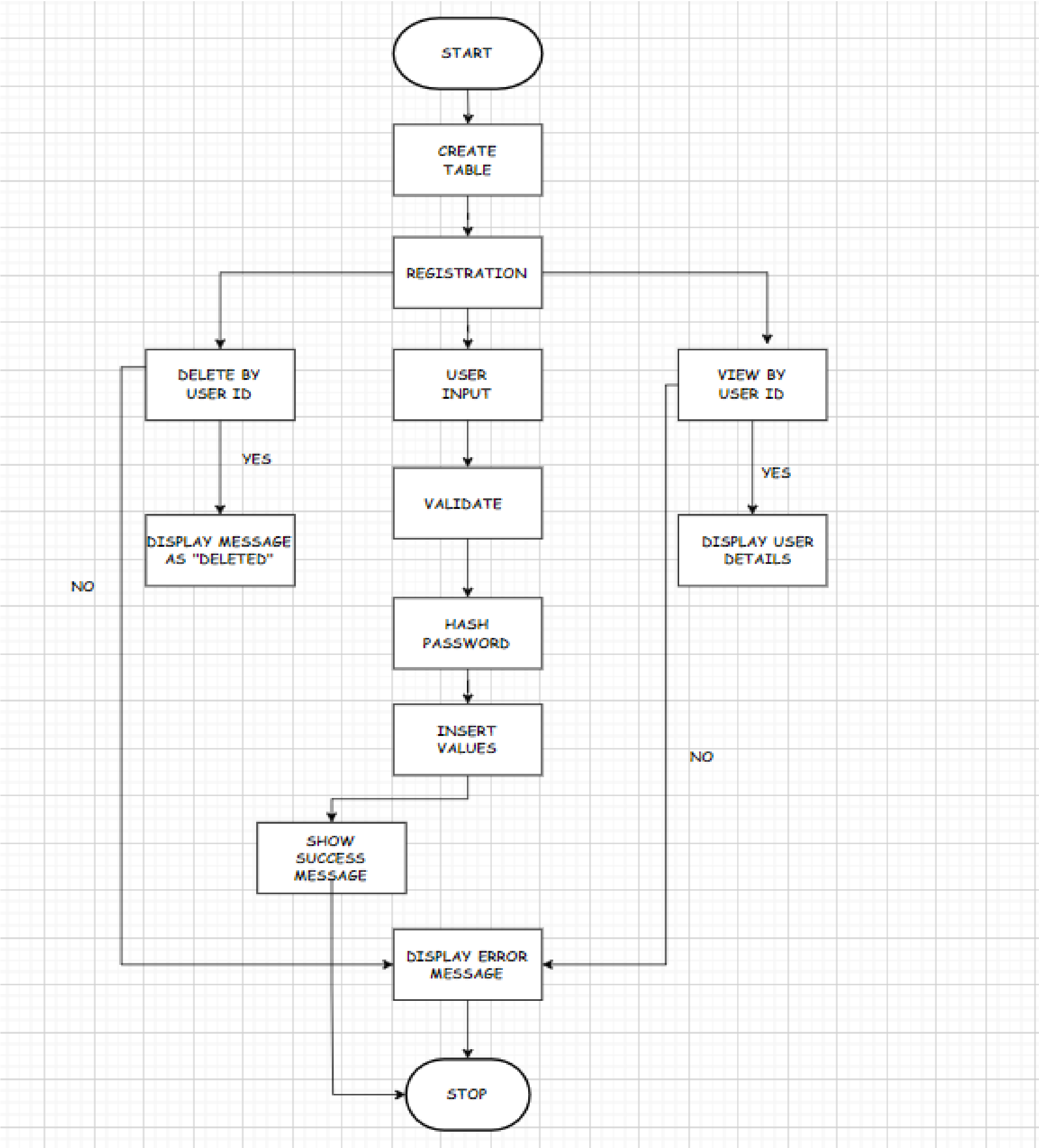


Figure 1.1 Block Diagram

1.2. DESCRIPTION

1.2.1. Database Connection:

The project begins by establishing a connection to a MySQL database hosted locally. The database is named "project," and connection details include the host, user, password, and database name.

1.2.2. Table Creation:

Two tables are created in the database using SQL queries: user_details and biological_details.
The user_details table stores user authentication details (username, password) and health-related information (height, weight, blood pressure, sugar level, BMI, water intake, diet plan).
The biological_details table is linked to user_details through a foreign key relationship.

1.2.3. Triggers:

Two triggers (after_insert_user_details and after_delete_user_details) are created to automate data synchronization between the user_details and biological_details tables when inserting or deleting records in the former.

1.2.4. Health Metric Calculations:

Functions are defined for calculating Body Mass Index (BMI), suggesting water intake based on weight, and suggesting a diet plan based on BMI.

