Project Documentation

Medical Drug Recommender

Overview

This project is a web application built using Streamlit, designed to manage user registration, login, and provide personalized drug recommendations based on user input. The application connects to a MySQL database for storing user information and uses machine learning techniques for drug recommendations.

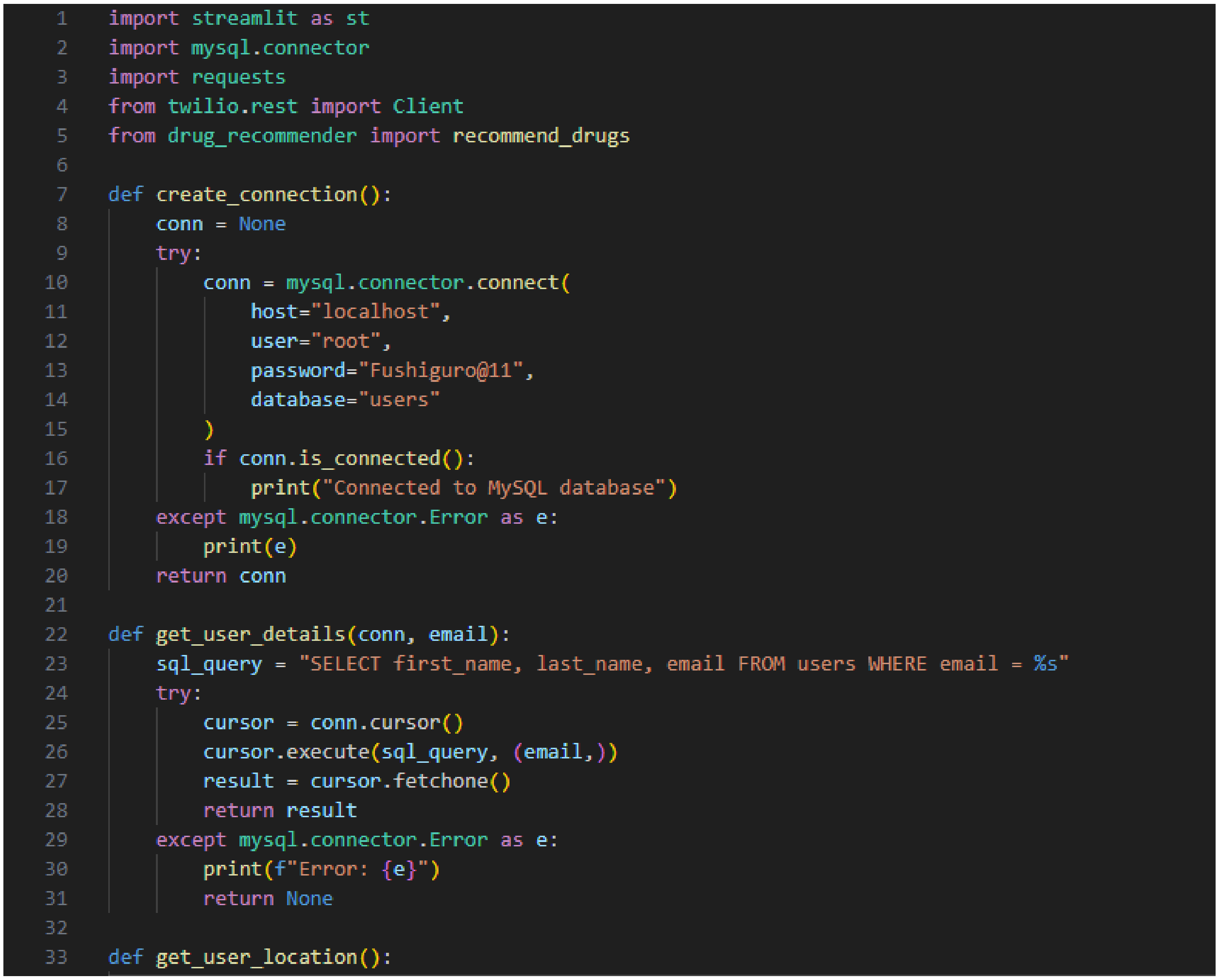
Table of Contents

User_page.py

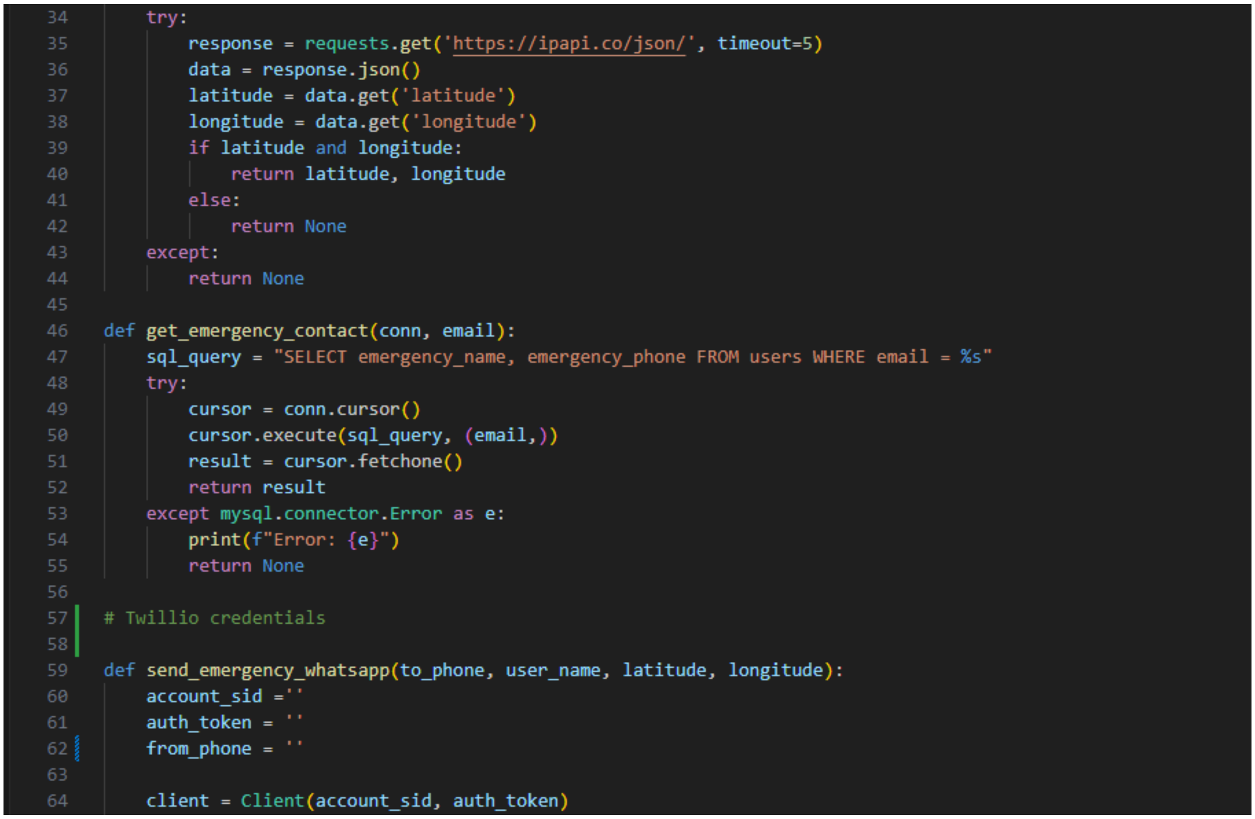
- Purpose: Creates a user interface for displaying user details, fetching location, sending emergency contact messages, and recommending drugs.

