# Infosys Springboard Virtual Internship

**“Project Name ” Developing chatbot on global wellness"**

**Milestone 1:- User authentication and Profile Management**

**Name - Raghvendra Pratap Singh**
**Date- 27-08-2025**

# Milestone 1:- User authentication and Profile Management

## Introduction

This project is a Global Wellness Chatbot System that provides a secure and interactive platform for user management and wellness tracking. The core focus of this phase is User Authentication and Profile Management, where users can create an account, log in securely, and update their personal wellness profiles.

The backend has been developed using FastAPI (Python) with SQLite as the database to persist user details. Security has been ensured through hashed passwords (bcrypt) and JWT-based authentication tokens. A clean and professional Streamlit UI has been integrated to allow users to interact with the system easily without needing manual API calls.

By completing this phase, we have established a solid backend + frontend foundation where wellness-related chatbot features can be integrated in the future.

## Objectives

The main objectives successfully achieved in this project phase are:

1. **Virtual Environment Setup**
   - Created an isolated Python virtual environment for dependency management.
   - Installed required packages such as FastAPI, Streamlit, Passlib, bcrypt, python-jose, and SQLite integration.

2. **User Authentication System**
   - Implemented **Signup API** for new user registration.
   - Implemented **Login API** that validates credentials and generates a secure **JWT token**.
   - Ensured **password security** by hashing with bcrypt.

3. **Profile Management**
   - Created endpoints to fetch user details and update personal wellness-related information (age group, language preference, wellness goals).
   - Designed a professional **Streamlit interface** for creating accounts, logging in, and updating profiles.

4. **Database Integration**
   - Used **SQLite** to store and manage user data persistently.
   - Automatically created `users` and `profiles` tables during app initialization.

5. **Security & Session Handling**
    - Protected routes with **JWT token authentication**.
    - Ensured session persistence in the frontend through Streamlit's state management.

6. **Frontend-Backend Integration**
    - Developed a simple, professional, and user-friendly interface using **Streamlit**.
    - Connected the frontend with the FastAPI backend using **HTTP requests** for signup, login, and profile management.

# Code Implementation

The project has been implemented using **FastAPI (Python)** as the backend, **SQLite** for persistent storage, and **Streamlit** as the frontend interface. Below are the key components of the implementation:

---

## 1 Database Setup (SQLite)

We use SQLite to store user authentication and profile details. On application startup, tables are created automatically if they don't exist.

***database.py***

```
import sqlite3

from pathlib import Path

DB_PATH = Path(__file__).resolve().parent / "app.db"

print(DB_PATH)

# One shared connection for simplicity

conn = sqlite3.connect(DB_PATH, check_same_thread=False)

conn.row_factory = sqlite3.Row

def init_db():

    with conn:

        conn.execute("""

        CREATE TABLE IF NOT EXISTS users (
```

```
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        email TEXT UNIQUE NOT NULL,
        password_hash TEXT NOT NULL,
        full_name TEXT,
        created_at TEXT DEFAULT CURRENT_TIMESTAMP
    );
    """)
    conn.execute("""
    CREATE TABLE IF NOT EXISTS profiles (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER UNIQUE NOT NULL,
        age_group TEXT,
        language TEXT,
        wellness_goals TEXT,
        updated_at TEXT DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY(user_id) REFERENCES users(id)
    );
    """)
def get_conn():
    return conn
```

---

## 2 User Authentication with JWT

We implemented **signup, login, and authentication** using **bcrypt for hashing** and **JWT tokens** for session management.

### *auth.py*

```
from datetime import datetime, timedelta, timezone
from typing import Optional
```

```
from jose import jwt, JWTError

from passlib.context import CryptContext

import os

from dotenv import load_dotenv

load_dotenv()

SECRET_KEY = os.getenv("JWT_SECRET", "change-me")

ALGORITHM = os.getenv("JWT_ALGORITHM", "HS256")

ACCESS_TOKEN_EXPIRE_MINUTES = int(os.getenv("JWT_EXPIRE_MINUTES", "60"))

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# ---------------- Password utils ----------------

def hash_password(plain: str) -> str:

    return pwd_context.hash(plain)

def verify_password(plain: str, hashed: str) -> bool:

    return pwd_context.verify(plain, hashed)


# ---------------- Access tokens (login) ----------------

def create_access_token(data: dict, expires_minutes: Optional[int] = None) -> str:

    to_encode = data.copy()

    expire = datetime.now(timezone.utc) + timedelta(

        minutes=expires_minutes or ACCESS_TOKEN_EXPIRE_MINUTES

    )

    to_encode.update({"exp": expire, "purpose": "access"})

    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def decode_token(token: str) -> Optional[dict]:

    try:

        return jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])

    except JWTError:
```

```
        return None


# ---------------- Reset tokens (forgot password) ----------------
def create_reset_token(email: str, expires_minutes: int = 10) -> str:
    """Short-lived token used only for password reset."""
    expire = datetime.now(timezone.utc) + timedelta(minutes=expires_minutes)
    to_encode = {"sub": email, "exp": expire, "purpose": "reset"}
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
def verify_reset_token(token: str) -> Optional[str]:
    """Return email if token is valid & for reset; otherwise None."""
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        if payload.get("purpose") != "reset":
            return None
        return payload.get("sub")
    except JWTError:
        return None
```

---

## 3️⃣ API Endpoints (FastAPI)

We created routes for **signup, login, and profile management**.