1.2.5. Tkinter GUI:

The project features a GUI developed using the Tkinter library for Python.Entry fields and buttons are provided for users to input their health-related details, submit information, view details, and perform other actions.

1.2.6. User Data Manipulation:

Functions are implemented for inserting user details into the database, viewing user details, and deleting users by their ID.User data is inserted into the database, and relevant health metrics are calculated and stored.

1.2.7. Health Status Checks:

The project includes a function to check for optimum blood pressure based on specified ranges.BMI, water intake, blood pressure, and sugar level are categorized into different statuses (e.g., Normal, Not Normal, Optimum, Not Optimum).

1.2.8. Execution and Cleanup:

The script checks if it is the main module and, if so, executes functions to create tables and triggers.A Tkinter GUI is created, and the script enters a main loop until the GUI is closed by the user.After the GUI is closed, the script closes the cursor and the database connection.

CHAPTER 2
DATABASE DESIGN

2.1. E-R DIAGRAM

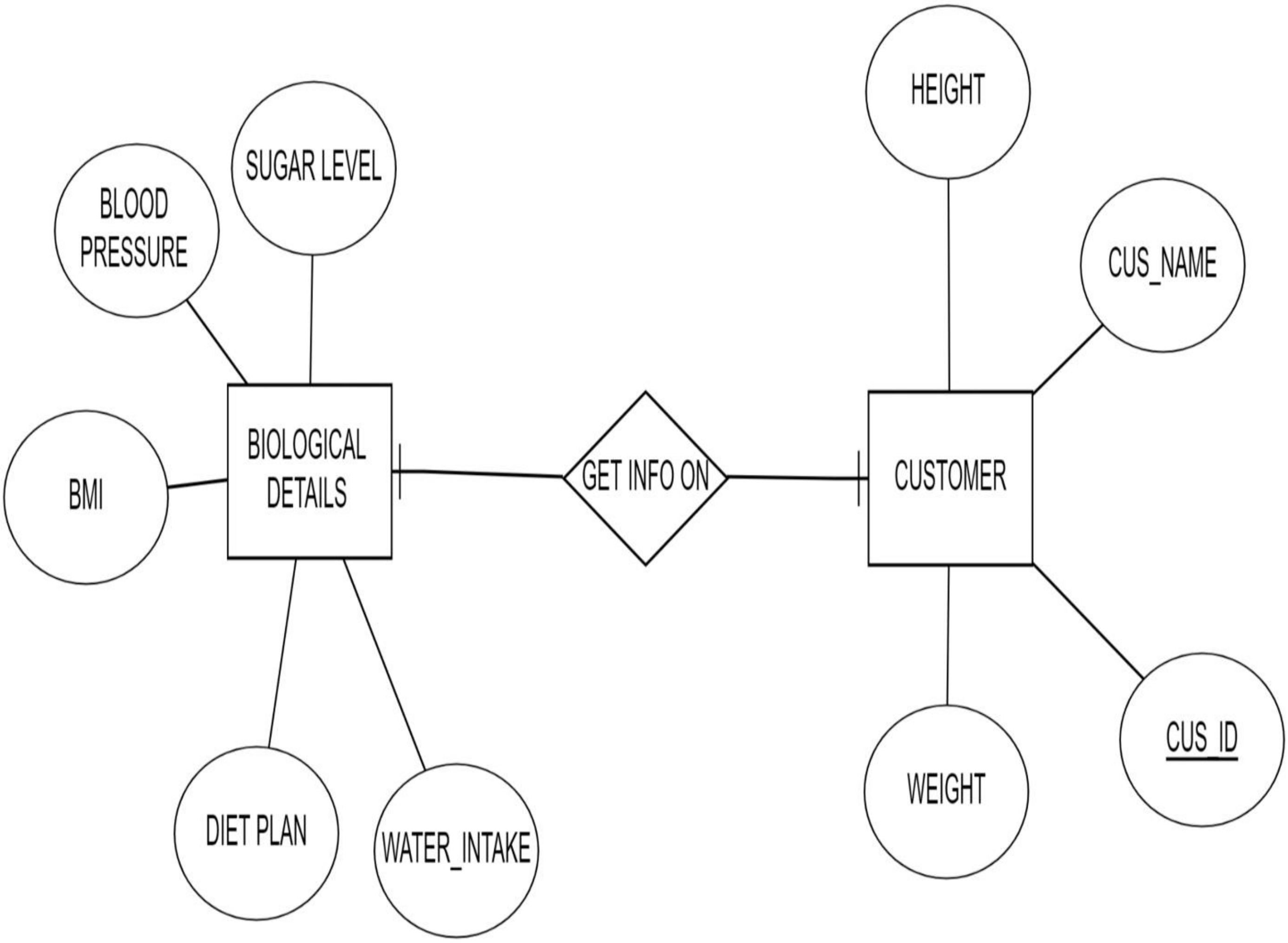


Figure 2.1 E-R Diagram

Attributes :