- Key Functions:

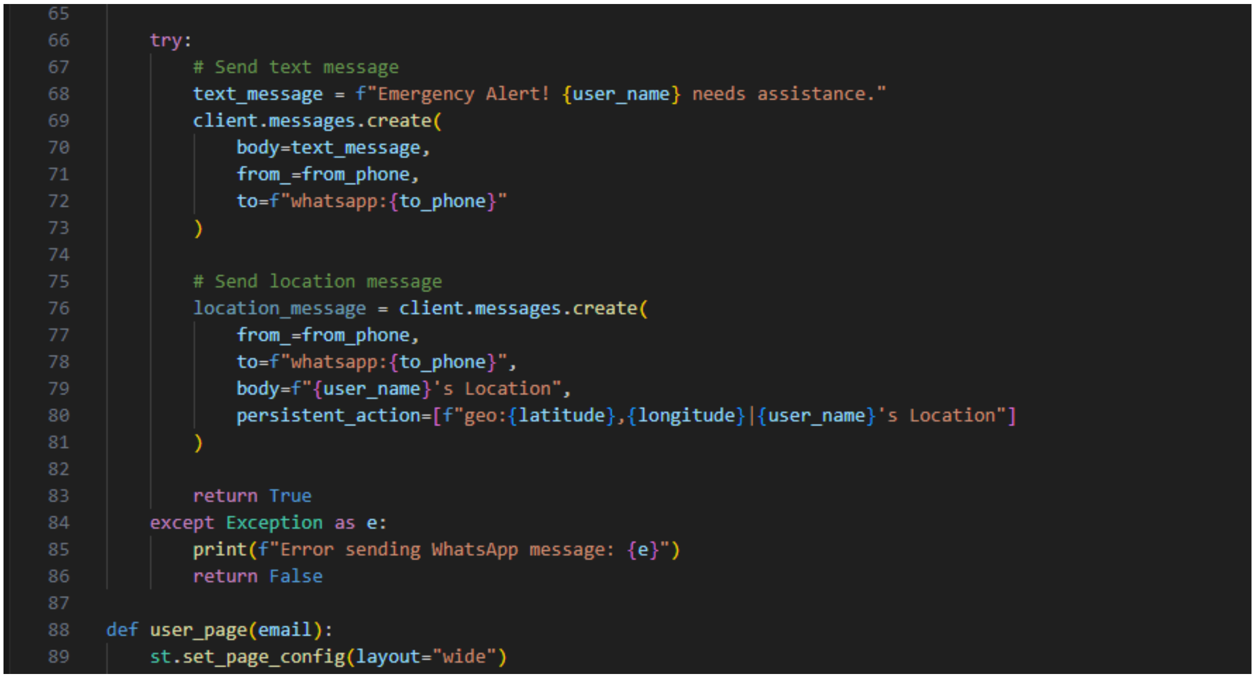
  - create_connection(): Establishes a connection to the MySQL database.
  - get_user_details(user_id): Retrieves user details from the database.
  - fetch_location(user_id): Fetches the user's location.
  - send_emergency_message(contact, message): Sends an emergency message via WhatsApp using Twilio.
  - recommend_drugs(description): Recommends drugs based on the user's input description.



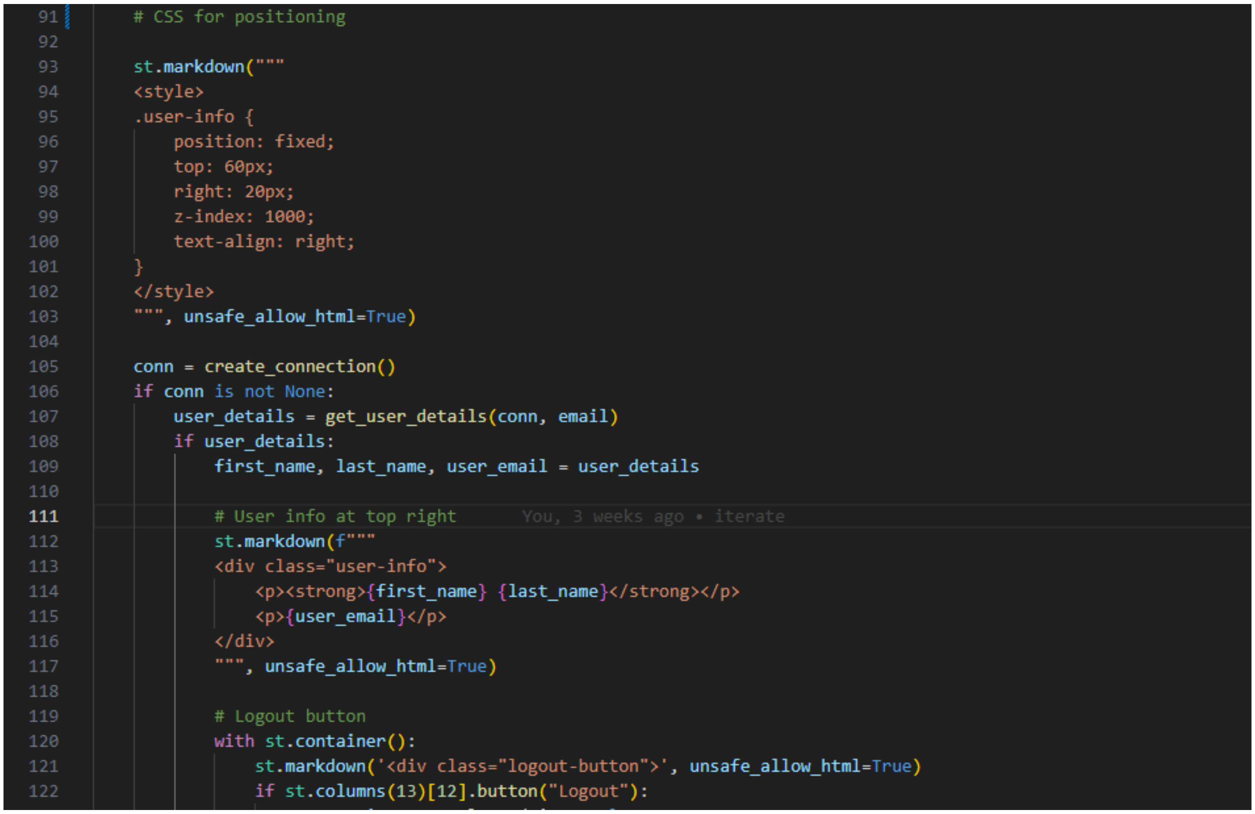
```python
import streamlit as st
import mysql.connector
import requests
from twilio.rest import Client
from drug_recommender import recommend_drugs

def create_connection():
    conn = None
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Fushiguro@11",
            database="users"
        )
        if conn.is_connected():
            print("Connected to MySQL database")
    except mysql.connector.Error as e:
        print(e)
    return conn

def get_user_details(conn, email):
    sql_query = "SELECT first_name, last_name, email FROM users WHERE email = %s"
    try:
        cursor = conn.cursor()
        cursor.execute(sql_query, (email,))
        result = cursor.fetchone()
        return result
    except mysql.connector.Error as e:
        print(f"Error: {e}")
        return None

def get_user_location():
```