### *main.py*

```
from fastapi import FastAPI, Depends, HTTPException, status, Header, Body
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel, EmailStr, Field
from typing import Optional
import sqlite3
```

```
from .database import init_db, get_conn

from .schemas import UserCreate, UserLogin, Token, ProfileUpdate, ProfileOut

from .auth import (

    hash_password,

    verify_password,

    create_access_token,

    decode_token,

    create_reset_token,

    verify_reset_token,

)


app = FastAPI(title="Global Wellness Chatbot - Auth & Profiles")


# Allow Streamlit frontend

app.add_middleware(

    CORSMiddleware,

    allow_origins=["*"],  # tighten in production

    allow_credentials=True,

    allow_methods=["*"],

    allow_headers=["*"],

)


# Ensure tables exist on startup

init_db()


# ---------- DB Helpers ----------

def get_user_by_email(conn: sqlite3.Connection, email: str):
```

```
    cur = conn.execute("SELECT * FROM users WHERE email=?", (email,))
    return cur.fetchone()


def create_user(conn: sqlite3.Connection, email: str, password: str, full_name: Optional[str]):
    password_hash = hash_password(password)
    try:
        with conn:
            conn.execute(
                "INSERT INTO users (email, password_hash, full_name) VALUES (?, ?, ?)",
                (email, password_hash, full_name),
            )
    except sqlite3.IntegrityError:
        raise HTTPException(status_code=400, detail="Email already registered.")


def authenticate_user(conn: sqlite3.Connection, email: str, password: str):
    row = get_user_by_email(conn, email)
    if not row:
        return None
    if not verify_password(password, row["password_hash"]):
        return None
    return row


def get_profile(conn: sqlite3.Connection, user_id: int):
    cur = conn.execute("SELECT * FROM profiles WHERE user_id=?", (user_id,))
    return cur.fetchone()


def upsert_profile(conn: sqlite3.Connection, user_id: int, p: ProfileUpdate):
```

```
    existing = get_profile(conn, user_id)
    if existing:
        with conn:
            conn.execute("""
                UPDATE profiles
                SET age_group=?, language=?, wellness_goals=?, updated_at=CURRENT_TIMESTAMP
                WHERE user_id=?
            """, (p.age_group, p.language, p.wellness_goals, user_id))
    else:
        with conn:
            conn.execute("""
                INSERT INTO profiles (user_id, age_group, language, wellness_goals)
                VALUES (?, ?, ?, ?)
            """, (user_id, p.age_group, p.language, p.wellness_goals))


# ---------- Auth Dependency ----------
def get_current_user(authorization: Optional[str] = Header(None)):
    if not authorization or not authorization.lower().startswith("bearer "):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Missing bearer token.")
    token = authorization.split()[1]
    payload = decode_token(token)
    if not payload or payload.get("purpose") != "access" or "sub" not in payload:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid token.")
    return payload  # contains: {"sub": email, "uid": id, "purpose": "access"}


# ---------- Extra Schemas (only for this file) ----------
class PasswordResetRequest(BaseModel):
```

```
    email: EmailStr


class PasswordResetConfirm(BaseModel):
    token: str
    new_password: str = Field(min_length=6)


# ---------- Routes ----------
@app.post("/auth/signup", status_code=201)
def signup(user: UserCreate):
    conn = get_conn()
    if get_user_by_email(conn, user.email):
        raise HTTPException(status_code=400, detail="Email already registered.")
    create_user(conn, user.email, user.password, user.full_name)
    return {"message": "Account created. Please sign in."}


@app.post("/auth/login", response_model=Token)
def login(creds: UserLogin):
    conn = get_conn()
    row = authenticate_user(conn, creds.email, creds.password)
    if not row:
        raise HTTPException(status_code=400, detail="Invalid email or password.")
    token = create_access_token({"sub": row["email"], "uid": row["id"]})
    return {"access_token": token, "token_type": "bearer"}


@app.get("/auth/me", response_model=ProfileOut)
def me(user=Depends(get_current_user)):
    conn = get_conn()
```

```
    u = get_user_by_email(conn, user["sub"])
    p = get_profile(conn, u["id"])
    profile = {
        "email": u["email"],
        "full_name": u["full_name"],
        "age_group": p["age_group"] if p else None,
        "language": p["language"] if p else None,
        "wellness_goals": p["wellness_goals"] if p else None,
    }
    return profile


@app.put("/profile", response_model=ProfileOut)
def save_profile(update: ProfileUpdate, user=Depends(get_current_user)):
    conn = get_conn()
    u = get_user_by_email(conn, user["sub"])
    upsert_profile(conn, u["id"], update)
    p = get_profile(conn, u["id"])
    return {
        "email": u["email"],
        "full_name": u["full_name"],
        "age_group": p["age_group"],
        "language": p["language"],
        "wellness_goals": p["wellness_goals"],
    }


# ---------- Forgot Password Flow ----------
@app.post("/auth/request-password-reset")
```

```
def request_password_reset(payload: PasswordResetRequest):
    conn = get_conn()
    user = get_user_by_email(conn, payload.email)
    if not user:
        # For privacy you can still return 200 with generic message;
        # using 404 here so you see the difference in dev.
        raise HTTPException(status_code=404, detail="Email not found.")
    token = create_reset_token(payload.email, expires_minutes=10)
    # In production: email this token (magic link). For now, return it.
    return {"reset_token": token, "message": "Reset token created (valid for 10 minutes)."}


@app.post("/auth/reset-password")
def reset_password(payload: PasswordResetConfirm):
    email = verify_reset_token(payload.token)
    if not email:
        raise HTTPException(status_code=400, detail="Invalid or expired reset token.")
    if len(payload.new_password) < 6:
        raise HTTPException(status_code=400, detail="Password must be at least 6 characters.")
    conn = get_conn()
    user = get_user_by_email(conn, email)
    if not user:
        raise HTTPException(status_code=404, detail="User not found.")
    hashed = hash_password(payload.new_password)
    with conn:
        conn.execute("UPDATE users SET password_hash=? WHERE email=?", (hashed, email))
    return {"message": "Password reset successful. Please log in with your new password."}
```