- TABLE1 (User_details) : User_id – Primary Key
- TABLE2 (Biological details)

2.2. RELATIONAL SCHEMA

```
+------------------+      +------------------------+
|   UserDetails    |      |  BiologicalDetails     |
+------------------+      +------------------------+
| UserID (PK)      |      | UserID (PK, FK)        |
| Password         |      | Height                 |
| Username         |      | Weight                 |
| Height           |----->| BloodPressure          |
| Weight           |      | SugarLevel             |
| BloodPressure    |      | BMI                    |
| SugarLevel       |      |                        |
| BMI              |      +------------------------+
| WaterIntake      |
| DietPlan         |
|                  |
+------------------+
```

Figure 2.2 Relational Schema

2.2.1. RELATIONSHIPS:

User_details and Biological_details :One To One Relation.

2.2.1.1. One to one relationship:

This relationship refers to a relationship between two tables in which each record in the first table (parent table) corresponds to exactly one record in the second table (child table), and vice versa.

Here one user cane store his /her one detail only.

2.3. NORMALIZATION

2.3.1. Initial Table Schema:

	userid	User name	password	height	weight	Blood pressure	Sugar_level	bmi	Water_intake	Diet_plan

Table 2.1 Initial Table

The Table satisfies 1-NF as there is no Multivalued Attribute.
It has composite Primary Key :userid,password

2.3.2. Partial Functional Dependencies:

- {userid,password}->username
- {userid,password}->Diet_plan
- {userid,password}->sugar_level
- {userid,password}->Blood pressure

But only userid can determine all these attributes.
Thus,decompose as

userid	username	Diet_plan	Sugar_level	Blood_pressure

Table 2.2 Users

- {userid,b_user_id}->bname
- {userid,b_user_id}->b_Sugar_level
- {userid,b_user_id}->b_bmi

But only b_user_id can determine all the above attributes.
Thus, decompose as

B_user_id	B_username	B_sugar_level	B_bmi

Table 2.3 Biological_details

userid	B_user_id

Table 2.4 New table

Here, b_user_id is a foreign key of table
userid is the primary key of the table3.

2.4. NORMALISED TABLE DESCRIPTION

Field	Type	Null	Key	Default	Extra
user_id	int	NO	PRI	NULL	auto_increment
username	varchar(50)	NO		NULL	
password	varchar(255)	NO		NULL	
height	float	YES		NULL	
weight	float	YES		NULL	
blood_pressure	varchar(10)	YES		NULL	
sugar_level	float	YES		NULL	
bmi	float	YES		NULL	
water_intake	float	YES		NULL	
diet_plan	text	YES		NULL	

Figure 2.3 user_details Schema

Field	Type	Null	Key	Default	Extra
user_id	int	YES	MUL	NULL	
height	float	YES		NULL	
weight	float	YES		NULL	
blood_pressure	varchar(10)	YES		NULL	
sugar_level	float	YES		NULL	
bmi	float	YES		NULL	

Figure 2.4 biological_details Schema

2.5. Query Explanation :

A query is a request for data or information from a database table or combination of tables.

2.5.1. CREATE :

The CREATE TABLE command creates a new table in the database.

2.5.1.1. Syntax:

CREATE TABLE <table_name> (<column1value> <data type>, <column2value> <data type>,…);

Queries:

```
CREATE TABLE IF NOT EXISTS user_details (
        user_id INT PRIMARY KEY,
        username VARCHAR(50) NOT NULL,
        password VARCHAR(255) NOT NULL,
        height FLOAT,
        weight FLOAT,
        blood_pressure VARCHAR(10),
        sugar_level FLOAT,
        bmi FLOAT,
        water_intake FLOAT,
        diet_plan TEXT
        )
CREATE TABLE IF NOT EXISTS biological_details (
        user_id INT,
        height FLOAT,
        weight FLOAT,
        blood_pressure VARCHAR(10),
        sugar_level FLOAT,
        bmi FLOAT,
        FOREIGN KEY (user_id) REFERENCES user_details(user_id) ON DELETE
CASCADE
        )
```

2.5.2. INSERT:

The INSERT INTO statement is used to insert single or multiple records into a table in the SQL Server database.

2.5.2.1. Syntax:

INSERT INTO <table_name> <column_list> VALUES (<column1value>, <column2value>,…);

Queries:
INSERT INTO user_details
(user_id, username, password, height, weight, blood_pressure, sugar_level, bmi, water_intake, diet_plan)

2.5.3. UPDATE:

The UPDATE statement is used to update single or multiple records in a table based on condition in the SQL Server database.

2.5.3.1. Syntax:

UPDATE <table_name> SET <column name> = <value> WHERE <condition>;

2.5.4. SELECT:

The SELECT statement is used to select specific rows or entire column list from the table based on the condition. The data returned is stored in a result table, called the result-set.

