**Inspection Code Document**

**Team Name: Group 8**

**Core Functionalities:**

This project’s core functionalities focus on user management, health data collection, and data export. Below is an explanation of the essential modules and processes included in Phase 1 of the project.

**1. User Authentication and Authorization (Sign-Up, Sign-In, Google Sign-In)**

**Description**:

* This functionality handles user registration and login.
* Users can sign up by providing a username, password, and email, which is stored in a secure SQLite database.
* Existing users can log in using their credentials, and Google OAuth is integrated for Google account-based sign-in.

**Key Features**:

* User credentials stored securely in a database.
* Password inputs are masked to ensure privacy.
* Google OAuth implementation for alternative login.

python

Copy code

# Import necessary modules

import streamlit as st

from auth import sign\_up, sign\_in, google\_auth

import sqlite3

# Establish a database connection to store user credentials

conn = sqlite3.connect('users.db')

c = conn.cursor()

# Main App: Create the Streamlit UI components

st.title("Health Data Analytics Platform")

# A sidebar allows the user to select between signing up, signing in, or using Google sign-in

option = st.sidebar.selectbox("Choose action", ["Sign Up", "Sign In", "Google Sign-In"])

if option == "Sign Up":

# Sign up form to collect user details

username = st.text\_input("Username") # Text input for the username

password = st.text\_input("Password", type="password") # Password input (masked)

email = st.text\_input("Email") # Email input

if st.button("Sign Up"): # When user clicks "Sign Up", data is sent for registration

sign\_up(username, password, email) # Call the function to sign up the user

elif option == "Sign In":

# Sign in form for existing users

username = st.text\_input("Username") # Text input for the username

password = st.text\_input("Password", type="password") # Password input (masked)

if st.button("Sign In"): # When user clicks "Sign In", it triggers sign-in function

user = sign\_in(username, password) # Verify credentials

if user: # If user is authenticated, welcome them

st.success(f"Welcome back, {username}!")

else:

st.error("Invalid username or password.") # Error message for invalid credentials

elif option == "Google Sign-In":

# Google OAuth2-based authentication

st.write("Google Authentication")

if st.button("Login with Google"):

google\_auth() # Call the Google OAuth function

**Explanation**:

* sqlite3.connect('users.db'): Connects to the SQLite database that stores user information.
* st.sidebar.selectbox: Creates a sidebar with options to sign up, sign in, or use Google Sign-In.
* sign\_up(username, password, email): Adds new user information to the database.
* sign\_in(username, password): Authenticates an existing user by checking credentials against the database.
* google\_auth(): Handles OAuth2 flow for Google-based login.

**2. Database Management**

**Description**:

* The SQLite database stores user credentials and health data, ensuring data persistence across sessions.
* Two tables are used: users and health\_data.
  + users: Contains user credentials (username, password, and email).
  + health\_data: Stores personal health metrics for each user.

**Key Features**:

* A foreign key relationship between users and health\_data, ensuring data consistency.
* Health data tied to specific user IDs to enable personalized data retrieval.

python

Copy code

# Import necessary modules

import sqlite3

import os

import streamlit as st

# Determine the path of the current script and place the database in the same directory

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_)) # Get current script directory

db\_path = os.path.join(current\_dir, 'users.db') # Path to the database file

# Database setup

conn = sqlite3.connect(db\_path) # Connect to the SQLite database

c = conn.cursor() # Create a cursor object to interact with the database

# Create a user table if it doesn't exist already

c.execute('''

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT, # Auto-incremented user ID

username TEXT NOT NULL, # Username (must not be NULL)

password TEXT NOT NULL, # Password (hashed, must not be NULL)

email TEXT NOT NULL # Email (must not be NULL)

)

''')

conn.commit() # Commit the transaction to save changes

# Function for signing up new users

def sign\_up(username, password, email):

# Insert user credentials into the database

c.execute('INSERT INTO users (username, password, email) VALUES (?, ?, ?)', (username, password, email))

conn.commit() # Commit the transaction to save the new user

st.success("Account created successfully! You can now log in.") # Success message

# Function for signing in existing users

def sign\_in(username, password):

# Query the database to check if the username and password match

c.execute('SELECT \* FROM users WHERE username=? AND password=?', (username, password))

return c.fetchone() # Return the user record if found

**Explanation**:

* CREATE TABLE IF NOT EXISTS users: Ensures the users table exists to store user data.
* sign\_up: Inserts new user credentials (username, password, email) into the database.
* sign\_in: Queries the database to authenticate users based on their input.