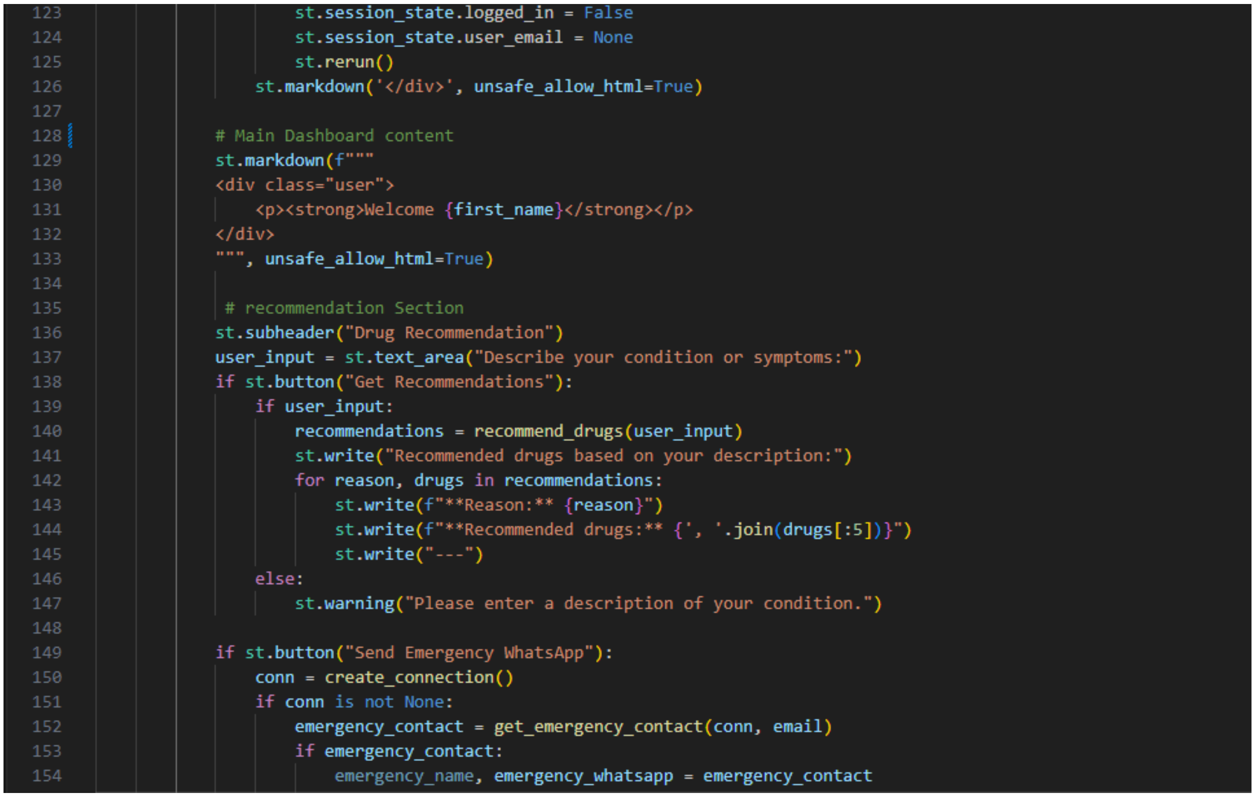
```python
    try:
        response = requests.get('https://ipapi.co/json/', timeout=5)
        data = response.json()
        latitude = data.get('latitude')
        longitude = data.get('longitude')
        if latitude and longitude:
            return latitude, longitude
        else:
            return None
    except:
        return None

def get_emergency_contact(conn, email):
    sql_query = "SELECT emergency_name, emergency_phone FROM users WHERE email = %s"
    try:
        cursor = conn.cursor()
        cursor.execute(sql_query, (email,))
        result = cursor.fetchone()
        return result
    except mysql.connector.Error as e:
        print(f"Error: {e}")
        return None

# Twillio credentials

def send_emergency_whatsapp(to_phone, user_name, latitude, longitude):
    account_sid =''
    auth_token = ''
    from_phone = ''

    client = Client(account_sid, auth_token)
```



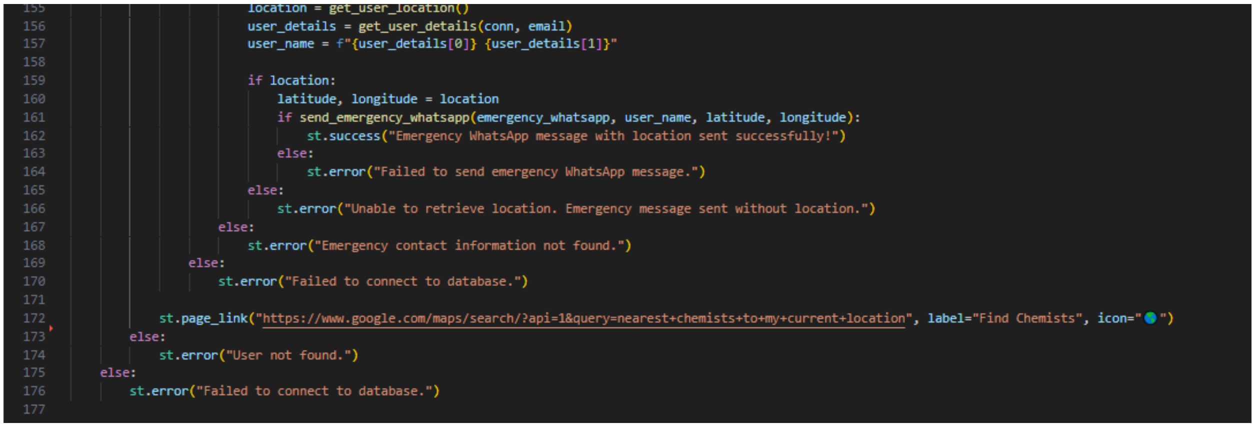
```python

    try:
        # Send text message
        text_message = f"Emergency Alert! {user_name} needs assistance."
        client.messages.create(
            body=text_message,
            from_=from_phone,
            to=f"whatsapp:{to_phone}"
        )

        # Send location message
        location_message = client.messages.create(
            from_=from_phone,
            to=f"whatsapp:{to_phone}",
            body=f"{user_name}'s Location",
            persistent_action=[f"geo:{latitude},{longitude}|{user_name}'s Location"]
        )

        return True
    except Exception as e:
        print(f"Error sending WhatsApp message: {e}")
        return False

def user_page(email):
    st.set_page_config(layout="wide")
```



```python
    # CSS for positioning

    st.markdown("""
    <style>
    .user-info {
        position: fixed;
        top: 60px;
        right: 20px;
        z-index: 1000;
        text-align: right;
    }
    </style>
    """, unsafe_allow_html=True)

    conn = create_connection()
    if conn is not None:
        user_details = get_user_details(conn, email)
        if user_details:
            first_name, last_name, user_email = user_details

            # User info at top right
            st.markdown(f"""
            <div class="user-info">
                <p><strong>{first_name} {last_name}</strong></p>
                <p>{user_email}</p>
            </div>
            """, unsafe_allow_html=True)

            # Logout button
            with st.container():
                st.markdown('<div class="logout-button">', unsafe_allow_html=True)
                if st.columns(13)[12].button("Logout"):
```