2.5.4.1. Syntax:

SELECT <column1>,<column2>….. FROM <table_name> WHERE <condition> ;

Queries:

"SELECT * FROM user_details;

“SELECT * FROM biological”;

CHAPTER 3

SYSTEM REQUIREMENTS

3.1. HARDWARE REQUIREMENTS:

3.1.1. Computer with Windows/Linux Operating System:

The project requires a computer with either the Windows or Linux operating system. This is the basic hardware requirement to run the software and execute the Python script. The specific hardware specifications (processor, memory, storage) can be relatively modest for a typical desktop or laptop computer.

3.2. SOFTWARE REQUIREMENTS:

3.2.1. IDLE (Python 3.10 64-bit):

IDLE stands for "Integrated Development and Learning Environment." It is the default integrated development environment that comes with Python. In this case, the project specifically requires Python 3.10 in a 64-bit version. This version of Python serves as the programming language for developing and running the Health Information System.

3.2.2. MySQL 8.0:

MySQL is a popular open-source relational database management system (RDBMS). The project utilizes MySQL version 8.0 as the backend database to store and manage health-related information. MySQL provides a structured and efficient way to organize, query, and manipulate data.

3.2.3. MysqlConnector Module:

MysqlConnector is a Python module that provides an interface to interact with MySQL databases. It allows the Python script to connect to the MySQL database, execute SQL queries, and manage the data stored in the database. This module facilitates the integration between the Python application and the MySQL database.

3.2.4. Tkinter Module:

Tkinter is the standard GUI toolkit that comes with Python. It is used to create the graphical user interface for the Health Information System. Tkinter provides a set of tools and widgets for building windows, buttons, entry fields, and other GUI components, making it easier to interact with the user.

CHAPTER 4
IMPLEMENTATION

The Project is implemented by using Python Tkinter Module as the front-end application while MySQL is used as Back-end for storing the information.