## 4️⃣ Frontend (Streamlit)

The Streamlit UI provides a simple and professional frontend for account creation, login, and profile updates

### *app.py*

```
import streamlit as st

import httpx

API = "http://127.0.0.1:8000"

st.set_page_config(page_title="Global Wellness – Auth & Profile", page_icon="🧘", layout="centered")

# Simple brand header

st.markdown("<h1 style='text-align:center'>Global Wellness</h1>", unsafe_allow_html=True)

st.markdown("<p style='text-align:center;color:gray'>Sign in, manage your profile, and start chatting for better well-being.</p>", unsafe_allow_html=True)

st.divider()

# -------- Session state --------

if "token" not in st.session_state:

    st.session_state.token = None

if "show_reset" not in st.session_state:

    st.session_state.show_reset = False

if "reset_token" not in st.session_state:

    st.session_state.reset_token = None

def api_headers():

    if st.session_state.token:

        return {"Authorization": f"Bearer {st.session_state.token}"}

    return {}

# -------------------- Auth Panels --------------------

if not st.session_state.token:

    tabs = st.tabs(["Sign in", "Create account", "Forgot password"])
```

```
# ---------- Sign in ----------
with tabs[0]:
    st.subheader("Sign in")
    email = st.text_input("Email", key="login_email")
    password = st.text_input("Password", type="password", key="login_password")
    col1, col2 = st.columns([1,1])
    with col1:
        if st.button("Sign in"):
            if not (email and password):
                st.error("Please enter email and password.")
            else:
                try:
                    r = httpx.post(f"{API}/auth/login",
                                   json={"email": email, "password": password},
                                   timeout=10)
                    if r.status_code == 200:
                        st.session_state.token = r.json()["access_token"]
                        st.rerun()
                    else:
                        st.error(r.json().get("detail", "Login failed."))
                except Exception as e:
                    st.error(f"Error: {e}")
    with col2:
        st.caption("")

# ---------- Create account ----------
with tabs[1]:
```

```
    st.subheader("Create account")
    email = st.text_input("Email", key="signup_email")
    full_name = st.text_input("Full name (optional)")
    password = st.text_input("Password (min 6 chars)", type="password", key="signup_password")
    if st.button("Create account"):
        try:
            r = httpx.post(f"{API}/auth/signup",
                           json={"email": email, "password": password, "full_name": full_name or None},
                           timeout=10)
            if r.status_code == 201:
                st.success("Account created. Please sign in.")
            else:
                st.error(r.json().get("detail", "Could not create account."))
        except Exception as e:
            st.error(f"Error: {e}")

# ---------- Forgot password ----------
with tabs[2]:
    st.subheader("Forgot password")
    reset_email = st.text_input("Enter your registered email", key="reset_email")
    if st.button("Request reset token"):
        if not reset_email:
            st.error("Please enter your email.")
        else:
            try:
                r = httpx.post(f"{API}/auth/request-password-reset",
                               json={"email": reset_email}, timeout=10)
```

