

<pre> 1 import json 2 import requests 3 from flask import Flask, render_template, jsonify, request, redirect, url_for, session 4 import urllib3 5 from datetime import datetime 6 from functools import wraps </pre>	<ul style="list-style-type: none"> • <code>json</code>: to parse API responses and handle JSON errors • <code>requests</code>: for HTTP calls to the Ecowitt API • <code>flask</code> imports: core Flask object and helper functions for rendering templates, managing sessions, and building redirects • <code>urllib3</code>: to suppress SSL verification warnings (CERTIFICATE_VERIFY_FAILED) • <code>datetime</code>: timestamp conversion from API epoch seconds • <code>functools.wraps</code>: decorator utility to preserve function metadata
<pre> 1 app = Flask(__name__) 2 app.secret_key = 'your_secret_key_here' </pre>	<ul style="list-style-type: none"> • Initialize Flask app. • <code>secret_key</code> secures session cookies; should be moved to environment variables for safety
<pre> STATIONS = { 'station1': { 'name': 'G Block', 'api_url': 'https://api.ecowitt.net/api/v3/device/real_time?...&mac=D8:BC:38:AA:96:AF&call_back=all' }, 'station2': { 'name': 'Art Block', 'api_url': 'https://api.ecowitt.net/api/v3/device/real_time?...&mac=D8:BC:38:AA:EF:E1&call_back=all' } } </pre>	<ul style="list-style-type: none"> • Keys (<code>station1</code>, <code>station2</code>) used in URL query parameters. • Values include a human-readable <code>name</code> and the full <code>api_url</code>.
<pre> def safe_float(value): """Safely converts a value to a float, handling None or non-numeric strings.""" if value is None: return None try: return float(value) except (ValueError, TypeError): return None </pre>	<p>Normalizes numeric inputs; returns <code>None</code> on invalid or missing data.</p> <p>Ensures downstream calculations won't crash if the API returns unexpected types</p>
<pre> def to_celsius(temp, unit_str): """Converts a temperature value to Celsius if it's in Fahrenheit.""" temp_val = safe_float(temp) if temp_val is None: return None if unit_str and 'f' in unit_str.lower(): # Formula: C = (F - 32) * 5 / 9 return (temp_val - 32) * 5 / 9 return temp_val # Assume Celsius if not Fahrenheit </pre>	<p>Unit-aware conversion from Fahrenheit to Celsius.</p> <p>Gracefully handles missing unit information</p>

<pre>def fetch_ecowitt_data(api_url, station_name, display_units='c'): """Fetches and processes weather data from the Ecowitt API.""" try: response = requests.get(api_url, timeout=10) response.raise_for_status() api_data = response.json().get('data', {}) if not api_data: return None, "Received empty data payload from API."</pre>		<ul style="list-style-type: none"> • Perform HTTP GET with a 10 s timeout. • <code>raise_for_status()</code> throws <code>HTTPError</code> on bad status codes. • Check for empty data payload; return a user-friendly error message if missing
<pre>processed_observation = { 'local_time_obj': formatted_time_obj, 'air_temp': convert_display_temperature(temp_c, display_units), 'apparent_temp': convert_display_temperature(feels_like_c, display_units), 'humidity': safe_float(outdoor_data.get('humidity', {}).get('value')), 'wind_spd_kmh': safe_float(wind_data.get('wind_speed', {}).get('value')), 'gust_kmh': safe_float(wind_data.get('wind_gust', {}).get('value')), 'pressure_val': safe_float(pressure_data.get('value')), 'pressure_unit': pressure_data.get('unit', 'hPa'), 'rain_since_9am': safe_float(rainfall_data.get('daily', {}).get('value')) }</pre>		<ul style="list-style-type: none"> • Build a normalized dict for one observation. • Units: speed in km/h, pressure in hPa by default. <p>This payload can easily be extended to include additional fields</p>
<pre>return { 'station_name': station_name, 'last_updated_product': f"Last updated: {formatted_time_obj['date']} {formatted_time_obj['time']}", 'observations': [processed_observation] }, None except requests.exceptions.RequestException as e: return None, f"Error fetching Ecowitt data: {e}" except (json.JSONDecodeError, KeyError) as e: return None, f"Error processing Ecowitt data response: {e}"</pre>		<ul style="list-style-type: none"> • On success: return a tuple <code>(data, None)</code> • On exceptions: return <code>(None, error_message)</code> for clear upstream handling

```

@route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        if username == 'admin' and password == 'admin':
            session['logged_in'] = True
            return redirect(url_for('home'))
        else:
            error = 'Invalid credentials. Please try again.'
    return render_template('login.html', error=error)

```

- GET displays login page.
- POST validates hard-coded credentials
- On success: set session flag and redirect to home.
- On failure: redisplay form with error message

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login - Weather App</title>
    <link rel="stylesheet" href="/static/style.css">
    <style> ... </style>
</head>
<body>
    <div class="login-container">
        <div class="login-title">Weather App Login</div>
        <form class="login-form" method="post">
            {% if error %}
                <div class="error-message">{{ error }}</div>
            {% endif %}
            <label for="username">Username</label>
            <input type="text" id="username" name="username" required autofocus>
            <label for="password">Password</label>
            <input type="password" id="password" name="password" required>
            <button type="submit">Login</button>
        </form>
    </div>
</body>
</html>

```

- **Lines 1–6:**
 - Declares HTML5 <!DOCTYPE> and root <html lang="en"> for accessibility.
 - Meta charset ensures proper Unicode rendering.
 - External stylesheet link for shared styles.
- **Lines 7–62** (inline <style> block):
 - Overrides or supplements shared CSS for login page only.
 - Sets background, typography, and container card styles.
 - Splits into subsections:
 1. .login-container: centering and card styling.
 2. .login-title: typography and spacing.
 3. .login-form label & inputs: accessibility, form spacing.
 4. Buttons: full-width, hover transitions.

5. `.error-message`: visibility of server-side errors

- **Lines 64–78:**

- Semantic grouping: a single `<form>` with `method=POST`.
- Jinja2 conditional to display `error`.
- `<label> + <input>` pairs link via `for/id` for screen-reader support.
- `required` and `autofocus` improve UX and validation.