4.1. SOURCE CODE :

```
import mysql.connector
from tkinter import Tk, Label, Entry, Button, messagebox
import hashlib

db__connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Amirdhasuba1241",
    database="proj"
)
cursor = db__connection.cursor()


def create_tables():
    # Create user_details table
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS user_details (
        user_id INT PRIMARY KEY,
        username VARCHAR(50) NOT NULL,
        password VARCHAR(255) NOT NULL,
        height FLOAT,
        weight FLOAT,
        blood_pressure VARCHAR(10),
        sugar_level FLOAT,
        bmi FLOAT,
        water_intake FLOAT,
        diet_plan TEXT
    )
    """)

    # Create biological_details table
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS biological_details (
        user_id INT,
        height FLOAT,
        weight FLOAT,
        blood_pressure VARCHAR(10),
        sugar_level FLOAT,
        bmi FLOAT,
        FOREIGN KEY (user_id) REFERENCES user_details(user_id) ON DELETE CASCADE
```

```
    )
    """)

    db__connection.commit()


# ... (other code remains unchanged)

def create_insert_trigger():
    # Drop the trigger if it exists
    cursor.execute("DROP TRIGGER IF EXISTS after_insert_user_details")

    # Create the insert trigger
    cursor.execute("""
    CREATE TRIGGER after_insert_user_details
    AFTER INSERT ON user_details
    FOR EACH ROW
    BEGIN
        INSERT INTO biological_details (user_id, height, weight, blood_pressure, sugar_level, bmi)
        VALUES (NEW.user_id, NEW.height, NEW.weight, NEW.blood_pressure, NEW.sugar_level,
NEW.bmi);
    END
    """)


def create_delete_trigger():
    # Drop the trigger if it exists
    cursor.execute("DROP TRIGGER IF EXISTS after_delete_user_details")

    # Create the delete trigger with CASCADE option
    cursor.execute("""
    CREATE TRIGGER after_delete_user_details
    AFTER DELETE ON user_details
    FOR EACH ROW
    BEGIN
        DELETE FROM biological_details WHERE user_id = OLD.user_id;
    END
    """)


def insert_user_details(user_id, username, password, height, weight, blood_pressure, sugar_level):
    bmi = calculate_bmi(height, weight)
    water_intake = suggest_water_intake(weight)
    diet_plan = suggest_diet_plan(bmi)

    query = """
    INSERT INTO user_details
    (user_id, username, password, height, weight, blood_pressure, sugar_level, bmi, water_intake, diet_plan)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """
    cursor.execute(query, (
```

```
        user_id, username, hash_password(password), height, weight, blood_pressure, sugar_level, bmi,
water_intake,
        diet_plan))
    db__connection.commit()


def view_details_button_clicked():
    user_id = int(user_id_entry.get())
    user_details = get_user_details_by_id(user_id)
    if user_details:
        bmi_status = "Normal" if 18.5 <= user_details[7] < 24.9 else "Not Normal"
        water_intake_status = "Optimum" if user_details[8] >= suggest_water_intake(user_details[4]) else "Not
Optimum"
        blood_pressure_status = "Optimum" if check_optimum_blood_pressure(user_details[5]) else "Not
Optimum"
        sugar_level_status = "Optimum" if 80 <= user_details[6] <= 140 else "Not Optimum"

        details_message = (
            f"User ID: {user_details[0]}\n"
            f"Username: {user_details[1]}\n"
            f"Height: {user_details[3]}\n"
            f"Weight: {user_details[4]}\n"
            f"BMI: {user_details[7]} ({bmi_status})\n"
            f"Water Intake: {user_details[8]} ({water_intake_status})\n"
            f"Diet Plan: {user_details[9]}\n"
            f"Blood Pressure: {user_details[5]} ({blood_pressure_status})\n"
            f"Sugar Level: {user_details[6]} ({sugar_level_status})"
        )

        messagebox.showinfo("User Details", details_message)
    else:
        messagebox.showinfo("User Details", "User not found.")


def check_optimum_blood_pressure(blood_pressure):
    # Split the blood pressure into systolic and diastolic values
    systolic, diastolic = map(int, blood_pressure.split('/'))
    # Check if both systolic and diastolic values are within the normal range
    return 90 <= systolic <= 120 and 60 <= diastolic <= 80


def calculate_bmi(height, weight):
    bmi = weight / (height ** 2)
    return round(bmi, 2)


def suggest_water_intake(weight):
    water_intake = weight * 0.033
    return round(water_intake, 2)
```

```
def suggest_diet_plan(bmi):
    if bmi < 18.5:
        return "Underweight - Increase calorie intake with balanced nutrition."
    elif 18.5 <= bmi < 24.9:
        return "Normal weight - Maintain a balanced diet and exercise regularly."
    elif 25 <= bmi < 29.9:
        return "Overweight - Focus on portion control and increase physical activity."
    else:
        return "Obese - Consult with a healthcare professional for personalized advice."


def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()


def get_user_details_by_id(user_id):
    query = "SELECT * FROM user_details WHERE user_id = %s"
    cursor.execute(query, (user_id,))
    user_details = cursor.fetchone()
    return user_details


def submit_button_clicked():
    user_id = int(user_id_entry.get())
    username = username_entry.get()
    password = password_entry.get()
    height = float(height_entry.get())
    weight = float(weight_entry.get())
    blood_pressure = blood_pressure_entry.get()