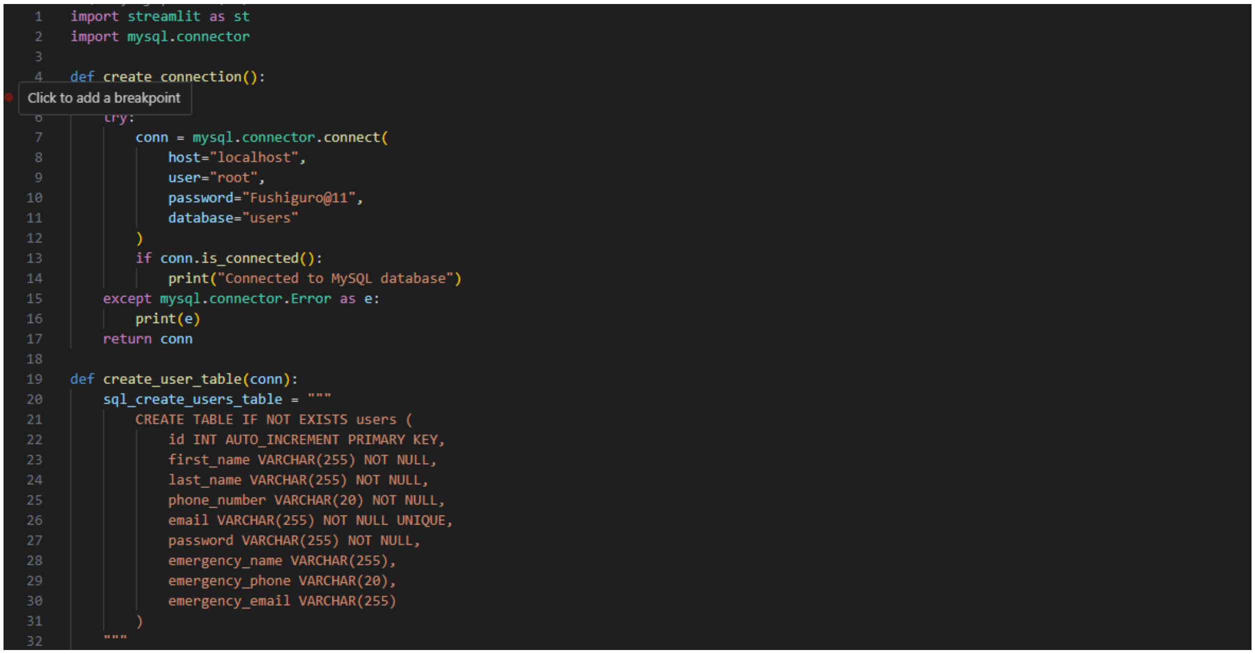
```python
                    st.session_state.logged_in = False
                    st.session_state.user_email = None
                    st.rerun()
                st.markdown('</div>', unsafe_allow_html=True)

            # Main Dashboard content
            st.markdown(f"""
            <div class="user">
                <p><strong>Welcome {first_name}</strong></p>
            </div>
            """, unsafe_allow_html=True)

             # recommendation Section
            st.subheader("Drug Recommendation")
            user_input = st.text_area("Describe your condition or symptoms:")
            if st.button("Get Recommendations"):
                if user_input:
                    recommendations = recommend_drugs(user_input)
                    st.write("Recommended drugs based on your description:")
                    for reason, drugs in recommendations:
                        st.write(f"**Reason:** {reason}")
                        st.write(f"**Recommended drugs:** {', '.join(drugs[:5])}")
                        st.write("---")
                else:
                    st.warning("Please enter a description of your condition.")

            if st.button("Send Emergency WhatsApp"):
                conn = create_connection()
                if conn is not None:
                    emergency_contact = get_emergency_contact(conn, email)
                    if emergency_contact:
                        emergency_name, emergency_whatsapp = emergency_contact
```



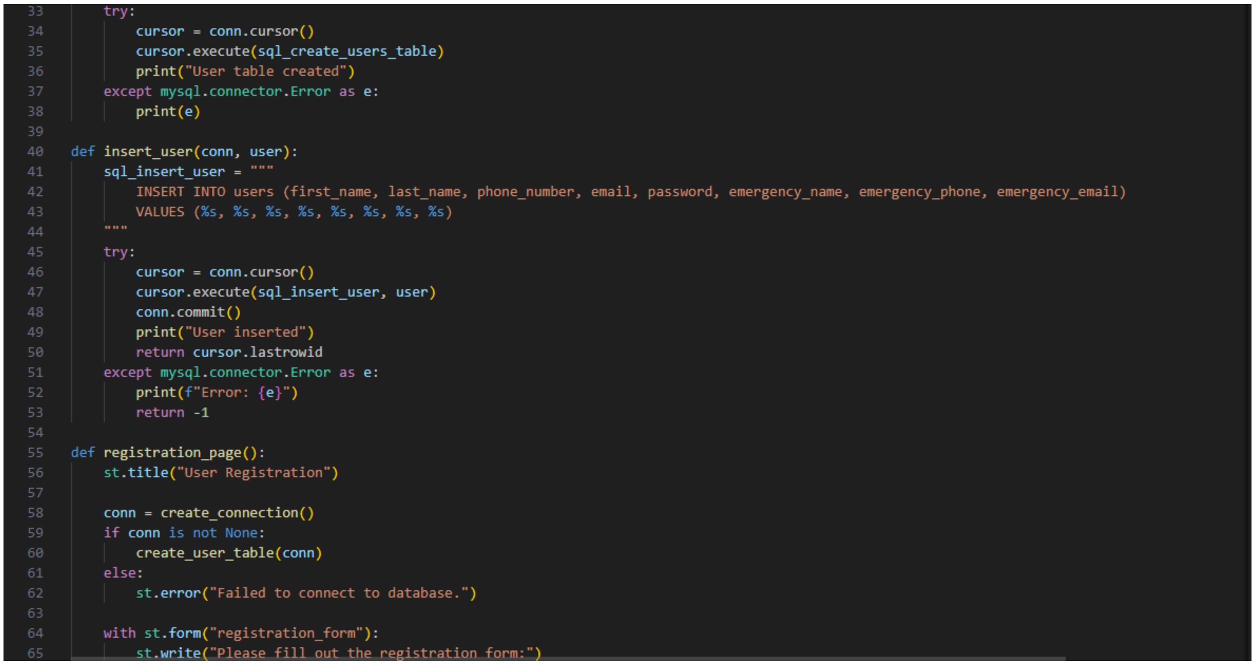
```
                        location = get_user_location()
                        user_details = get_user_details(conn, email)
                        user_name = f"{user_details[0]} {user_details[1]}"

                        if location:
                            latitude, longitude = location
                            if send_emergency_whatsapp(emergency_whatsapp, user_name, latitude, longitude):
                                st.success("Emergency WhatsApp message with location sent successfully!")
                            else:
                                st.error("Failed to send emergency WhatsApp message.")
                        else:
                            st.error("Unable to retrieve location. Emergency message sent without location.")
                    else:
                        st.error("Emergency contact information not found.")
                else:
                    st.error("Failed to connect to database.")

            st.page_link("https://www.google.com/maps/search/?api=1&query=nearest+chemists+to+my+current+location", label="Find Chemists", icon="🌎")
        else:
            st.error("User not found.")
    else:
        st.error("Failed to connect to database.")
```