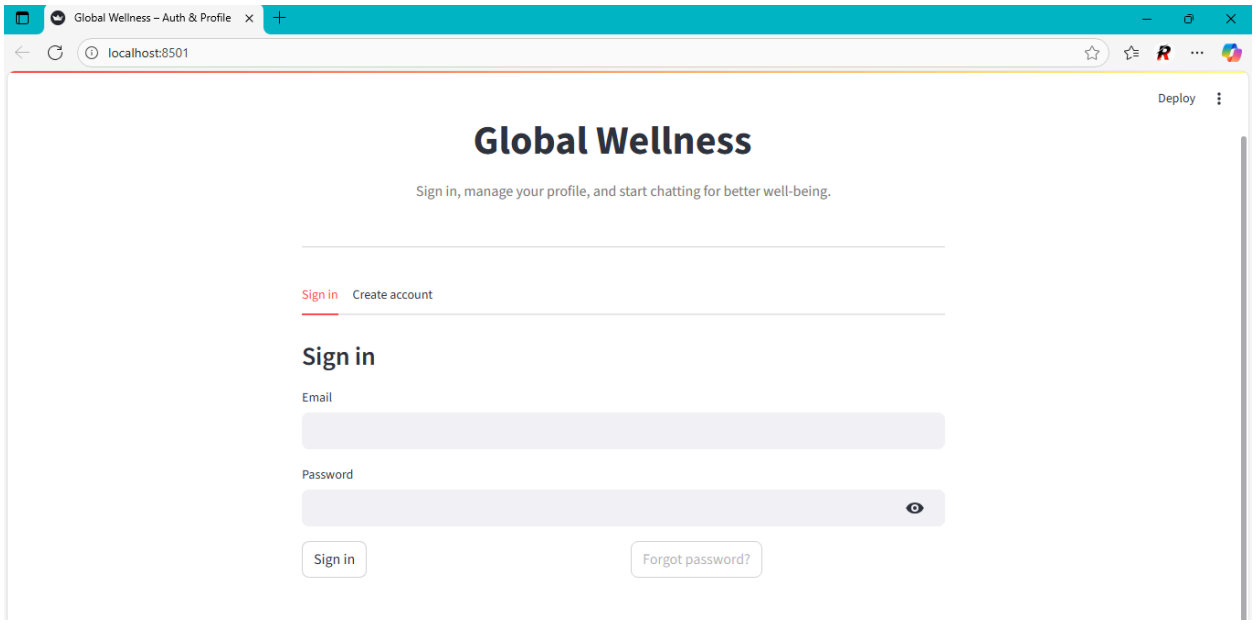
```
            if r.status_code == 200:
                token = r.json().get("reset_token")
                st.session_state.reset_token = token
                st.session_state.show_reset = True
                st.success("Reset token generated (valid 10 minutes). Use it below to set a new password.")
            else:
                st.error(r.json().get("detail", "Error requesting reset."))
        except Exception as e:
            st.error(f"Error: {e}")
if st.session_state.show_reset:
    st.markdown("#### Set a new password")
    st.code(st.session_state.reset_token or "No token", language="text")
    new_pass = st.text_input("New password (min 6 chars)", type="password")
    if st.button("Reset password"):
        if not st.session_state.reset_token:
            st.error("No reset token available. Request it above.")
        elif not new_pass or len(new_pass) < 6:
            st.error("Password must be at least 6 characters.")
        else:
            try:
                r = httpx.post(f"{API}/auth/reset-password",
                               json={"token": st.session_state.reset_token, "new_password": new_pass},
                               timeout=10)
                if r.status_code == 200:
                    st.success("Password reset successful. Please sign in.")
                    st.session_state.show_reset = False
                    st.session_state.reset_token = None
```

```
                else:
                    st.error(r.json().get("detail", "Reset failed."))
            except Exception as e:
                st.error(f"Error: {e}")
else:
    # -------------------- Profile Screen --------------------
    st.subheader("Profile Management")
    # Fetch current profile
    me = None
    try:
        r = httpx.get(f"{API}/auth/me", headers=api_headers(), timeout=10)
        if r.status_code == 200:
            me = r.json()
        else:
            st.error("Session expired. Please sign in again.")
            st.session_state.token = None
            st.rerun()
    except Exception as e:
        st.error(f"Error: {e}")
    if me:
        st.caption(f"Signed in as: **{me.get('email')}**")
        with st.form("profile_form"):
            age_group_options = ["", "Under 18", "18–24", "25–34", "35–44", "45–54", "55+"]
            lang_options = ["", "English", "Hindi", "Spanish", "French", "German", "Chinese"]
            age_group = st.selectbox(
                "Age Group",
                age_group_options,
```

