server\server.py

```python
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS
from datetime import datetime
import sqlite3, pytz, os, random

app = Flask(__name__)
CORS(app)
DATABASE = 'DATABASE.db'

# Time zone settings
local_tz = pytz.timezone('America/Argentina/Buenos_Aires')

# Database connection
def get_db_connection():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn

def query(table, where=None, equals=None, ordertime=False):
    conn = get_db_connection()
    cursor = conn.cursor()

    # Building the base SQL query with an ORDER BY clause for descending order on timestamp
    sql_query = f'SELECT * FROM {table}'

    if where is not None:
        sql_query += f' WHERE {where} = "{equals}"'

    if ordertime:
        sql_query += ' ORDER BY timestamp DESC LIMIT 1'

    # Execute the query
    cursor.execute(sql_query)

    rows = cursor.fetchall()

    # Convert rows to a list of dictionaries
    results = [dict(row) for row in rows]

    # Get column headers
    headers = results[0].keys() if results else []

    conn.close()

    return results, headers

def convert_to_local_time(utc_timestamp):
    # Convert a UTC timestamp to local time (GMT-4)
    if not utc_timestamp:
        return None
    utc_time = datetime.strptime(utc_timestamp, '%Y-%m-%d %H:%M:%S')
    utc_time = pytz.utc.localize(utc_time)
    local_time = utc_time.astimezone(local_tz)
    return local_time.strftime('%Y-%m-%d %H:%M:%S')

@app.route('/getrange/<int:plant_id>', methods=['GET'])
def plantRanges(plant_id):
    conn = get_db_connection()
    cursor = conn.cursor()

    # Step 1: Get the plantType_id from the `plants` table based on the given plant_id
    cursor.execute("SELECT plantType_id FROM plants WHERE plant_id = ?", (plant_id,))
    plant_type_row = cursor.fetchone()

    if plant_type_row:
        plant_type_id = plant_type_row['plantType_id']

        # Step 2: Use the retrieved plantType_id to get the range data from the
`plant_types` table
        cursor.execute("SELECT * FROM plant_types WHERE id = ?", (plant_type_id,))
        plant_type_data = cursor.fetchone()

        # If data for plant type exists, convert it to a dictionary and return it
        if plant_type_data:
            plant_type_dict = dict(plant_type_data)
            conn.close()
            return jsonify(plant_type_dict), 200
        else:
            conn.close()
            return jsonify({'error': 'No data found for the specified plant type'}), 404
    else:
        conn.close()
        return jsonify({'error': 'No plant found with the specified ID'}), 404


#/plants to see what plants are in the database
@app.route('/plants', methods=['GET'])
def get_plants():
    data, headers = query('plants')
    # return render_template('table.html', rows=rows, headers=headers)
    return jsonify(data), 200

@app.route('/plants/<user_id>', methods=['GET'])
def get_plants_spc(user_id):
    data, headers = query('plants', where='user_id', equals=user_id)
    # return render_template('table.html', rows=rows, headers=headers)
    return jsonify(data), 200

#/plant_types to get the types of plants
@app.route('/plant_types', methods=['GET'])
def get_plant_types():
    data, headers = query('plant_types')
    # return render_template('table.html', rows=rows, headers=headers)
    return jsonify(data), 200

```

```python
#/users to get a list of all users
@app.route('/users', methods=['GET'])
def get_users():
    data, headers = query('users')
    # return render_template('table.html', rows=rows, headers=headers)
    return jsonify(data), 200

#/data gets all of the entries in the data table
@app.route('/data', methods=['GET'])
def get_data():
    # rows, headers = query('data')
    # # Convert date_added from UTC to local time for each row
    # for row in rows:
    #     row['date_added'] = convert_to_local_time(row.get('date_added'))
    # return render_template('table.html', rows=rows, headers=headers)

    data, headers = query('data')

    for row in data:
        row['timestamp'] = convert_to_local_time(row['timestamp'])

    return jsonify(data), 200

#/data/<plant_id> gets entries for a specific plant
@app.route('/data/<plant_id>', methods=['GET'])
def get_data_specific(plant_id):
    data, headers = query('data', where='plant_id', equals=plant_id)

    # Convert date_added from UTC to local time for each row
    for row in data:
        row['timestamp'] = convert_to_local_time(row['timestamp'])

    # return render_template('table.html', rows=rows, headers=headers)

    return jsonify(data), 200

# logs plant data
# usage example: POST {plant_id=2, soil_humidity=134.42, light_level=835.43,
temperature=23.42}
@app.route('/logdata', methods=['POST'])
def log_data():
    data = request.get_json()  # Get JSON data from the request
    # ID
    try:
        plant_id = data.get('plant_id')
    except:
        return jsonify({'error': 'Plant id is required'}), 400

    # Soil
    try:
        hum = data.get('soil_humidity')
    except:
        return jsonify({'error': 'Humidity is required'}), 400

    # Light
    try:
        luz = data.get('light_level')
    except:
        return jsonify({'error': 'Light level is required'}), 400

    # Temp
    try:
        temp = data.get('temperature')
    except:
        return jsonify({'error': 'Temperature is required'}), 400


    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute('INSERT INTO data (plant_id, soil_humidity, light_level, temperature) VALUES (?, ?, ?, ?)', (plant_id, hum, luz, temp))
    conn.commit()

    new_data_id = cursor.lastrowid  # Get the ID of the newly inserted row

    conn.close()

    return jsonify({'id': new_data_id, 'plant_id': plant_id, 'Humidity': hum, 'Light Level': luz, 'Temperature': temp}), 201