    sugar_level = float(sugar_level_entry.get())

    insert_user_details(user_id, username, password, height, weight, blood_pressure, sugar_level)
    messagebox.showinfo("Success", "User details successfully stored!")


def view_details_button_clicked():
    user_id = int(user_id_entry.get())
    user_details = get_user_details_by_id(user_id)
    if user_details:
        bmi_status = "Normal" if 18.5 <= user_details[7] < 24.9 else "Not Normal"
        water_intake_status = "Optimum" if user_details[8] >= suggest_water_intake(user_details[4]) else "Not 
Optimum"
        blood_pressure_status = "Optimum" if "120/80" in user_details[5] else "Not Optimum"
        sugar_level_status = "Optimum" if 80 <= user_details[6] <= 140 else "Not Optimum"

        details_message = (
            f"User ID: {user_details[0]}\n"
            f"Username: {user_details[1]}\n"
            f"Height: {user_details[3]}\n"
            f"Weight: {user_details[4]}\n"
            f"BMI: {user_details[7]} ({bmi_status})\n"
```

```
            f"Water Intake: {user_details[8]} ({water_intake_status})\n"
            f"Diet Plan: {user_details[9]}\n"
            f"Blood Pressure: {user_details[5]} ({blood_pressure_status})\n"
            f"Sugar Level: {user_details[6]} ({sugar_level_status})"
        )

        messagebox.showinfo("User Details", details_message)
    else:
        messagebox.showinfo("User Details", "User not found.")


def delete_user_by_id():
    user_id = int(user_id_entry.get())

    try:
        # Delete related records from biological_details
        delete_biological_query = "DELETE FROM biological_details WHERE user_id = %s"
        cursor.execute(delete_biological_query, (user_id,))

        # Delete the user from user_details
        delete_user_query = "DELETE FROM user_details WHERE user_id = %s"
        cursor.execute(delete_user_query, (user_id,))

        db__connection.commit()
        messagebox.showinfo("Success", "User deleted successfully!")
    except mysql.connector.Error as err:
        # Handle the error
        messagebox.showerror("Error", f"Error deleting user: {err}")


def view_my_details_button_clicked():
    user_id = int(user_id_entry.get())
    user_details = get_user_details_by_id(user_id)
    if user_details:
        bmi_status = "Normal" if 18.5 <= user_details[7] < 24.9 else "Not Normal"
        water_intake_status = "Optimum" if user_details[8] >= suggest_water_intake(user_details[4]) else "Not
Optimum"
        blood_pressure_status = "Optimum" if "120/80" in user_details[5] else "Not Optimum"
        sugar_level_status = "Optimum" if 80 <= user_details[6] <= 140 else "Not Optimum"

        details_message = (
            f"User ID: {user_details[0]}\n"
            f"Username: {user_details[1]}\n"
            f"Height: {user_details[3]}\n"
            f"Weight: {user_details[4]}\n"
            f"BMI: {user_details[7]} ({bmi_status})\n"
            f"Water Intake: {user_details[8]} ({water_intake_status})\n"
            f"Diet Plan: {user_details[9]}\n"
            f"Blood Pressure: {user_details[5]} ({blood_pressure_status})\n"
            f"Sugar Level: {user_details[6]} ({sugar_level_status})"
        )
```

```
        messagebox.showinfo("My Details", details_message)
    else:
        messagebox.showinfo("My Details", "User not found.")


if __name__ == "__main__":
    create_tables()
    create_insert_trigger()
    create_delete_trigger()

    # Example: Insert an admin user with user_id 1, username "admin_username", password
"admin_password", and action_status "active"

    root = Tk()
    root.title("Health Information System")

    user_id_label = Label(root, text="User ID:")
    username_label = Label(root, text="Username:")
    password_label = Label(root, text="Password:")
    height_label = Label(root, text="Height (meters):")
    weight_label = Label(root, text="Weight (kg):")
    blood_pressure_label = Label(root, text="Blood Pressure:")
    sugar_level_label = Label(root, text="Sugar Level:")

    user_id_entry = Entry(root)
    username_entry = Entry(root)
    password_entry = Entry(root, show="*")
    height_entry = Entry(root)
    weight_entry = Entry(root)
    blood_pressure_entry = Entry(root)
    sugar_level_entry = Entry(root)

    submit_button = Button(root, text="Submit", command=submit_button_clicked)
    view_details_button = Button(root, text="View Details", command=view_details_button_clicked)
    delete_button = Button(root, text="Delete User", command=delete_user_by_id)
    view_my_details_button = Button(root, text="View My Details",
command=view_my_details_button_clicked)

    user_id_label.grid(row=0, column=0)
    user_id_entry.grid(row=0, column=1)
    username_label.grid(row=1, column=0)
    username_entry.grid(row=1, column=1)
    password_label.grid(row=2, column=0)
    password_entry.grid(row=2, column=1)
    height_label.grid(row=3, column=0)
    height_entry.grid(row=3, column=1)
    weight_label.grid(row=4, column=0)
    weight_entry.grid(row=4, column=1)
    blood_pressure_label.grid(row=5, column=0)
    blood_pressure_entry.grid(row=5, column=1)
```

```
sugar_level_label.grid(row=6, column=0)
sugar_level_entry.grid(row=6, column=1)

submit_button.grid(row=7, column=0, columnspan=2, pady=10)
view_details_button.grid(row=8, column=0, columnspan=2, pady=10)
delete_button.grid(row=9, column=0, columnspan=2, pady=10)

root.mainloop()

cursor.close()
db__connection.close()
```