Register_page.py

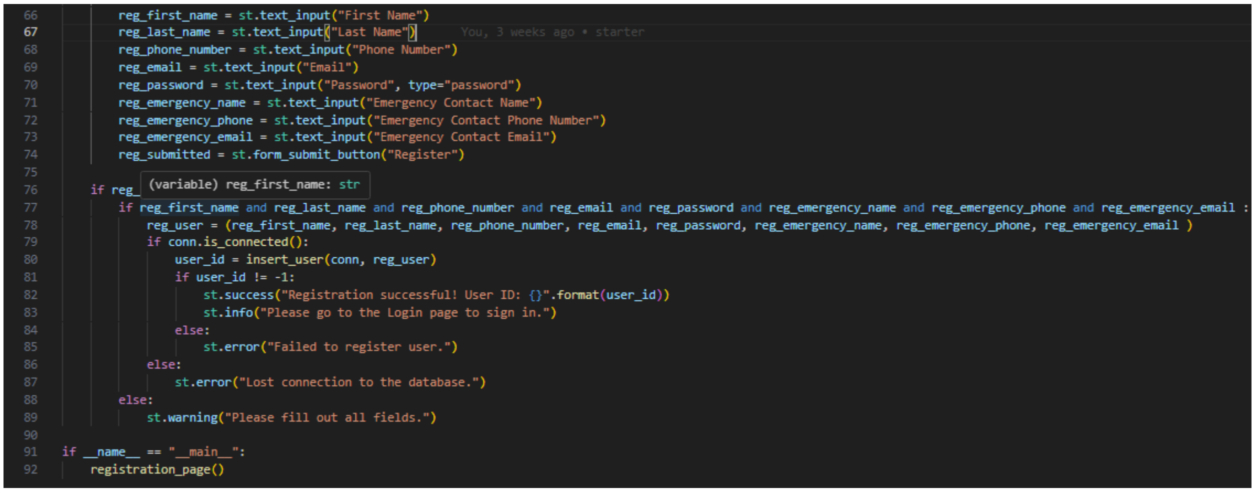
- Purpose: Creates a user registration page and stores user information in the MySQL database.
- Key Functions:
  - create_connection(): Establishes a connection to the MySQL database.
  - register_user(user_details): Registers a new user by inserting their details into the database.



```
import streamlit as st
import mysql.connector

def create_connection():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Fushiguro@11",
            database="users"
        )
        if conn.is_connected():
            print("Connected to MySQL database")
    except mysql.connector.Error as e:
        print(e)
    return conn

def create_user_table(conn):
    sql_create_users_table = """
        CREATE TABLE IF NOT EXISTS users (
            id INT AUTO_INCREMENT PRIMARY KEY,
            first_name VARCHAR(255) NOT NULL,
            last_name VARCHAR(255) NOT NULL,
            phone_number VARCHAR(20) NOT NULL,
            email VARCHAR(255) NOT NULL UNIQUE,
            password VARCHAR(255) NOT NULL,
            emergency_name VARCHAR(255),
            emergency_phone VARCHAR(20),
            emergency_email VARCHAR(255)
        )
    """
```



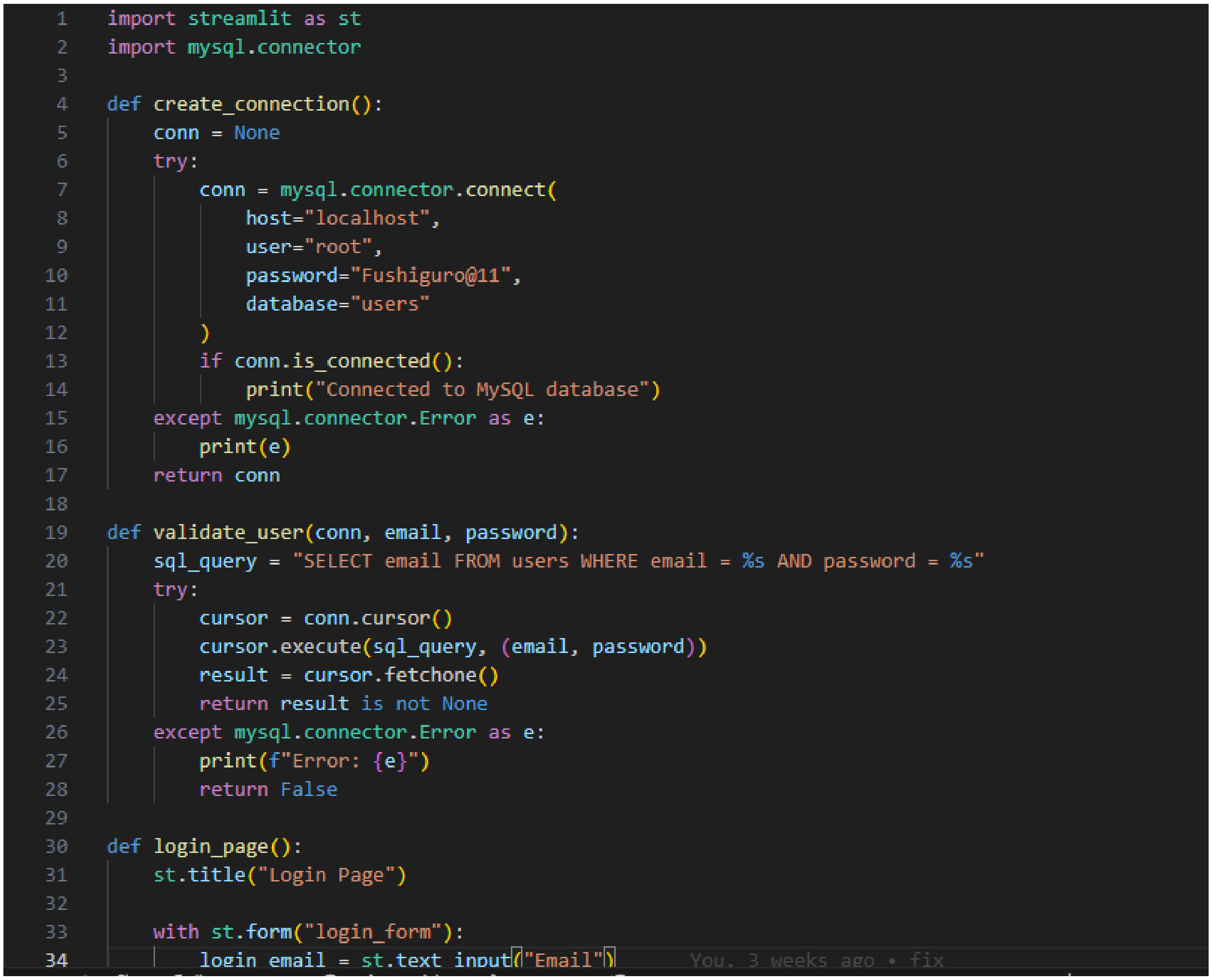
```
    try:
        cursor = conn.cursor()
        cursor.execute(sql_create_users_table)
        print("User table created")
    except mysql.connector.Error as e:
        print(e)

def insert_user(conn, user):
    sql_insert_user = """
        INSERT INTO users (first_name, last_name, phone_number, email, password, emergency_name, emergency_phone, emergency_email)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    """
    try:
        cursor = conn.cursor()
        cursor.execute(sql_insert_user, user)
        conn.commit()
        print("User inserted")
        return cursor.lastrowid
    except mysql.connector.Error as e:
        print(f"Error: {e}")
        return -1

def registration_page():
    st.title("User Registration")

    conn = create_connection()
    if conn is not None:
        create_user_table(conn)
    else:
        st.error("Failed to connect to database.")

    with st.form("registration_form"):
        st.write("Please fill out the registration form:")
```