**3. Health Data Collection**

**Description**:

* Authenticated users can input a wide range of health data, including weight, height, blood pressure, heart rate, glucose level, activity levels, etc.
* The collected data is stored in the health\_data table associated with the user’s ID.

**Key Features**:

* Inputs are gathered through Streamlit’s interactive interface, providing a user-friendly experience.
* Data inputs are validated to ensure proper formatting and ranges (e.g., numbers for weight, text for blood pressure).
* The functionality allows users to save their health data to the database for future reference and analysis.

python

Copy code

# Import necessary modules

import sqlite3

import streamlit as st

# Database setup for health data

conn = sqlite3.connect('users.db') # Connect to the user database

c = conn.cursor() # Create a cursor to interact with the database

# Create a health data table if it doesn't exist already

c.execute('''

CREATE TABLE IF NOT EXISTS health\_data (

user\_id INTEGER, # Foreign key referencing the user ID

weight REAL, height REAL, blood\_pressure TEXT, heart\_rate REAL,

body\_temp REAL, bmi REAL, glucose\_level REAL, cholesterol TEXT,

oxygen\_saturation REAL, activity\_level TEXT, dietary\_intake TEXT,

sleep\_patterns TEXT, medications TEXT, symptoms TEXT,

FOREIGN KEY (user\_id) REFERENCES users (id) # Link health data to users

)

''')

conn.commit() # Commit changes

# Function to collect health data from users

def collect\_health\_data(user\_id):

st.title("Health Data Collection") # Set the title of the page

# Collect various health data from the user via Streamlit input components

weight = st.number\_input("Weight (kg)", min\_value=0.0)

height = st.number\_input("Height (cm)", min\_value=0.0)

blood\_pressure = st.text\_input("Blood Pressure (Systolic/Diastolic)")

heart\_rate = st.number\_input("Heart Rate (BPM)", min\_value=0)

body\_temp = st.number\_input("Body Temperature (Celsius)", min\_value=0.0)

bmi = st.number\_input("Body Mass Index (BMI)", min\_value=0.0)

glucose = st.number\_input("Blood Glucose Level", min\_value=0.0)

cholesterol = st.text\_input("Cholesterol Levels (Total, HDL, LDL, Triglycerides)")

oxygen = st.number\_input("Oxygen Saturation (%)", min\_value=0)

activity = st.text\_area("Activity Level")

dietary = st.text\_area("Dietary Intake (Calories, Macronutrients)")

sleep = st.text\_input("Sleep Patterns (Hours)")

medications = st.text\_area("Medications (Name, Dosage, Frequency)")

symptoms = st.text\_area("Symptoms or Concerns")

if st.button("Save Data"): # Button to save the collected data

# Insert the collected data into the health\_data table

c.execute('''INSERT INTO health\_data (

user\_id, weight, height, blood\_pressure, heart\_rate, body\_temp, bmi, glucose\_level,

cholesterol, oxygen\_saturation, activity\_level, dietary\_intake, sleep\_patterns,

medications, symptoms) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)''',

(user\_id, weight, height, blood\_pressure, heart\_rate, body\_temp, bmi, glucose,

cholesterol, oxygen, activity, dietary, sleep, medications, symptoms))

conn.commit() # Commit the transaction

st.success("Health data saved successfully!") # Display success message

**4. Data Export (CSV, JSON)**

**Description**:

* Users can export their health data in different formats (CSV or JSON) for personal use or further analysis.
* Data export ensures that users have ownership of their data and can easily integrate it into other platforms.

