Infosys Springboard Virtual Internship

Project Name: "Developing chatbot on global wellness"

Milestone 1:- User authentication and Profile Management

Name: Injeti Venkata Nikhil

Date:09-09-2025

Milestone 1:- User authentication and Profile Management

Introduction:-

The Global Wellness Chatbot is a web-based application designed to promote personal wellness and healthy living by providing users with an intuitive platform to manage their wellness information. Built using Streamlit for the frontend, it integrates seamlessly with a backend API to handle user authentication, profile management, and secure data storage. The application focuses on delivering a clean, modern, and user-friendly interface that encourages users to interact with the platform effortlessly.

The application offers core features such as **user registration, login, password recovery**, and **profile management**. Users can update their personal information including age, location, preferred language, and contact details.

The primary goal of the Global Wellness Chatbot is to **simplify wellness management** and provide users with an accessible platform to keep their personal health data organized. By integrating a chatbot interface with modern UI elements, it enhances user engagement while maintaining security and reliability.

Objectives:

The main objectives successfully achieved in this project phase are:

1. Virtual Environment Setup:-

- Set up the project structure combining **FastAPI backend** and **Streamlit frontend**.
- Configured a Python virtual environment to isolate dependencies and avoid conflicts.

2. User Authentication System:-

- Designed APIs to support **user registration and login** workflows.
- Implemented secure storage of credentials with hashing for enhanced safety.

- Enabled account recovery through a **Forgot Password feature** with email support.

3.<u>User Profile Features:-</u>

- Developed functionality for viewing and editing personal profile details.
- Allowed customization of fields such as **age, phone, location, and preferred language**.
- Added frontend components for interactive updates and displaying profile information.

4. <u>Database Integration:-</u>

- Integrated **SQLite** as a lightweight relational database for storing user and profile data.
- Automated schema creation and ensured smooth handling of CRUD operations.

5. <u>Security & Authorization:-</u>

- Applied **JWT authentication** to safeguard sensitive endpoints.
- Ensured authenticated sessions persist securely within the frontend.

6. <u>Seamless UI–API Connectivity:-</u>

- Established communication between the Streamlit interface and the FastAPI backend.
- Tested smooth workflows such as **sign up → log in → profile management** through API calls.

Tech stack:

- Frontend: Streamlit
- Backend API: FastAPI (assumed)
- Authentication: JWT tokens
- HTTP Requests: httpx
- Styling: Custom CSS in Streamlit

Code Implementation:

The backend was developed using **FastAPI** with SQLite for data storage and JWT for secure authentication.The frontend was built in **Streamlit**, providing an interactive interface for login, signup, and profile management.

1.Database setup(SQLite):

We use SQLite to store user authentication and profile details.

database.py

```
import sqlite3
from typing import Optional
DB_PATH = "users.db"
def get_db():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn
def init_db():
    conn = get_db()
    conn.execute("""
        CREATE TABLE IF NOT EXISTS users(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT,
password TEXT,
```

```
        email TEXT UNIQUE,
        age INTEGER,
        location TEXT,
        phone TEXT,
        language TEXT
    )
    """)
    conn.commit()
    conn.close()
def row_to_dict(row: sqlite3.Row) -> dict:
    return dict(row) if row else None
def get_user_by_email_or_username(identifier: str) -> Optional[sqlite3.Row]:
    conn = get_db()
    row = conn.execute(
        "SELECT * FROM users WHERE email = ? OR username = ?",
        (identifier, identifier)
    ).fetchone()
    conn.close()
    return row
```

2.Aunthentication & API Endpoints(FastAPI):

User authentication was implemented with JWT tokens to ensure secure access. FastAPI endpoints were created for signup, login, password reset, and profile management.

