

## server\server.py

```
1 from flask import Flask, request, jsonify, render_template
2 from flask_cors import CORS
3 from datetime import datetime
4 import sqlite3, pytz, os, random
5
6 app = Flask(__name__)
7 CORS(app)
8 DATABASE = 'DATABASE.db'
9
10 # Time zone settings
11 local_tz = pytz.timezone('America/Argentina/Buenos_Aires')
12
13 # Database connection
14 def get_db_connection():
15     conn = sqlite3.connect(DATABASE)
16     conn.row_factory = sqlite3.Row
17     return conn
18
19 def query(table, where=None, equals=None, ordertime=False):
20     conn = get_db_connection()
21     cursor = conn.cursor()
22
23     # Building the base SQL query with an ORDER BY clause for descending order on timestamp
24     sql_query = f'SELECT * FROM {table}'
25
26     if where is not None:
27         sql_query += f' WHERE {where} = "{equals}"'
28
29     if ordertime:
30         sql_query += ' ORDER BY timestamp DESC LIMIT 1'
31
32     # Execute the query
33     cursor.execute(sql_query)
34
35     rows = cursor.fetchall()
36
37     # Convert rows to a list of dictionaries
38     results = [dict(row) for row in rows]
39
40     # Get column headers
41     headers = results[0].keys() if results else []
42
43     conn.close()
44
45     return results, headers
46
47 def convert_to_local_time(utc_timestamp):
48     # Convert a UTC timestamp to local time (GMT-4)
49     if not utc_timestamp:
50         return None
51     utc_time = datetime.strptime(utc_timestamp, '%Y-%m-%d %H:%M:%S')
```

```
52     utc_time = pytz.utc.localize(utc_time)
53     local_time = utc_time.astimezone(local_tz)
54     return local_time.strftime('%Y-%m-%d %H:%M:%S')
55
56 @app.route('/getrange/<int:plant_id>', methods=['GET'])
57 def plantRanges(plant_id):
58     conn = get_db_connection()
59     cursor = conn.cursor()
60
61     # Step 1: Get the plantType_id from the `plants` table based on the given plant_id
62     cursor.execute("SELECT plantType_id FROM plants WHERE plant_id = ?", (plant_id,))
63     plant_type_row = cursor.fetchone()
64
65     if plant_type_row:
66         plant_type_id = plant_type_row['plantType_id']
67
68         # Step 2: Use the retrieved plantType_id to get the range data from the
        # `plant_types` table
69         cursor.execute("SELECT * FROM plant_types WHERE id = ?", (plant_type_id,))
70         plant_type_data = cursor.fetchone()
71
72         # If data for plant type exists, convert it to a dictionary and return it
73         if plant_type_data:
74             plant_type_dict = dict(plant_type_data)
75             conn.close()
76             return jsonify(plant_type_dict), 200
77         else:
78             conn.close()
79             return jsonify({'error': 'No data found for the specified plant type'}), 404
80     else:
81         conn.close()
82         return jsonify({'error': 'No plant found with the specified ID'}), 404
83
84
85 #/plants to see what plants are in the database
86 @app.route('/plants', methods=['GET'])
87 def get_plants():
88     data, headers = query('plants')
89     # return render_template('table.html', rows=rows, headers=headers)
90     return jsonify(data), 200
91
92 @app.route('/plants/<user_id>', methods=['GET'])
93 def get_plants_spc(user_id):
94     data, headers = query('plants', where='user_id', equals=user_id)
95     # return render_template('table.html', rows=rows, headers=headers)
96     return jsonify(data), 200
97
98 #/plant_types to get the types of plants
99 @app.route('/plant_types', methods=['GET'])
100 def get_plant_types():
101     data, headers = query('plant_types')
102     # return render_template('table.html', rows=rows, headers=headers)
103     return jsonify(data), 200
104
```

```
105 #/users to get a list of all users
106 @app.route('/users', methods=['GET'])
107 def get_users():
108     data, headers = query('users')
109     # return render_template('table.html', rows=rows, headers=headers)
110     return jsonify(data), 200
111
112 #/data gets all of the entries in the data table
113 @app.route('/data', methods=['GET'])
114 def get_data():
115     # rows, headers = query('data')
116     # # Convert date_added from UTC to local time for each row
117     # for row in rows:
118     #     row['date_added'] = convert_to_local_time(row.get('date_added'))
119     # return render_template('table.html', rows=rows, headers=headers)
120
121     data, headers = query('data')
122
123     for row in data:
124         row['timestamp'] = convert_to_local_time(row['timestamp'])
125
126     return jsonify(data), 200
127
128 #/data/<plant_id> gets entries for a specific plant
129 @app.route('/data/<plant_id>', methods=['GET'])
130 def get_data_specific(plant_id):
131     data, headers = query('data', where='plant_id', equals=plant_id)
132
133     # Convert date_added from UTC to local time for each row
134     for row in data:
135         row['timestamp'] = convert_to_local_time(row['timestamp'])
136
137     # return render_template('table.html', rows=rows, headers=headers)
138
139     return jsonify(data), 200
140
141 # logs plant data
142 # usage example: POST {plant_id=2, soil_humidity=134.42, light_level=835.43,
143 # temperature=23.42}
144 @app.route('/logdata', methods=['POST'])
145 def log_data():
146     data = request.get_json() # Get JSON data from the request
147     # ID
148     try:
149         plant_id = data.get('plant_id')
150     except:
151         return jsonify({'error': 'Plant id is required'}), 400
152
153     # Soil
154     try:
155         hum = data.get('soil_humidity')
156     except:
157         return jsonify({'error': 'Humidity is required'}), 400
```

```
158     # Light
159     try:
160         luz = data.get('light_level')
161     except:
162         return jsonify({'error': 'Light level is required'}), 400
163
164     # Temp
165     try:
166         temp = data.get('temperature')
167     except:
168         return jsonify({'error': 'Temperature is required'}), 400
169
170
171     conn = get_db_connection()
172     cursor = conn.cursor()
173
174     cursor.execute('INSERT INTO data (plant_id, soil_humidity, light_level, temperature)
VALUES (?, ?, ?, ?)', (plant_id, hum, luz, temp))
175     conn.commit()
176
177     new_data_id = cursor.lastrowid # Get the ID of the newly inserted row
178
179     conn.close()
180
181     return jsonify({'id': new_data_id, 'plant_id': plant_id, 'Humidity': hum, 'Light Level':
luz, 'Temperature': temp}), 201
182
183 @app.route('/addplant', methods=['POST'])
184 def add_plant():
185     data = request.get_json() # Get JSON data from the request
186     user_id = data.get('user_id')
187     plantType = data.get('plantType')
188
189     if not user_id:
190         return jsonify({'error': 'user_id is required'}), 400
191
192     if not plantType:
193         return jsonify({'error': 'plantType is required'}), 400
194
195     conn = get_db_connection()
196     cursor = conn.cursor()
197
198     cursor.execute('INSERT INTO plants (user_id, plantType_id) VALUES (?, ?)', (user_id,
plantType))
199     conn.commit()
200
201     new_plant_id = cursor.lastrowid # Get the ID of the newly inserted row
202
203     conn.close()
204
205     return jsonify({'plant_id': new_plant_id, 'user_id': user_id, 'plantType_id':
plantType}), 201
206
207 @app.route('/login', methods=['POST'])
```

```
208 def user_login():
209     data = request.get_json()
210     try:
211         username = data.get('username')
212     except:
213         return 'No username key found in JSON body', 400
214
215     try:
216         password = data.get('password')
217     except:
218         return 'No password key found in JSON body', 400
219
220     if not username:
221         return 'username not detected \n', 400
222     if not password:
223         return 'password not detected \n', 400
224
225     conn = get_db_connection()
226     cursor = conn.cursor()
227
228     cursor.execute(f'SELECT password FROM users WHERE name = "{username}"')
229
230     rows = cursor.fetchall()
231
232     results = [dict(row) for row in rows]
233     try:
234         saved_password = results[0]['password']
235     except:
236         return 'Wrong username of password \n', 409
237
238     if password != saved_password:
239         conn.close()
240         return 'Wrong username or password. \n', 409
241
242     cursor.execute(f'SELECT id FROM users WHERE name = "{username}"')
243     rows = cursor.fetchall()
244     results = [dict(row) for row in rows]
245     user_id = results[0]
246
247     return jsonify(user_id), 201
248
249 @app.route('/alert', methods=["POST"])
250 def log_alert():
251     data = request.get_json()
252     message = data.get('message')
253     plant_id = data.get('plant_id')
254
255     conn = get_db_connection()
256     cursor = conn.cursor()
257
258     cursor.execute('INSERT INTO alerts (message, plant_id) VALUES (?, ?)', (message,
plant_id))
259     conn.commit()
260
```

```
261     conn.close()
262
263     return jsonify({'plant_id': plant_id, 'message': message}), 201
264
265 @app.route('/getalert', methods=['GET'])
266 def get_alert():
267     conn = get_db_connection()
268     cursor = conn.cursor()
269
270     # SQL query to get the latest alert based on timestamp
271     sql_query = 'SELECT * FROM alerts ORDER BY timestamp DESC LIMIT 1'
272
273     # Execute the query to fetch the latest row
274     cursor.execute(sql_query)
275     row = cursor.fetchone()
276
277     # If a row is found, process it
278     if row:
279         columns = [desc[0] for desc in cursor.description] # Get column names
280         result = dict(zip(columns, row)) # Map columns to values for JSON output
281
282         # Delete the fetched row
283         delete_query = 'DELETE FROM alerts WHERE timestamp = ?'