**Key Features**:

* Download buttons in the interface allow users to export their data with a single click.
* The data export feature supports both CSV (for spreadsheets) and JSON (for web applications).

import pandas as pd

import json

# Function to export health data as CSV or JSON

def export\_data(user\_id):

c.execute('SELECT \* FROM health\_data WHERE user\_id=?', (user\_id,))

data = c.fetchall()

# Create a DataFrame from the fetched data

df = pd.DataFrame(data, columns=['UserID', 'Weight', 'Height', 'Blood Pressure', 'Heart Rate',

'Body Temp', 'BMI', 'Glucose', 'Cholesterol',

'Oxygen', 'Activity', 'Dietary', 'Sleep', 'Medications', 'Symptoms'])

# Button to download the data as CSV

st.download\_button(label="Download as CSV",

data=df.to\_csv(index=False),

file\_name="health\_data.csv",

mime='text/csv')

# Button to download the data as JSON

st.download\_button(label="Download as JSON",

data=json.dumps(df.to\_dict(orient='records')),

file\_name="health\_data.json",

mime='application/json')

**Explanation:**

1. **Imports**: Imports pandas for DataFrame operations and json for JSON serialization.
2. **Function Definition**: Defines export\_data(user\_id) to fetch and export health data.
3. **Data Retrieval**: Queries the database for health data associated with the specified user ID.
4. **DataFrame Creation**: Converts the fetched data into a Pandas DataFrame.
5. **CSV Download Button**: Creates a Streamlit button to download the data as a CSV file.
6. **JSON Download Button**: Creates a Streamlit button to download the data as a JSON file.

**Architectural Overview**

The architecture of this platform follows a **modular design**, where different components handle specific tasks such as user authentication, health data management, and data export. Below is a breakdown of the architecture:

**1. Front-End Layer (Streamlit Interface)**

* **Technology**: [Streamlit](https://streamlit.io/)
* **Responsibility**: Provides an interactive and user-friendly interface for users to sign in, collect data, and export results.
* **Components**:
  + **Sign Up/Sign In**: User input forms to enter and verify credentials.
  + **Health Data Collection**: Interactive inputs to collect user health data (e.g., weight, height, activity levels).
  + **Data Export**: Buttons to export collected data in CSV or JSON formats.

**2. Logic Layer**

* **Technology**: Python
* **Responsibility**: Contains the core functionalities, including authentication, data management, and export logic.
* **Components**:
  + **Authentication Functions**: Handle user registration and login processes.
  + **Health Data Functions**: Validate, store, and retrieve health data for each user.
  + **Export Functions**: Convert the health data into CSV and JSON formats for easy export.

**3. Data Layer (SQLite Database)**

* **Technology**: SQLite
* **Responsibility**: Persistent storage of user credentials and health data.
* **Components**:
  + **users Table**: Stores user credentials such as username, hashed password, and email.
  + **health\_data Table**: Stores individual user health metrics, linked to the users table by the user\_id.

**Flow of Information**

1. **User Interaction**: Users interact with the platform through the Streamlit interface (sign up, log in, data collection).
2. **Logic Processing**: Based on user actions, the system performs the necessary logic (authentication, data validation, and storage).
3. **Data Storage**: Data is saved in the SQLite database (users and health\_data tables) for long-term persistence.
4. **Data Export**: Users can download their health data in CSV or JSON format for personal use or integration with other systems.

**UML Design (High-Level Diagrams)**

**1. Class Diagram**

* **Classes**:
  + User: Stores information about users (username, password, email).
  + HealthData: Represents health metrics like weight, height, blood pressure, etc.
  + AuthManager: Handles sign-up and sign-in functionalities.
  + DataManager: Collects and stores health data.
  + ExportManager: Handles exporting health data in different formats.

**2. Sequence Diagram**

* **Example Use Case: Sign In & Data Collection**:
  1. User provides login credentials.
  2. AuthManager checks credentials against the users table.
  3. If authenticated, the user can enter health data through the DataManager.
  4. Health data is saved in the health\_data table.