main.py:

```
import os
from datetime import datetime, timedelta
from typing import Optional
from fastapi import FastAPI, Depends, HTTPException, Query
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel
from passlib.context import CryptContext
```

```
from jose import jwt, JWTError
from dotenv import load_dotenv
from pathlib import Path
import smtplib
import sqlite3
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# ---------- Import DB functions ----------
from .database import get_db, init_db, row_to_dict, get_user_by_email_or_username

# ---------- Load .env ----------
load_dotenv(dotenv_path=Path(__file__).parent / ".env")
SMTP_HOST = os.getenv("EMAIL_HOST", "smtp.gmail.com")
SMTP_PORT = int(os.getenv("EMAIL_PORT", 587))
SMTP_USER = os.getenv("EMAIL_USER")
SMTP_PASS = os.getenv("EMAIL_PASS")
SECRET_KEY = os.getenv("SECRET_KEY", "supersecretkey123")

print("SMTP_USER:", SMTP_USER)
print("SMTP_PASS:", SMTP_PASS)

# ---------- Config ----------
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60
RESET_TOKEN_EXPIRE_MINUTES = 15
FRONTEND_URL = "http://localhost:8501"   # Streamlit frontend

app = FastAPI(title="Global Wellness Backend")
```

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="login")

# ---------- Initialize DB ----------
init_db()

# ---------- Schemas ----------
class RegisterUser(BaseModel):
    username: str
    email: str
    password: str
    age: Optional[int] = None
    location: Optional[str] = None
    phone: Optional[str] = None
    language: Optional[str] = None

class UpdateProfile(BaseModel):
    username: Optional[str] = None
    age: Optional[int] = None
    location: Optional[str] = None
    phone: Optional[str] = None
    language: Optional[str] = None

class ResetPasswordRequest(BaseModel):
    token: str
    new_password: str

# ---------- Utils ----------
def get_password_hash(pw: str) -> str:
```

```
    return pwd_context.hash(pw)

def verify_password(plain: str, hashed: str) -> bool:
    return pwd_context.verify(plain, hashed)

def create_access_token(data: dict, minutes: int = ACCESS_TOKEN_EXPIRE_MINUTES) -> str:
    to_encode = data.copy()
    to_encode["exp"] = datetime.utcnow() + timedelta(minutes=minutes)
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)

def get_current_user(token: str = Depends(oauth2_scheme)) -> dict:
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        email = payload.get("sub")
        if not email:
            raise HTTPException(status_code=401, detail="Invalid token")
    except JWTError:
        raise HTTPException(status_code=401, detail="Invalid token")

    conn = get_db()
    row = conn.execute("SELECT * FROM users WHERE email = ?", (email,)).fetchone()
    conn.close()
    if not row:
        raise HTTPException(status_code=401, detail="User not found")
    return row_to_dict(row)

def send_reset_email(to_email: str, token: str):
    reset_link = f"{FRONTEND_URL}/reset_password?token={token}"
    subject = "Password Reset Request"
```

```
body = f"""
Hi,

We received a request to reset your password.
Click the link below to reset it:

{reset_link}

If you didn’t request this, you can ignore this email.

Best,
Global Wellness Team
"""
try:
    msg = MIMEMultipart()
    msg["From"] = SMTP_USER
    msg["To"] = to_email
    msg["Subject"] = subject
    msg.attach(MIMEText(body, "plain"))

    with smtplib.SMTP(SMTP_HOST, SMTP_PORT) as server:
        server.starttls()
        server.login(SMTP_USER, SMTP_PASS)
        server.sendmail(SMTP_USER, to_email, msg.as_string())

    print(f"✅ Reset email sent to {to_email}")
    print(f"🔗 Reset link (debug): {reset_link}")
except Exception as e:
    print(f"❌ Failed to send email: {e}")
```

```
# ---------- Endpoints ----------
@app.post("/register")
def register(user: RegisterUser):
    conn = get_db()
    try:
        conn.execute(
            """INSERT INTO users (username, password, email, age, location, phone, language)
               VALUES (?, ?, ?, ?, ?, ?, ?)""",
            (
                user.username,
                get_password_hash(user.password),
                user.email,
                user.age,
                user.location,
                user.phone,
                user.language
            )
        )
        conn.commit()
        return {"message": "User registered successfully"}
    except sqlite3.IntegrityError:
        raise HTTPException(status_code=400, detail="Email already registered")
    finally:
        conn.close()