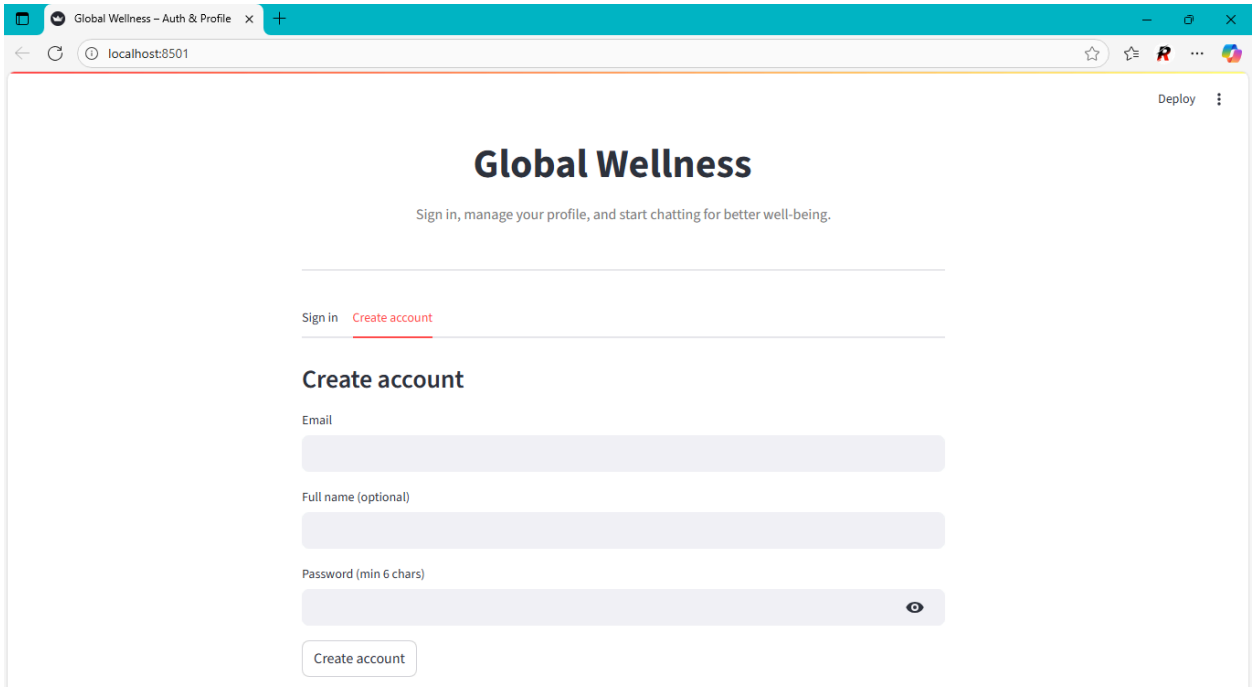
```
            index=age_group_options.index(me.get("age_group")) if me.get("age_group") in age_group_options else 0,
        )
        language = st.selectbox(
            "Language Preference",
            lang_options,
            index=lang_options.index(me.get("language")) if me.get("language") in lang_options else 0,
        )
        wellness_goals = st.text_area("Wellness Goals (optional)", value=me.get("wellness_goals") or "", height=120)
        submitted = st.form_submit_button("Save Profile")
        if submitted:
            try:
                r = httpx.put(
                    f"{API}/profile",
                    json={"age_group": age_group or None, "language": language or None, "wellness_goals":
wellness_goals or None},
                    headers=api_headers(),
                    timeout=10,
                )
                if r.status_code == 200:
                    st.success("Profile saved.")
                else:
                    st.error(r.json().get("detail", "Save failed."))
            except Exception as e:
                st.error(f"Error: {e}")
    st.divider()
    if st.button("Sign out"):
        st.session_state.token = None
        st.rerun()
```