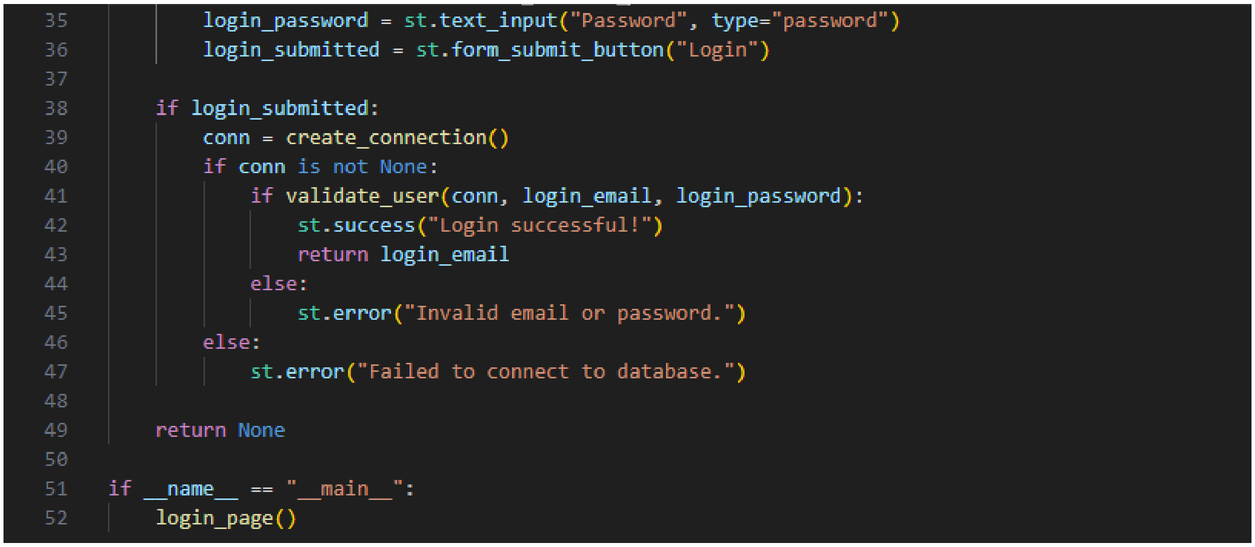
```
        reg_first_name = st.text_input("First Name")
        reg_last_name = st.text_input("Last Name")
        reg_phone_number = st.text_input("Phone Number")
        reg_email = st.text_input("Email")
        reg_password = st.text_input("Password", type="password")
        reg_emergency_name = st.text_input("Emergency Contact Name")
        reg_emergency_phone = st.text_input("Emergency Contact Phone Number")
        reg_emergency_email = st.text_input("Emergency Contact Email")
        reg_submitted = st.form_submit_button("Register")

    if reg_
        if reg_first_name and reg_last_name and reg_phone_number and reg_email and reg_password and reg_emergency_name and reg_emergency_phone and reg_emergency_email :
            reg_user = (reg_first_name, reg_last_name, reg_phone_number, reg_email, reg_password, reg_emergency_name, reg_emergency_phone, reg_emergency_email )
            if conn.is_connected():
                user_id = insert_user(conn, reg_user)
                if user_id != -1:
                    st.success("Registration successful! User ID: {}".format(user_id))
                    st.info("Please go to the Login page to sign in.")
                else:
                    st.error("Failed to register user.")
            else:
                st.error("Lost connection to the database.")
        else:
            st.warning("Please fill out all fields.")

if __name__ == "__main__":
    registration_page()
```

Login_page.py

- Purpose: Creates a login page and validates user credentials against the MySQL database.
- Key Functions:
  - create_connection(): Establishes a connection to the MySQL database.
  - validate_user(username, password): Validates the user’s credentials.



```python
import streamlit as st
import mysql.connector

def create_connection():
    conn = None
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Fushiguro@11",
            database="users"
        )
        if conn.is_connected():
            print("Connected to MySQL database")
    except mysql.connector.Error as e:
        print(e)
    return conn

def validate_user(conn, email, password):
    sql_query = "SELECT email FROM users WHERE email = %s AND password = %s"
    try:
        cursor = conn.cursor()
        cursor.execute(sql_query, (email, password))
        result = cursor.fetchone()
        return result is not None
    except mysql.connector.Error as e:
        print(f"Error: {e}")
        return False

def login_page():
    st.title("Login Page")

    with st.form("login_form"):
        login_email = st.text_input("Email")        You, 3 weeks ago • fix
```

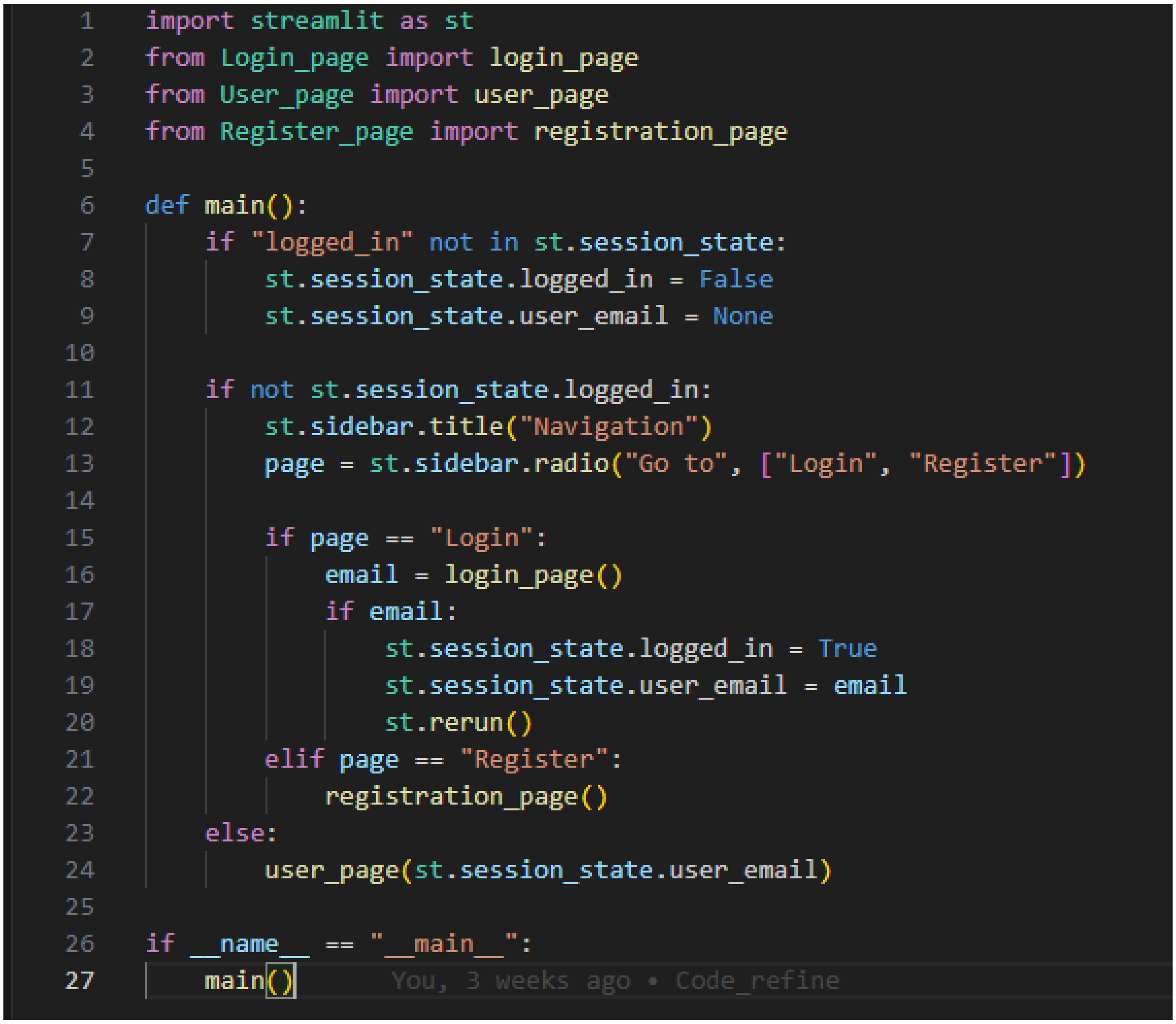


```python
        login_password = st.text_input("Password", type="password")
        login_submitted = st.form_submit_button("Login")

    if login_submitted:
        conn = create_connection()
        if conn is not None:
            if validate_user(conn, login_email, login_password):
                st.success("Login successful!")
                return login_email
            else:
                st.error("Invalid email or password.")
        else:
            st.error("Failed to connect to database.")

    return None

if __name__ == "__main__":
    login_page()
```

app.py

- Purpose: Integrates the login, registration, and user functionalities into a multi-page web application.