4.2. RESULTS :

4.2.1. LOGIN WINDOW:

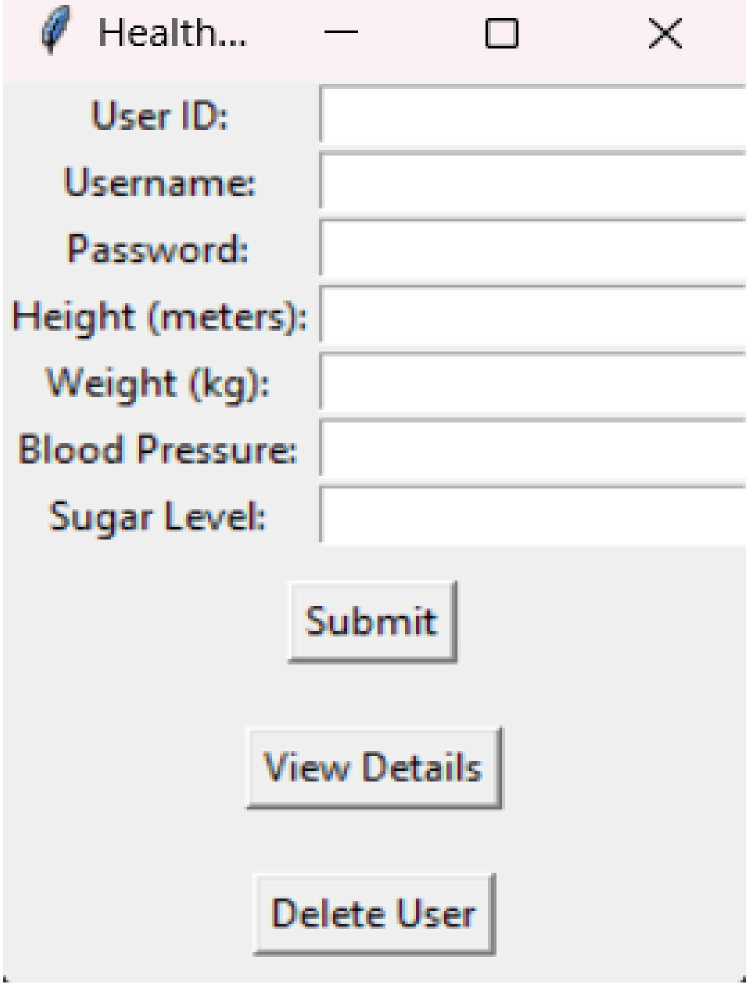


Figure 4.1 Login

4.2.2. AFTER SUBMISSION:

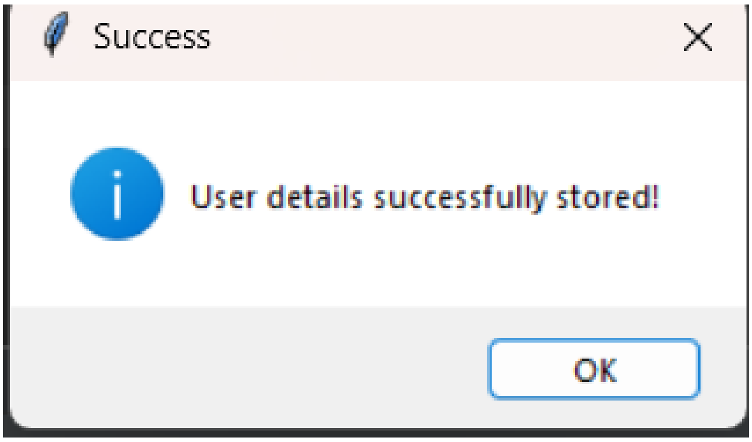


Figure 4.2 Submitted window

4.2.3. VIEW DETAILS:

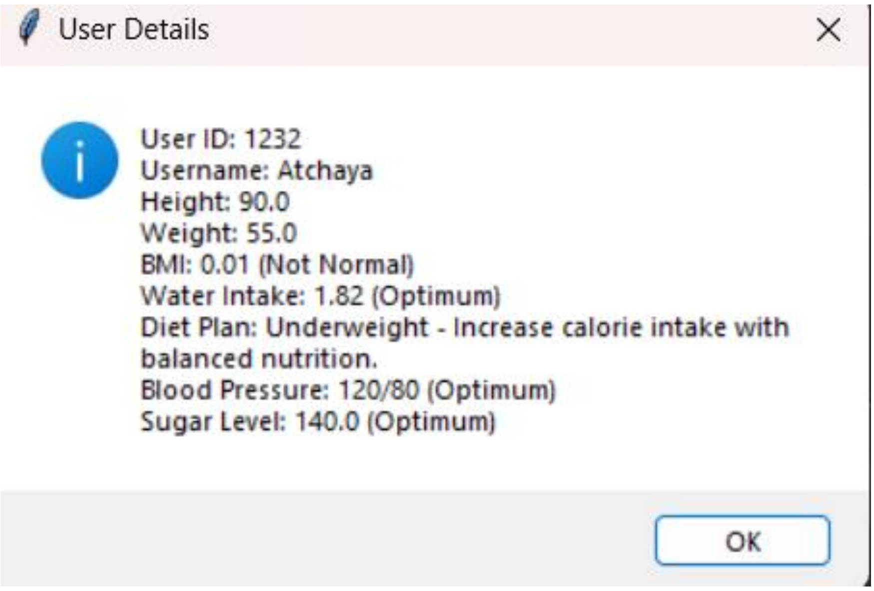
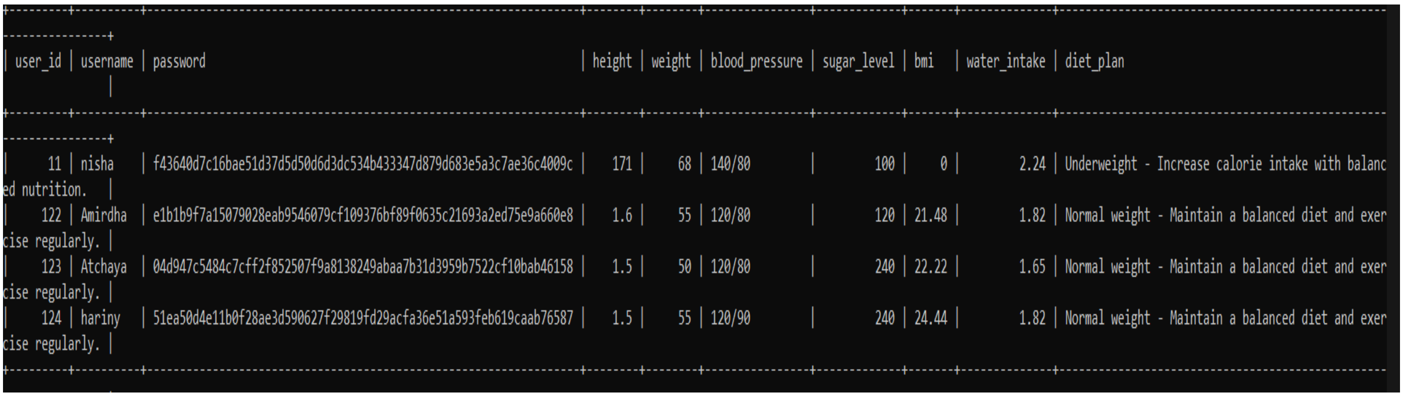