# Output Screenshorts

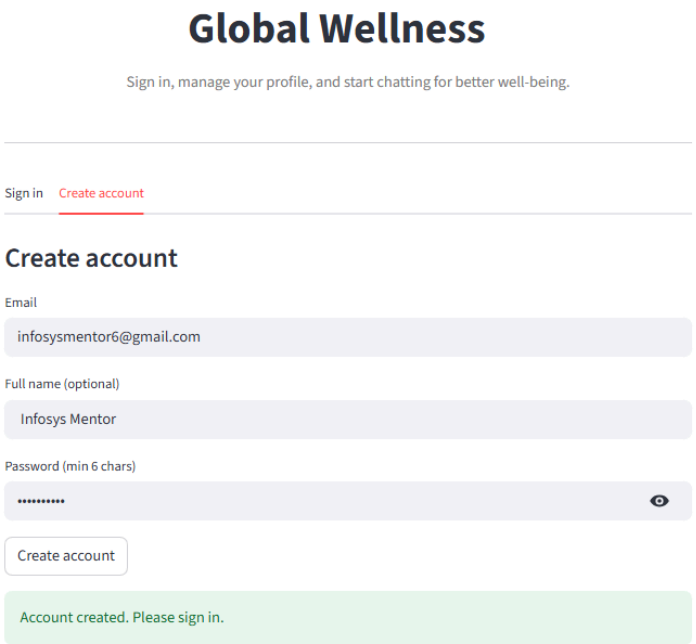
## *UI Page (Sign in)*



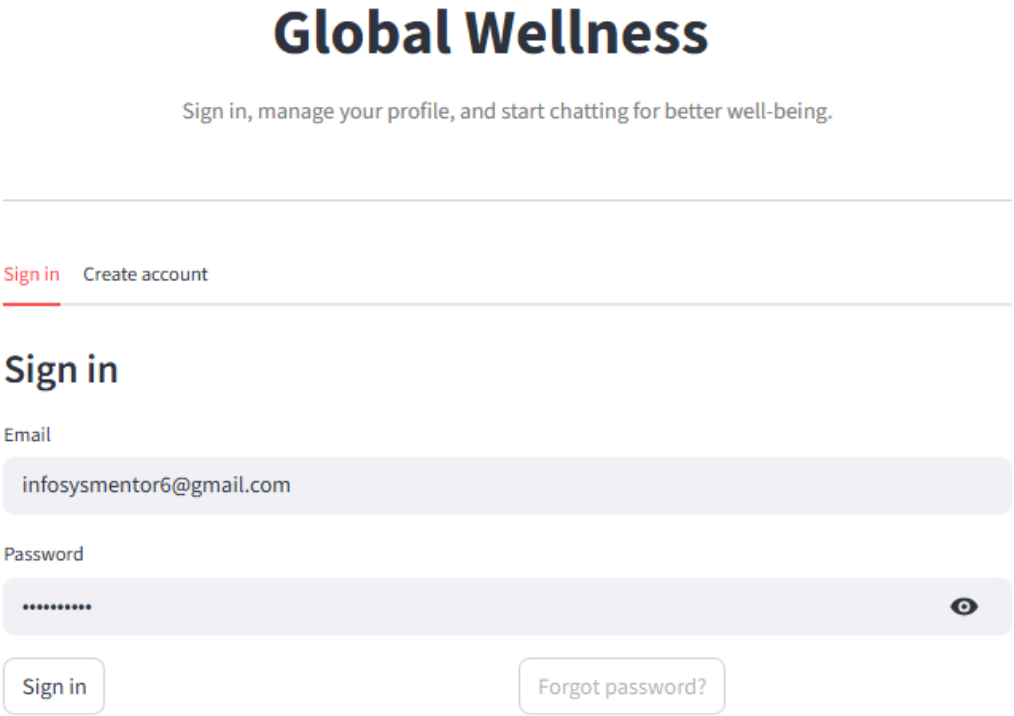

## *UI Page (Create Account)*



## *Creating an account*




## *Sign in*



## *Profile Management*

# Global Wellness

Sign in, manage your profile, and start chatting for better well-being.

---

## Profile Management

Signed in as: infosysmentor6@gmail.com

Age Group

25–34

Language Preference

English

Wellness Goals (optional)

The aim is to achieve a balanced state of mind, body and spirit beyond just the absence of illness, leading to greater overall health and happiness.