284         cursor.execute(delete_query, (row[columns.index('timestamp')],))
285         conn.commit() # Commit the delete operation
286     else:
287         result = "" # Return an empty JSON object if no alerts are found
288
289     conn.close()
290
291     return jsonify(result), 200
292
293 @app.route('/create_data/<ammount>', methods=["GET"])
294 def create_data(ammount):
295     conn = get_db_connection()
296
297     for id in range(5):
298         plant_id = id + 1
299         for i in range(int(ammount)):
300             soil_humidity = random.randint(200, 3000)
301             light_level = random.randint(100, 1000)
302             temperature = random.randint(-8, 40)
303
304             conn.cursor().execute('INSERT INTO data (plant_id, soil_humidity, light_level,
temperature) VALUES (?, ?, ?, ?)', (plant_id, soil_humidity, light_level, temperature))
305             conn.commit()
306
307     return(f"Created {ammount} more entries for ids 1-5 \n"), 201
308
309 def create_db():
310     if not os.path.exists('test.db'):
311
312         conn = get_db_connection()
313         cursor = conn.cursor()
```

```
314
315     cursor.execute('''
316     CREATE TABLE IF NOT EXISTS users (
317         id INTEGER PRIMARY KEY AUTOINCREMENT,
318         name TEXT NOT NULL,
319         password TEXT DEFAULT ""
320     )
321     ''')
322
323     cursor.execute('''
324     CREATE TABLE IF NOT EXISTS data (
325         plant_id INTEGER NOT NULL,
326         soil_humidity REAL NOT NULL,
327         light_level REAL NOT NULL,
328         temperature REAL NOT NULL,
329         timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
330     )
331     ''')
332
333     cursor.execute('''
334     CREATE TABLE IF NOT EXISTS plants (
335         plant_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
336         user_id INTEGER NOT NULL,
337         plantType_id INTEGER NOT NULL
338     )
339     ''')
340
341     cursor.execute('''
342     CREATE TABLE IF NOT EXISTS alerts (
343         message TEXT NOT NULL,
344         plant_id INTEGER NOT NULL,
345         timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
346     )
347     ''')
348
349     cursor.execute('''
350     CREATE TABLE IF NOT EXISTS plant_types (
351         id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, -- Fixed: Corrected 'PRIMERY' to
'PRIMARY'
352         nombre TEXT NOT NULL,
353         max_hum REAL NOT NULL,
354         min_hum REAL NOT NULL,
355         max_temp REAL NOT NULL,
356         min_temp REAL NOT NULL,
357         max_luz REAL NOT NULL,
358         min_luz REAL NOT NULL
359     )
360     ''')
361
362     conn.commit()
363     conn.close()
364
365     print("Database created ", 201)
366
```

```
367         create_data(2)
368         return
369
370     print("Database already exists ", 418)
371
372 if not os.path.exists('test.db'):
373     create_db()
374
375 try:
376     PORT = os.environ['PORT']
377 except:
378     PORT = 8080
379
380 try:
381     debug = os.environ['DEBUG']
382 except:
383     debug = ""
384
385 if debug == 1:
386     debug = True
387 else:
388     debug = False
389
390 if __name__ == '__main__':
391     app.run(debug=debug, host='0.0.0.0', port=PORT)
392
393
```