Figure 4.3 view details



user_id	username	password	height	weight	blood_pressure	sugar_level	bmi	water_intake	diet_plan
11	nisha	f43640d7c16bae51d37d5d50d6d3dc534b433347d879d683e5a3c7ae36c4009c	171	68	140/80	100	0	2.24	Underweight - Increase calorie intake with balanced nutrition.
122	Amirdha	e1b1b9f7a15079028eab9546079cf109376bf89f0635c21693a2ed75e9a660e8	1.6	55	120/80	120	21.48	1.82	Normal weight - Maintain a balanced diet and exercise regularly.
123	Atchaya	04d947c5484c7cff2f852507f9a8138249abaa7b31d3959b7522cf10bab46158	1.5	50	120/80	240	22.22	1.65	Normal weight - Maintain a balanced diet and exercise regularly.
124	hariny	51ea50d4e11b0f28ae3d590627f29819fd29acfa36e51a593feb619caab76587	1.5	55	120/90	240	24.44	1.82	Normal weight - Maintain a balanced diet and exercise regularly.

Figure 4.4 After inserting into user_details table

user_id	height	weight	blood_pressure	sugar_level	bmi
122	1.6	55	120/80	120	21.48
123	1.5	50	120/80	240	22.22
124	1.5	55	120/90	240	24.44
11	171	68	140/80	100	0

Figure 4.5 Values automatically inserted into biological_details table

user_id	username	password	height	weight	blood_pressure	sugar_level	bmi	water_intake	diet_plan
122	Amirdha	e1b1b9f7a15079028eab9546079cf109376bf89f0635c21693a2ed75e9a660e8	1.6	55	120/80	120	21.48	1.82	Normal weight - Maintain a balanced diet and exercise regularly.
123	Atchaya	04d947c5484c7cff2f852507f9a8138249abaa7b31d3959b7522cf10bab46158	1.5	50	120/80	240	22.22	1.65	Normal weight - Maintain a balanced diet and exercise regularly.
124	hariny	51ea50d4e11b0f28ae3d590627f29819fd29acfa36e51a593feb619caab76587	1.5	55	120/90	240	24.44	1.82	Normal weight - Maintain a balanced diet and exercise regularly.

Figure 4.6 Deleting in user_details table

user_id	height	weight	blood_pressure	sugar_level	bmi
122	1.6	55	120/80	120	21.48
123	1.5	50	120/80	240	22.22
124	1.5	55	120/90	240	24.44

Figure 4.7 Values automatically deleted in biological details table

CHAPTER 5
CONCLUSION

In conclusion, the Health Tracking System (HTS) heralds a new era in personal healthmanagement, transcending the limitations of traditional monitoring practices. By seamlessly integrating wearable devices and a user-centric mobile application, the HTS empowers individuals to actively participate in their health journey. The system's real-time insights, personalized recommendations, and advanced analytics redefine the way users engage with their well-being, shifting from reactive healthcare to a more proactive and preventive approach.

Furthermore, the HTS not only addresses the technological aspect of health monitoring but also prioritizes user privacy and data security through robust encryption measures. The collaborative nature of the system, enabling seamless integration with healthcare professionals for remote monitoring, signifies a holistic transformation in the healthcare landscape. As we witness the pervasive impact of the Health Tracking System, it becomes evident that the future of healthcare lies in personalized, technology-driven solutions that empower individuals to take charge of their health and make informed choices for a healthier and more fulfilling life.

CHAPTER 6
BIBLIOGRAPHY

- “Database Management System” – Elmasri and Navathe
- “Python Programming using Problem Solving Approach”-Reema Thareja
- https://www.geeksforgeeks.org/python-gui-tkinter/
- https://www.lovelycoding.org/grossary-management-system/
- https://www.slideshare.net/pujithaboggarapu/grocery-management-system
- https://www.geeksforgeeks.org/mysql-connector-python-module-in-python/