- Key Functions:
  - main(): Main function to run the Streamlit application, handling navigation between login, registration, and user pages.

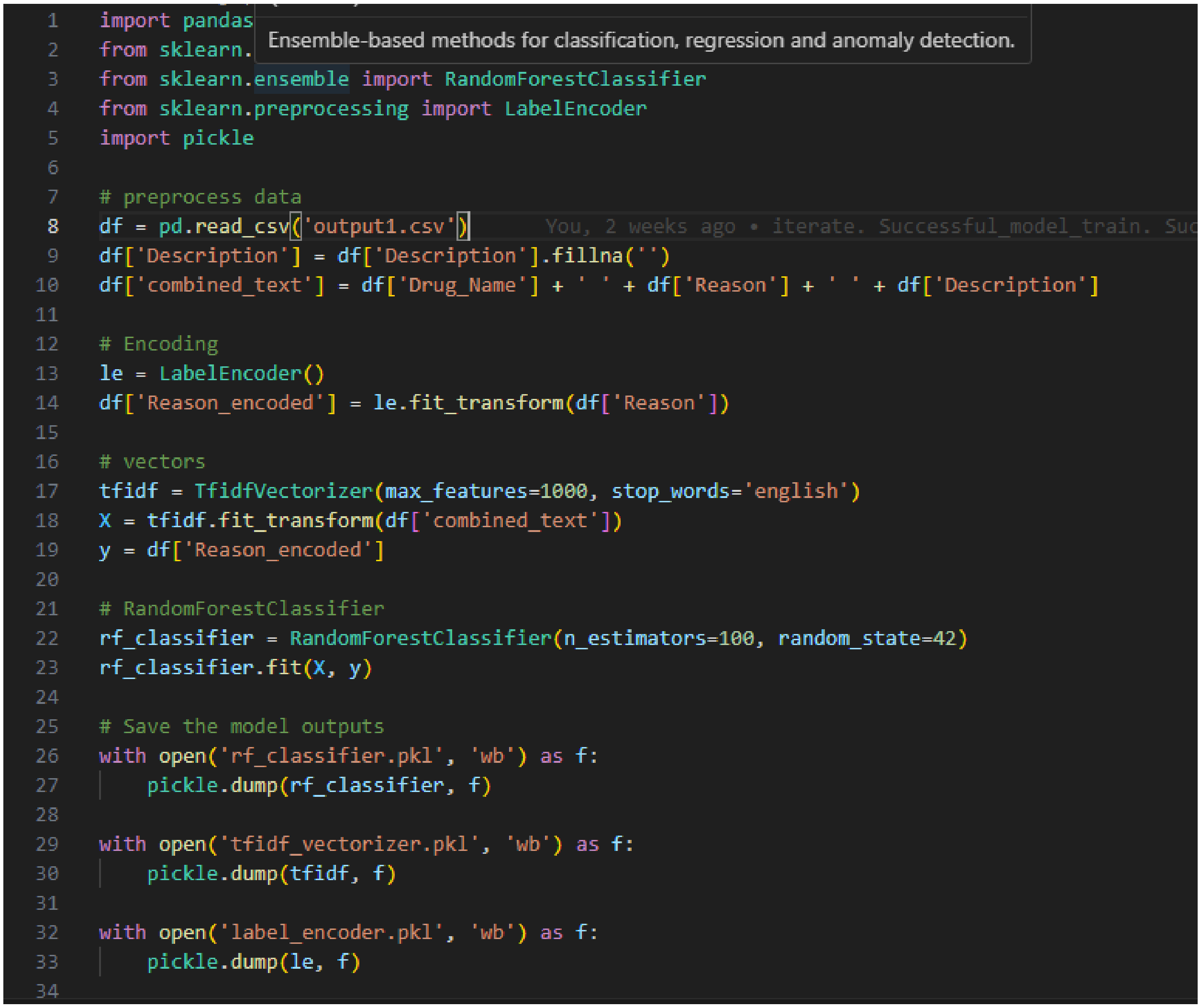


```
import streamlit as st
from Login_page import login_page
from User_page import user_page
from Register_page import registration_page

def main():
    if "logged_in" not in st.session_state:
        st.session_state.logged_in = False
        st.session_state.user_email = None

    if not st.session_state.logged_in:
        st.sidebar.title("Navigation")
        page = st.sidebar.radio("Go to", ["Login", "Register"])

        if page == "Login":
            email = login_page()
            if email:
                st.session_state.logged_in = True
                st.session_state.user_email = email
                st.rerun()
        elif page == "Register":
            registration_page()
    else:
        user_page(st.session_state.user_email)

if __name__ == "__main__":
    main()      You, 3 weeks ago • Code_refine
```

drug_recommender.py

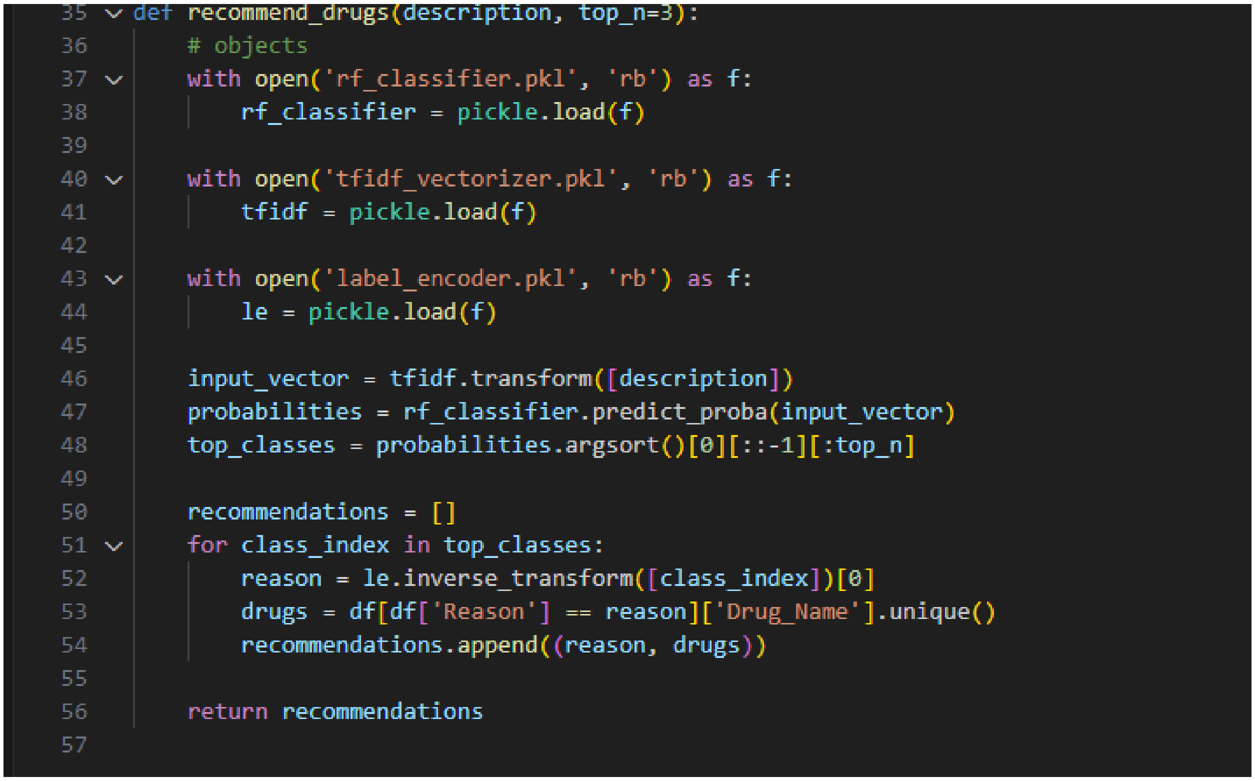
- Purpose: Recommends drugs based on user input descriptions using machine learning techniques.
- Key Functions:
  - recommend_drugs(description, top_n=3): Recommends top n drugs based on the input description.
- Key Components:
  - **Data Preprocessing**: Reads and preprocesses data from output1.csv.
  - **Model Training**: Trains a RandomForestClassifier on the preprocessed data.