Save Profile

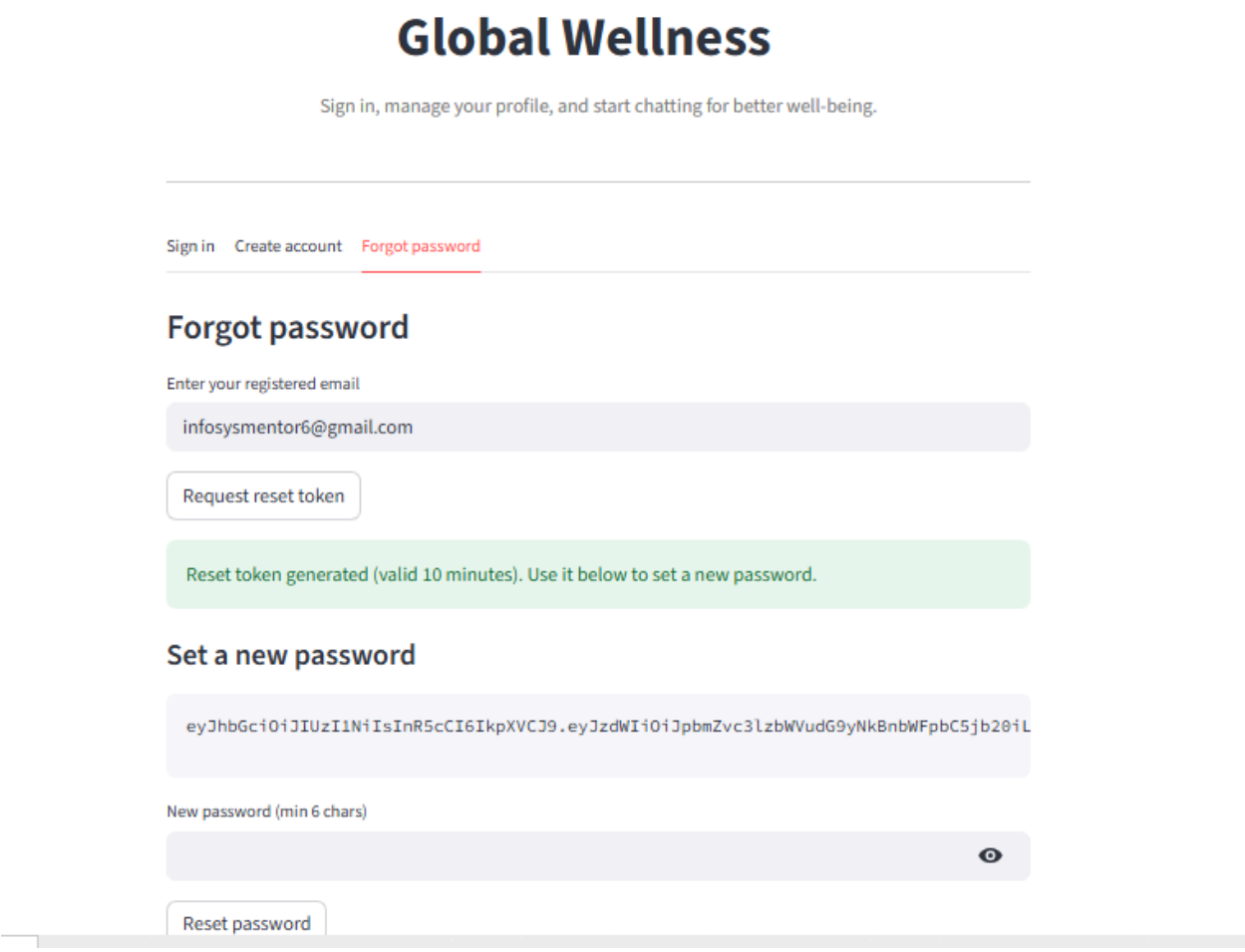
Profile saved.

---

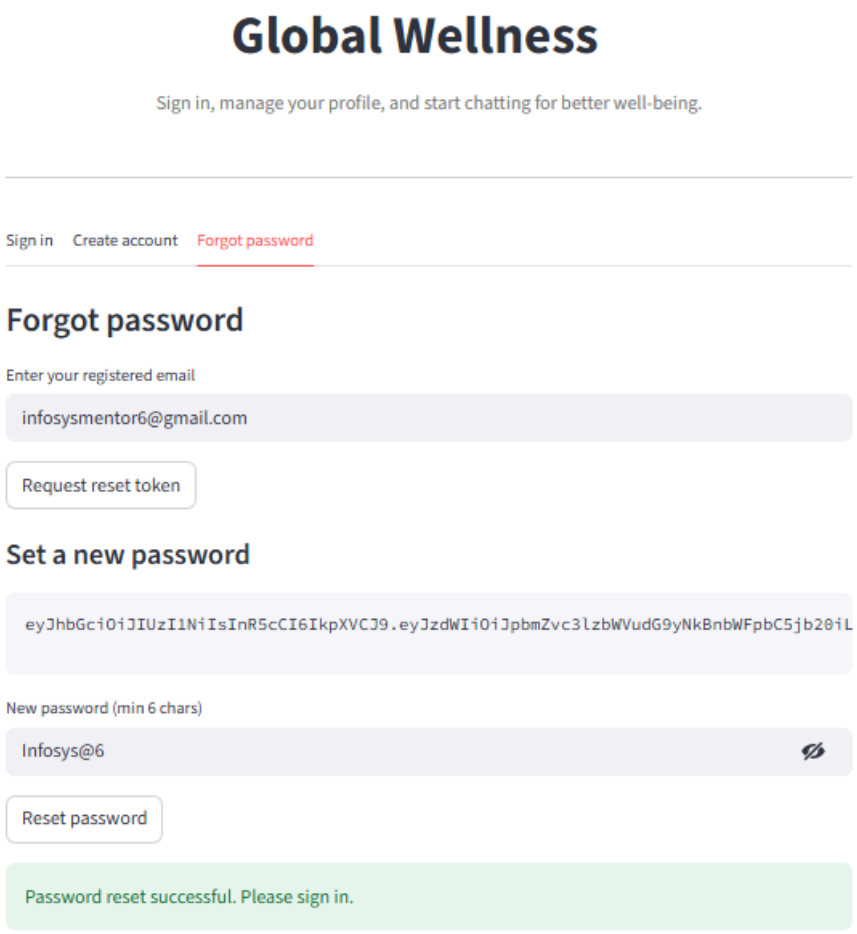
Sign out

## *Forgot password*

Reset token will generate and we will get 10 min to reset our password



The password is get reset and we our ready to Sign in again….