@app.post("/login")
def login(form: OAuth2PasswordRequestForm = Depends()):
    user = get_user_by_email_or_username(form.username)
```

```
    if not user:
        raise HTTPException(status_code=401, detail="Invalid username/email or password")

    if not verify_password(form.password, user["password"]):
        raise HTTPException(status_code=401, detail="Invalid username/email or password")

    token = create_access_token({"sub": user["email"]})
    return {"access_token": token, "token_type": "bearer"}

@app.get("/profile")
def get_profile(current_user: dict = Depends(get_current_user)):
    return {
        "username": current_user["username"],
        "email": current_user["email"],
        "age": current_user.get("age"),
        "location": current_user.get("location"),
        "phone": current_user.get("phone"),
        "language": current_user.get("language"),
    }

@app.put("/profile")
def update_profile(payload: UpdateProfile, current_user: dict = Depends(get_current_user)):
    fields, values = [], []
    for name in ["username", "age", "location", "phone", "language"]:
        val = getattr(payload, name)
        if val is not None:
            fields.append(f"{name} = ?")
            values.append(val)
```

```
    if not fields:
        return {"message": "Nothing to update"}

    values.append(current_user["email"])
    conn = get_db()
    conn.execute(f"UPDATE users SET {', '.join(fields)} WHERE email = ?", values)
    conn.commit()
    conn.close()
    return {"message": "Profile updated successfully"}

@app.post("/forgot-password")
def forgot_password(email: str = Query(...)):
    user = get_user_by_email_or_username(email)
    if not user:
        raise HTTPException(status_code=404, detail="Email not found")

    reset_token = create_access_token({"sub": user["email"]}, minutes=RESET_TOKEN_EXPIRE_MINUTES)
    send_reset_email(user["email"], reset_token)
    return {"message": "Password reset email sent"}

@app.post("/reset-password")
def reset_password(payload: ResetPasswordRequest):
    try:
        data = jwt.decode(payload.token, SECRET_KEY, algorithms=[ALGORITHM])
        email = data.get("sub")
        if not email:
            raise HTTPException(status_code=400, detail="Invalid token")
    except JWTError:
        raise HTTPException(status_code=400, detail="Invalid or expired token")
```

```
    hashed_pw = get_password_hash(payload.new_password)
    conn = get_db()
    conn.execute("UPDATE users SET password = ? WHERE email = ?", (hashed_pw, email))
    conn.commit()
    conn.close()
    return {"message": "Password reset successful"}
```

3.Frontend(streamlit):

The frontend was built with Streamlit, offering a clean and user-friendly interface. It enables users to register, log in, update profiles, and interact with the backend APIs seamlessly.

app.py

```
import streamlit as st
import httpx
import base64
API_URL = "http://127.0.0.1:8000"
# Session state
if "token" not in st.session_state:
    st.session_state.token = None
if "email" not in st.session_state:
    st.session_state.email = None


# ---- API Helpers ----
def api_register(username, email, password, age, location, phone, language):
    return httpx.post(f"{API_URL}/register", json={
        "username": username,
        "email": email,
        "password": password,
```

```
        "age": age if age else None,
        "location": location or None,
        "phone": phone or None,
        "language": language or None
    })

def api_login(email_or_username, password):
    r = httpx.post(f"{API_URL}/login", data={
        "username": email_or_username,
        "password": password
    })
    if r.status_code == 200:
        data = r.json()
        st.session_state.token = data["access_token"]
        st.session_state.email = email_or_username
    return r

def api_get_profile():
    headers = {"Authorization": f"Bearer {st.session_state.token}"}
    return httpx.get(f"{API_URL}/profile", headers=headers)

def api_update_profile(updates: dict):
    headers = {"Authorization": f"Bearer {st.session_state.token}"}
    return httpx.put(f"{API_URL}/profile", json=updates, headers=headers)

def logout():
    st.session_state.token = None
    st.session_state.email = None
    st.success("✅ Logged out successfully")