- **Model Saving**: Saves the trained model, TF-IDF vectorizer, and label encoder using pickle.



Ensemble-based methods for classification, regression and anomaly detection.

```
import pandas
from sklearn.
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
import pickle

# preprocess data
df = pd.read_csv('output1.csv')        You, 2 weeks ago • iterate. Successful_model_train. Suc
df['Description'] = df['Description'].fillna('')
df['combined_text'] = df['Drug_Name'] + ' ' + df['Reason'] + ' ' + df['Description']

# Encoding
le = LabelEncoder()
df['Reason_encoded'] = le.fit_transform(df['Reason'])

# vectors
tfidf = TfidfVectorizer(max_features=1000, stop_words='english')
X = tfidf.fit_transform(df['combined_text'])
y = df['Reason_encoded']

# RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X, y)

# Save the model outputs
with open('rf_classifier.pkl', 'wb') as f:
    pickle.dump(rf_classifier, f)

with open('tfidf_vectorizer.pkl', 'wb') as f:
    pickle.dump(tfidf, f)

with open('label_encoder.pkl', 'wb') as f:
    pickle.dump(le, f)

```



```python
def recommend_drugs(description, top_n=3):
    # objects
    with open('rf_classifier.pkl', 'rb') as f:
        rf_classifier = pickle.load(f)

    with open('tfidf_vectorizer.pkl', 'rb') as f:
        tfidf = pickle.load(f)

    with open('label_encoder.pkl', 'rb') as f:
        le = pickle.load(f)

    input_vector = tfidf.transform([description])
    probabilities = rf_classifier.predict_proba(input_vector)
    top_classes = probabilities.argsort()[0][::-1][:top_n]

    recommendations = []
    for class_index in top_classes:
        reason = le.inverse_transform([class_index])[0]
        drugs = df[df['Reason'] == reason]['Drug_Name'].unique()
        recommendations.append((reason, drugs))

    return recommendations

```

How to Run the Project

Setup:

- Ensure you have Python installed.
- Install the required libraries using: *pip*
- pip install streamlit, mysql-connector-python, requests, twilio, scikit-learn, pandas

Database Setup:

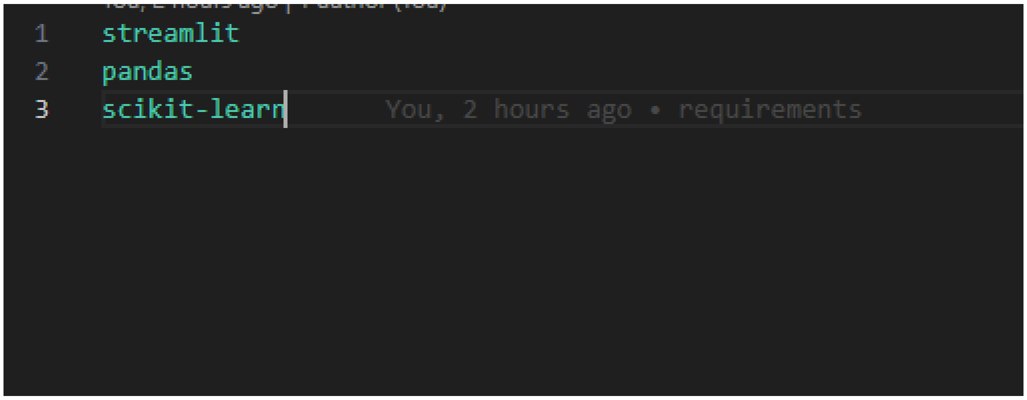
- Set up a MySQL database and create the necessary tables for storing user information.

Running the Application:

- Run the Streamlit application using: *streamlit run (app_name).py*
- streamlit run app.py

Additional Notes

- **Dependencies**: Ensure all required libraries are installed.



```
streamlit
pandas
scikit-learn
```

- **Configuration**: Update the database connection details in each script as needed.
- **Security**: Ensure sensitive information like database credentials and Twilio API keys are securely managed. Streamlit hosting uses secrets to enforce encrypted and secure credentials of the developer.

Machine Learning Model for Drug Recommendations

Overview

The drug recommendation system utilizes a machine learning model to suggest appropriate medications based on user-described symptoms/conditions. The model is implemented in the `drug_recommender.py` file and uses a combination of natural language processing (NLP) and classification techniques.

Model Architecture

The recommendation system employs a Random Forest Classifier coupled with TF-IDF (Term Frequency-Inverse Document Frequency) vectorization for text processing.

TF-IDF Vectorization

- Used to convert text descriptions into numerical features
- Implemented using `TfidfVectorizer` from scikit-learn
- Parameters:
- `max_features=1000`: Limits the vocabulary to the top 1000 terms
- `stop_words='english'`: Removes common English stop words

Random Forest Classifier

- - Ensemble learning method for classification
- - Implemented using `RandomForestClassifier` from scikit-learn
- - Parameters:
- - `n_estimators=100`: Uses 100 trees in the forest
- - `random_state=42`: Ensures reproducibility of results

Data Preprocessing

- 1. The model uses a CSV file (`output1.csv`) containing drug information
- 2. Text data is combined from 'Drug_Name', 'Reason', and 'Description' columns
- 3. The 'Reason' column is encoded using `LabelEncoder` for classification

Training Process

- 1. The combined text data is vectorized using TF-IDF
- 2. The encoded 'Reason' serves as the target variable
- 3. The Random Forest model is trained on the TF-IDF vectors and encoded reasons

Recommendation Process

- 1. User input is transformed using the same TF-IDF vectorizer
- 2. The model predicts probabilities for each possible reason
- 3. Top N reasons are selected based on these probabilities
- 4. Drugs associated with these reasons are retrieved from the dataset

Model Persistence

The trained model and associated transformers are saved to disk using pickle:

- - `rf_classifier.pkl`: Trained Random Forest model
- - `tfidf_vectorizer.pkl`: Fitted TF-IDF vectorizer

- - `label_encoder.pkl`: Fitted Label Encoder

Usage in Application

The `recommend_drugs` function in `drug_recommender.py`:

- Loads the saved model and transformers
- Processes user input
- Makes predictions
- Returns top N recommendations with associated drugs

Performance and Limitations

- The model's performance depends on the quality and quantity of the training data
- It may not capture complex medical relationships or contraindications
- Recommendations should be treated as suggestions and not as professional medical advice

Future Improvements

- Incorporate more advanced NLP techniques like word embeddings or BERT
- Implement a more sophisticated ranking system for drug recommendations
- Include a feedback loop to continuously improve recommendations based on user feedback
- Integrate with a regularly updated medical database for more accurate and current recommendations
- Include Government and Health organization APIs that will give a massive amount of data for training.
- Include Geocoding for location information.

https://safespace.streamlit.app/