@app.route('/addplant', methods=['POST'])
def add_plant():
    data = request.get_json()  # Get JSON data from the request
    user_id = data.get('user_id')
    plantType = data.get('plantType')

    if not user_id:
        return jsonify({'error': 'user_id is required'}), 400

    if not plantType:
        return jsonify({'error': 'plantType is required'}), 400

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute('INSERT INTO plants (user_id, plantType_id) VALUES (?, ?)', (user_id, plantType))
    conn.commit()

    new_plant_id = cursor.lastrowid  # Get the ID of the newly inserted row

    conn.close()

    return jsonify({'plant_id': new_plant_id, 'user_id': user_id, 'plantType_id': plantType}), 201

@app.route('/login', methods=['POST'])
```

```python
def user_login():
    data = request.get_json()
    try:
        username = data.get('username')
    except:
        return 'No username key found in JSON body', 400

    try:
        password = data.get('password')
    except:
        return 'No password key found in JSON body', 400

    if not username:
        return 'username not detected \n', 400
    if not password:
        return 'password not detected \n', 400

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute(f'SELECT password FROM users WHERE name = "{username}"')

    rows = cursor.fetchall()

    results = [dict(row) for row in rows]
    try:
        saved_password = results[0]['password']
    except:
        return 'Wrong username of password \n', 409

    if password != saved_password:
        conn.close()
        return 'Wrong username or password. \n', 409

    cursor.execute(f'SELECT id FROM users WHERE name = "{username}"')
    rows = cursor.fetchall()
    results = [dict(row) for row in rows]
    user_id = results[0]

    return jsonify(user_id), 201

@app.route('/alert', methods=["POST"])
def log_alert():
    data = request.get_json()
    message = data.get('message')
    plant_id = data.get('plant_id')

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute('INSERT INTO alerts (message, plant_id) VALUES (?, ?)', (message, 
plant_id))
    conn.commit()

    conn.close()

    return jsonify({'plant_id': plant_id, 'message': message}), 201

@app.route('/getalert', methods=['GET'])
def get_alert():
    conn = get_db_connection()
    cursor = conn.cursor()

    # SQL query to get the latest alert based on timestamp
    sql_query = 'SELECT * FROM alerts ORDER BY timestamp DESC LIMIT 1'

    # Execute the query to fetch the latest row
    cursor.execute(sql_query)
    row = cursor.fetchone()

    # If a row is found, process it
    if row:
        columns = [desc[0] for desc in cursor.description]  # Get column names
        result = dict(zip(columns, row))  # Map columns to values for JSON output

        # Delete the fetched row
        delete_query = 'DELETE FROM alerts WHERE timestamp = ?'
        cursor.execute(delete_query, (row[columns.index('timestamp')],))
        conn.commit()  # Commit the delete operation
    else:
        result = ""  # Return an empty JSON object if no alerts are found

    conn.close()

    return jsonify(result), 200

@app.route('/create_data/<ammount>', methods=["GET"])
def create_data(ammount):
    conn = get_db_connection()

    for id in range(5):
        plant_id = id + 1
        for i in range(int(ammount)):
            soil_humidity = random.randint(200, 3000)
            light_level = random.randint(100, 1000)
            temperature = random.randint(-8, 40)

            conn.cursor().execute('INSERT INTO data (plant_id, soil_humidity, light_level, 
temperature) VALUES (?, ?, ?, ?)', (plant_id, soil_humidity, light_level, temperature))
            conn.commit()

    return(f"Created {ammount} more entries for ids 1-5 \n"), 201

def create_db():
    if not os.path.exists('test.db'):

        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            password TEXT DEFAULT ""
        )
        ''')

        cursor.execute('''
        CREATE TABLE IF NOT EXISTS data (
            plant_id INTEGER NOT NULL,
            soil_humidity REAL NOT NULL,
            light_level REAL NOT NULL,
            temperature REAL NOT NULL,
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
        )
        ''')

        cursor.execute('''
        CREATE TABLE IF NOT EXISTS plants (
            plant_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
            user_id INTEGER NOT NULL,
            plantType_id INTEGER NOT NULL
        )
        ''')

        cursor.execute('''
        CREATE TABLE IF NOT EXISTS alerts (
            message TEXT NOT NULL,
            plant_id INTEGER NOT NULL,
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
        )
        ''')

        cursor.execute('''
        CREATE TABLE IF NOT EXISTS plant_types (
            id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  -- Fixed: Corrected 'PRIMERY' to 'PRIMARY'
            nombre TEXT NOT NULL,
            max_hum REAL NOT NULL,
            min_hum REAL NOT NULL,
            max_temp REAL NOT NULL,
            min_temp REAL NOT NULL,
            max_luz REAL NOT NULL,
            min_luz REAL NOT NULL
        )
        ''')

        conn.commit()
        conn.close()

        print("Database created ", 201)

        create_data(2)
        return

    print("Database already exists ", 418)

if not os.path.exists('test.db'):
    create_db()

try:
    PORT = os.environ['PORT']
except:
    PORT = 8080

try:
    debug = os.environ['DEBUG']
except:
    debug = ""

if debug == 1:
    debug = True
else:
    debug = False

if __name__ == '__main__':
    app.run(debug=debug, host='0.0.0.0', port=PORT)


```