# ---- UI ----
```

```
st.set_page_config(page_title="Global Wellness Chatbot", page_icon="🌍", layout="centered")
st.title("🌍 Global Wellness Chatbot")
st.markdown("### 🧘 Stay healthy. Stay connected.")

st.markdown("""
<style>
.stButton button {
  background: linear-gradient(to right, #6a11cb, #2575fc);
  color: white !important;
  border-radius: 8px; padding: 0.6rem 1.2rem; border: none;
  font-weight: bold; width: 100%; margin-bottom: 1rem;
}
.profile-card { background: white; padding: 2rem; border-radius: 15px;
  box-shadow: 0 6px 20px rgba(0,0,0,0.1); text-align: center; width: 60%; margin: auto; }
.avatar { width: 100px; height: 100px; border-radius: 50%; margin-bottom: 1rem; border: 3px solid #2575fc; }
</style>
""", unsafe_allow_html=True)

# ---- Login / Signup / Forgot ----
if not st.session_state.token:
    tab_login, tab_signup, tab_forgot = st.tabs(["🔐 Login", "✨ Signup", "❓ Forgot Password"])

    # ----- LOGIN -----
    with tab_login:
        st.subheader("Login to your account")
        login_id = st.text_input("📧 Email", key="login_id")
        password = st.text_input("🔑 Password", type="password", key="login_password")
        if st.button("✅ Login"):
            if login_id and password:
                res = api_login(login_id, password)
```

```
            if res.status_code == 200:
                st.success("🎉 Login successful!")
            else:
                try:
                    st.error(f"❌ {res.json().get('detail', 'Invalid credentials')}")
                except Exception:
                    st.error("❌ Invalid credentials")
        else:
            st.error("⚠️ Please enter your credentials.")

    # ----- SIGNUP -----
    with tab_signup:
        st.subheader("Create a new account")
        col1, col2 = st.columns(2)
        with col1:
            username = st.text_input("👤 Username", key="signup_username")
            email    = st.text_input("📧 Email", key="signup_email")
            password = st.text_input("🔑 Password", type="password", key="signup_password")
            age      = st.number_input("🎂 Age (optional)", min_value=0, max_value=120, step=1, value=0)
        with col2:
            location = st.text_input("📍 Location (optional)")
            phone    = st.text_input("📱 Phone (optional)")
            language = st.selectbox("🗣️ Preferred Language", ["", "English", "Hindi", "Telugu", "Tamil", "Kannada", "Malayalam", "Bengali", "Marathi", "Gujarati", "Urdu"])

        if st.button("🚀 Register"):
            if username and email and password:
                res = api_register(username, email, password, age if age > 0 else None, location.strip(), phone.strip(), language if language else None)
                if res.status_code == 200:
                    st.success("🎉 User registered! You can now login.")
```

```
            else:
                try:
                    st.error(res.json().get("detail", "Registration failed"))
                except Exception:
                    st.error("Registration failed")
        else:
            st.error("⚠️ Please fill username, email, and password.")

    # ----- FORGOT PASSWORD -----
    with tab_forgot:
        st.subheader("Forgot Password?")
        forgot_email = st.text_input("📧 Enter your registered email")
        if st.button("📨 Send Reset Link"):
            if forgot_email:
                with st.spinner("Sending reset email..."):
                    try:
                        res = httpx.post(f"{API_URL}/forgot-password", params={"email": forgot_email}, timeout=60)
                        if res.status_code == 200:
                            st.success("✅ Reset link sent! Check your email inbox.")
                        else:
                            st.error(res.json().get("detail", "⚠️ Error sending reset link"))
                    except Exception as e:
                        st.error(f"🔌 Connection error: {e}")
            else:
                st.error("⚠️ Please enter your email")