# Database (Sqlite GUI)

## Structure of database

DB Browser for SQLite - C:\Users\Lenovo\Desktop\AI_Global_Chatbot\app\app.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Undo | Open Project | Save Project | Attach Database | Close Database

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Create Table | Create Index | Modify Table | Delete Table | Print | Refresh

| Name | Type | Schema |
|---|---|---|
| Tables (3) | | |
| profiles | | CREATE TABLE profiles ( id INTEGER PRIMARY KEY AUTOINCREMENT, user_id INTEGER UNIQUE NOT NULL, age_ |
| id | INTEGER | "id" INTEGER |
| user_id | INTEGER | "user_id" INTEGER NOT NULL UNIQUE |
| age_group | TEXT | "age_group" TEXT |
| language | TEXT | "language" TEXT |
| wellness_goals | TEXT | "wellness_goals" TEXT |
| updated_at | TEXT | "updated_at" TEXT DEFAULT CURRENT_TIMESTAMP |
| sqlite_sequence | | CREATE TABLE sqlite_sequence(name,seq) |
| name | | "name" |
| seq | | "seq" |
| users | | CREATE TABLE users ( id INTEGER PRIMARY KEY AUTOINCREMENT, email TEXT UNIQUE NOT NULL, password_h |
| id | INTEGER | "id" INTEGER |
| email | TEXT | "email" TEXT NOT NULL UNIQUE |
| password_hash | TEXT | "password_hash" TEXT NOT NULL |
| full_name | TEXT | "full_name" TEXT |
| created_at | TEXT | "created_at" TEXT DEFAULT CURRENT_TIMESTAMP |
| Indices (0) | | |
| Views (0) | | |
| Triggers (0) | | |

## Users

Table: users

| | id | email | password_hash | full_name | created_at |
|---|---|---|---|---|---|
| 1 | 1 | test1@gmail.com | $2b$12$0UXXcWkedg.... | rp | 2025-08-23 13:49:12 |
| 2 | 2 | himanshuthakur@gmail.com | $2b$12$etivSCvVw5tIIxSHhgAzlO0FBKNm... | Himanshu Thakur | 2025-08-23 17:21:47 |
| 3 | 3 | raghvendra.partap@gmail.com | $2b$12$ZaWT.EODp9A1S6bcm3su9eUDD1vr... | Raghvendra Pratap Singh | 2025-08-25 12:51:24 |
| 4 | 4 | mentor6@gmail.com | $2b$12$jXOGM1IwMnscqlMH6PlfSOZh694i... | Mentor | 2025-08-25 13:44:24 |
| 5 | 5 | infosysmentor6@gmail.com | $2b$12$J66bJuy7Ua3lnNfdH9/... | Infosys Mentor | 2025-08-26 19:26:01 |

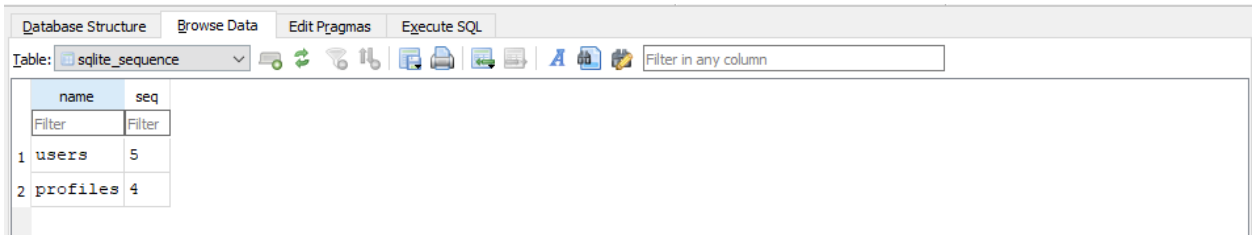
## Profiles

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: profiles

| | id | user_id | age_group | language | wellness_goals | updated_at |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 18-24 | English | To get healthy and fit in life | 2025-08-23 17:50:22 |
| 2 | 2 | 3 | 18-24 | English | i want to eat healthy and stay ... | 2025-08-25 12:53:35 |
| 3 | 3 | 4 | 18-24 | English | i want to get fit | 2025-08-25 13:45:29 |
| 4 | 4 | 5 | 25-34 | English | The aim is to achieve a balanced ... | 2025-08-26 19:33:18 |

## *Sqlite_Sequence*




## *Conclusion*

In this project, we successfully developed a User Authentication and Profile Management system using FastAPI, JWT authentication, and SQLite as the backend database. The implementation focused on providing secure account management through password hashing, token-based login sessions, and persistent storage of user information.

The system demonstrates the following key achievements:

- Creation of a signup and login API with secure password hashing (bcrypt).
- Implementation of JWT tokens for stateless authentication, ensuring session security.
- Integration of SQLite database for storing and managing user credentials.
- Establishment of a virtual environment and installation of required dependencies, ensuring modular and manageable development.

.