# ---- Profile (Logged In) ----
else:
    st.markdown("## 👤 User Profile")
```

```
res = api_get_profile()
if res.status_code != 200:
    st.error(" ⚠️ Failed to load profile")
else:
    profile = res.json()

    st.markdown("""
    <style>
    .profile-card {
        background: none;
        padding: 2rem;
        border-radius: 15px;
        box-shadow: 0 6px 20px rgba(0,0,0,0.05);
        text-align: center;
        width: 60%;
        margin: auto;
    }
    .avatar {
        width: 100px;
        height: 100px;
        border-radius: 50%;
        margin-bottom: 1rem;
        border: 3px solid #2575fc;
        object-fit: cover;
    }
    .buttons-row { display: flex; justify-content: center; gap: 1rem; margin-top: 1rem; }
    </style>
    """, unsafe_allow_html=True)

    st.markdown("<div class='profile-card'>", unsafe_allow_html=True)
```

```
    # Display avatar (no upload)
    avatar_url = profile.get("avatar_url") or "https://cdn-icons-png.flaticon.com/512/3135/3135715.png"
    st.markdown(f"<img class='avatar' src='{avatar_url}'>", unsafe_allow_html=True)
    st.markdown(f"### {profile.get('username') or ''} 👋")
    st.markdown(f"📧 **Email:** {profile.get('email') or ''}")

    col1, col2 = st.columns(2)
    with col1:
        new_username = st.text_input("✏️ Update Username", profile.get("username") or "")
        new_location = st.text_input("📍 Update Location", profile.get("location") or "")
        new_language = st.selectbox(
            "🗣️ Preferred Language",
            ["", "English", "Hindi", "Telugu", "Tamil", "Kannada", "Malayalam", "Bengali", "Marathi", "Gujarati", "Urdu"],
            index=(["", "English", "Hindi", "Telugu", "Tamil", "Kannada", "Malayalam", "Bengali", "Marathi", "Gujarati", "Urdu"].index(profile.get("language")) if profile.get("language") else 0)
        )
    with col2:
        new_age = st.number_input("🎂 Update Age", min_value=0, max_value=120, value=int(profile.get("age") or 0), step=1)
        new_phone = st.text_input("📱 Update Phone", profile.get("phone") or "")

    # Buttons in one row
    st.markdown("<div class='buttons-row'>", unsafe_allow_html=True)
    save_clicked = st.button("💾 Save Changes")
    logout_clicked = st.button("🚪 Logout")
    st.markdown("</div>", unsafe_allow_html=True)

    if save_clicked:
        updates = {}
        if new_username != (profile.get("username") or ""): updates["username"] = new_username.strip() or None
```

```
        if new_location != (profile.get("location") or ""): updates["location"] = new_location.strip() or None
        if new_language != (profile.get("language") or ""): updates["language"] = new_language or None
        if new_age != int(profile.get("age") or 0): updates["age"] = int(new_age)
        if new_phone != (profile.get("phone") or ""): updates["phone"] = new_phone.strip() or None

        if not updates:
            st.info("No changes to save.")
        else:
            ures = api_update_profile(updates)
            if ures.status_code == 200:
                st.success("✅ Profile updated successfully! Refreshing...")
                st.experimental_rerun()
            else:
                st.error("⚠️ Error updating profile")

    if logout_clicked:
        logout()

    st.markdown("</div>", unsafe_allow_html=True)
```

---

reset_password.py:

```
import streamlit as st
import httpx
# Backend API (FastAPI in main.py)
API_URL = "http://127.0.0.1:8000"
st.set_page_config(page_title="Reset Password", page_icon="🔑", layout="centered")
st.title("🔑 Reset Your Password")
# ✅ Fetch token from reset link
```

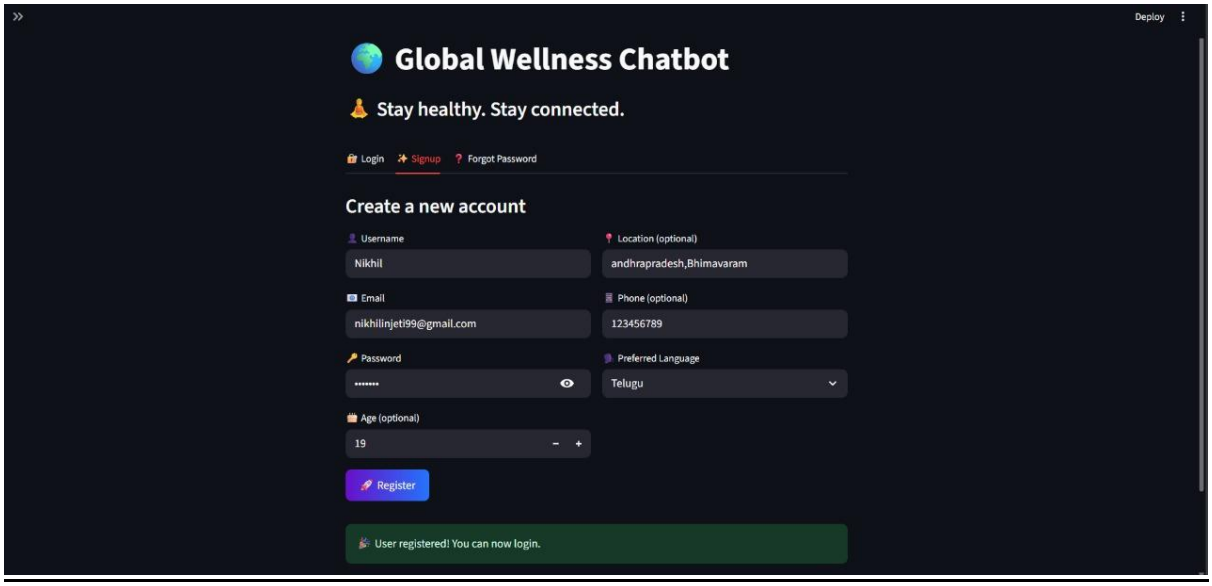
```
params = st.query_params
token = params.get("token", None)
if not token:
    st.error("❌ Invalid or missing reset link")
else:
    st.info("Please enter your new password below 👇")

    new_pw = st.text_input("🆕 New Password", type="password")
    confirm_pw = st.text_input("✅ Confirm Password", type="password")

    if st.button("💾 Save Password"):
        if not new_pw or not confirm_pw:
            st.warning(" ⚠️ Both fields are required")
        elif new_pw != confirm_pw:
            st.error(" ⚠️ Passwords do not match")
        else:
            try:
                res = httpx.post(
                    f"{API_URL}/reset-password",
                    json={"token": token, "new_password": new_pw},
                    timeout=10.0,
                )
                if res.status_code == 200:
                    st.success("✅ Password reset successful! You can now login.")
                else:
                    error_msg = res.json().get("detail", "Something went wrong")
                    st.error(f"❌ Error: {error_msg}")
            except Exception as e:
                st.error(f"🛑 Connection error: {e}")
```

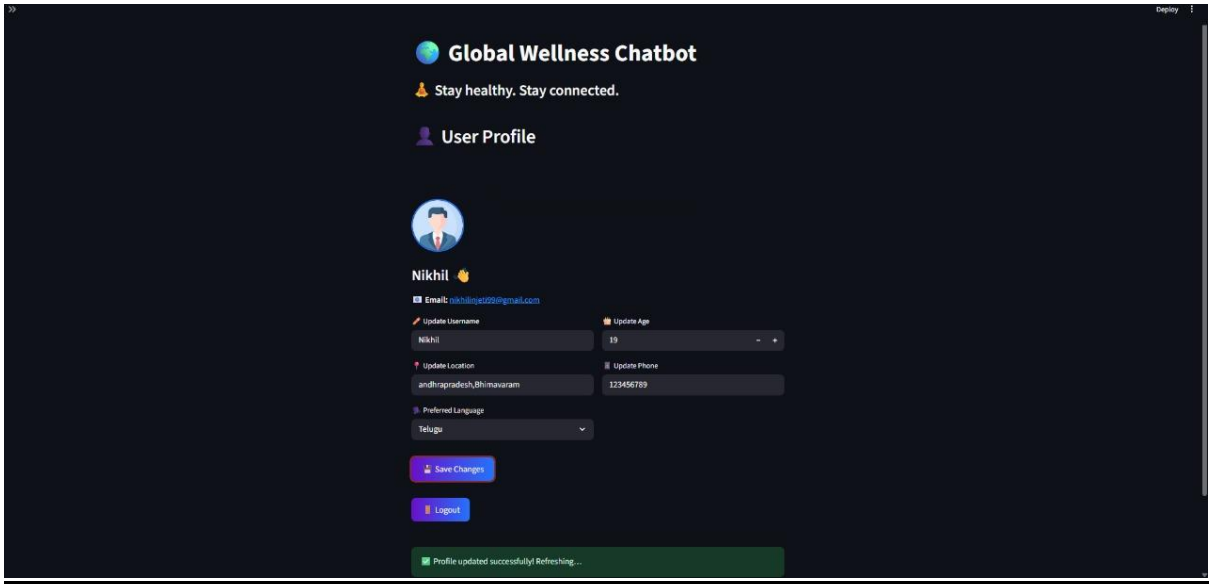
Output Screenshots:

UI page(sign up)

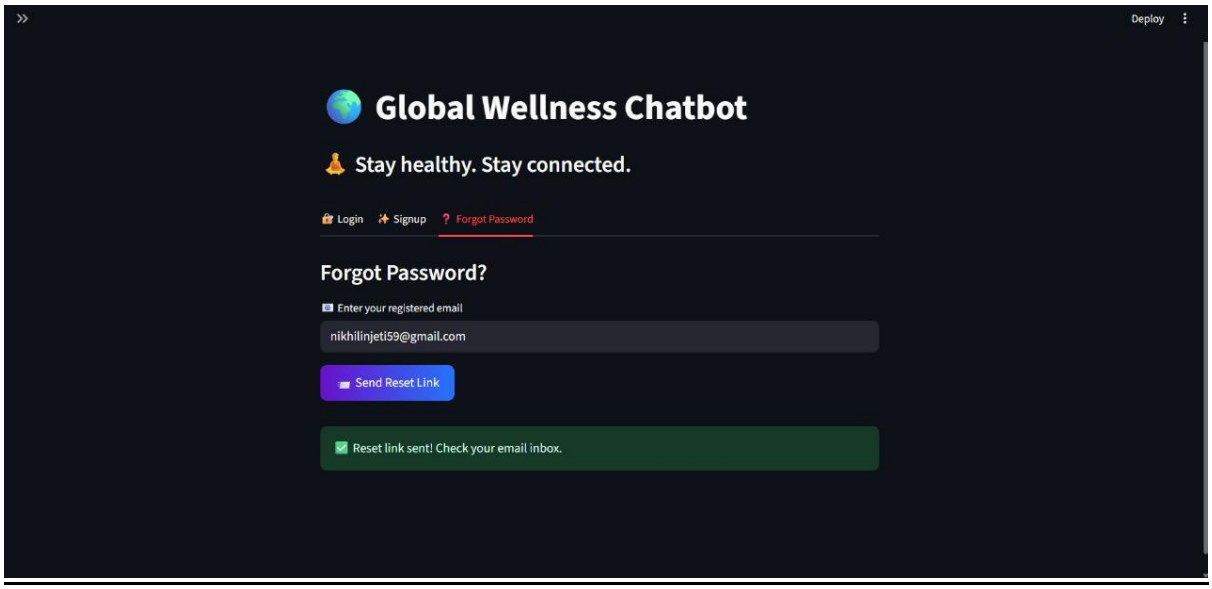


UI page(Sign in)

Profile Management

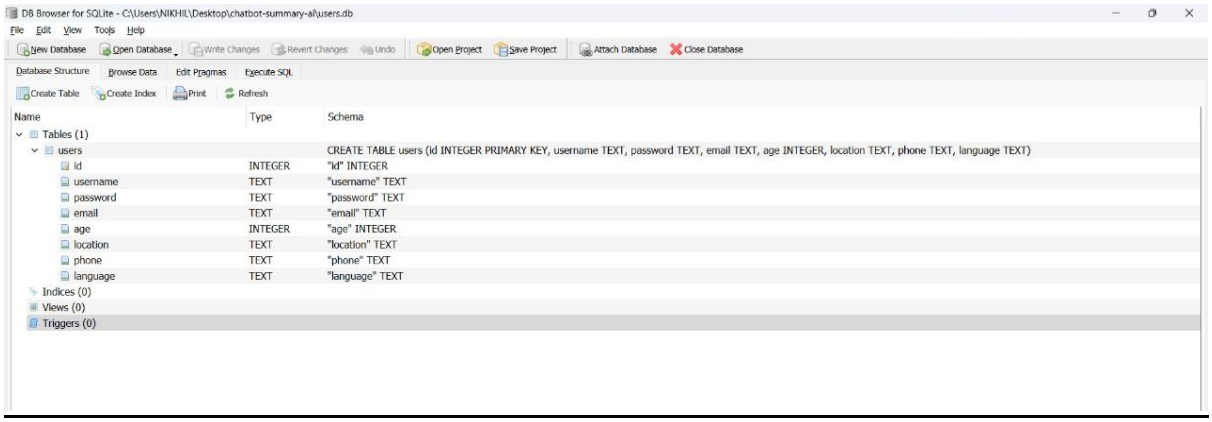


Forgot Password

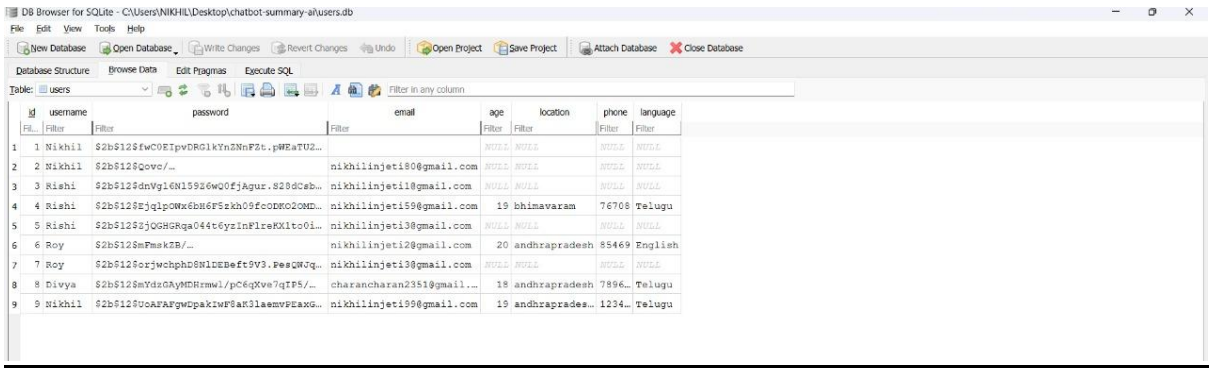


Database(Sqlite GUI)

Structure of Database



Users Data



Conclusion:

In this project we successfully integrated FastAPI as the backend and Streamlit as the frontend to deliver a secure, efficient, and user-friendly wellness management system. The implementation of JWT-based authentication, SQLite database integration, and profile management ensured both data security and smooth functionality. Overall, the project demonstrates a practical approach to building a full-stack application with clear scalability for future enhancements.