# A thermometer and sun Description automatically generated

# WEATHERLY

# Weather Notification App - Documentation

## 1. Overview

The Weather Notification App is designed to automatically analyze current weather conditions and provide recommendations based on the LiiNK weather guidelines. The app will alert users when it is safe or unsafe to go outside and suggest appropriate durations.

## 2. Features

- Weather Analysis: Fetch and analyze real-time weather data.

- Guideline Adherence: Evaluate conditions using LiiNK weather guidelines.

- Notifications: Provide user-friendly alerts and recommendations.

- AI Interface: Optional feature for conversational interactions.

## 3. Technology Stack

Frontend: HTML, CSS, JavaScript.

Backend: Python with Flask.

Database: SQLite/MySQL.

API: Tomorrow.io for weather data.

AI: TensorFlow or OpenAI API for conversational features.

## 4. Project Structure

- /templates: HTML templates.

- /static: CSS, JS, and images.

- /app.py: Main application logic.

- /models: AI or other computation modules.

## 5. Development Plan

Step 1: Setup and Skeleton - Flask project setup and API integration.

Step 2: LiiNK Guideline Integration - Logic to process weather data.

Step 3: Frontend UI - Responsive interface with dynamic displays.

**1. Make the Interface Responsive**

* **Mobile-First Design**: Ensure the layout adjusts gracefully for various screen sizes.
  + Use media queries to adapt styles for small, medium, and large screens.
  + Prioritize larger buttons and inputs for touch devices.
* **Flexbox/Grid Layouts**: Use CSS flexbox or grid for arranging elements.
  + Stack form fields vertically on smaller screens.
  + Arrange them side-by-side on larger screens for better use of space.
* **Viewport Meta Tag**: Ensure the webpage scales properly on mobile devices by using:

html

<meta name="viewport" content="width=device-width, initial-scale=1.0">

**2. Add Dynamic Displays**

* **Weather Cards**:
  + Display weather results in visually appealing cards or tiles.
  + Include temperature, condition, wind speed, AQI, and other relevant data with icons for clarity.
* **Dynamic Icons**:
  + Use weather-related icons (e.g., sunny, cloudy, rainy) that change based on the weather condition using libraries like [FontAwesome](https://fontawesome.com/) or Weather Icons.
* **Real-Time Updates**:
  + Show a loading animation while fetching data.
  + Dynamically update the display without reloading the page using JavaScript (AJAX).

**3. Enhance Usability**

* **Validation Feedback**:
  + Highlight required fields with a red border or tooltip if left blank.
  + Display error messages if the user inputs invalid city/state/country.
* **Auto-Complete Suggestions**:
  + Provide suggestions for city names or countries as the user types.
  + Use an API like Google Places for location autocomplete.
* **Unit Toggle**:
  + Add a toggle button to switch between Fahrenheit and Celsius for temperature.
* **Theme Options**:
  + Implement light/dark mode toggle for better accessibility and aesthetics.

**4. Add Visual Enhancements**

* **Backgrounds**:
  + Change the background dynamically to reflect the weather (e.g., sunny gradient for clear weather, darker tones for rain).
* **Animations**:
  + Add subtle animations to the cards or text as they appear on the screen.
  + Use CSS or libraries like [Animate.css](https://animate.style/) for smooth transitions.
* **Charts and Graphs**:
  + Include a temperature trend or AQI graph using libraries like [Chart.js](https://www.chartjs.org/).

**5. Accessibility Improvements**

* **Keyboard Navigation**:
  + Ensure users can navigate through the form using the Tab key.
* **Screen Reader Support**:
  + Use proper ARIA roles and labels for all interactive elements.
* **Color Contrast**:
  + Check and optimize the contrast ratio of text and background colors for readability.

**6. Test on Various Devices**

* **Cross-Browser Testing**:
  + Ensure compatibility with all major browsers like Chrome, Safari, Firefox, and Edge.
* **Device Testing**:
  + Use tools like [BrowserStack](https://www.browserstack.com/) to test the interface on various devices and screen sizes.

By implementing these enhancements, the UI will become more responsive, user-friendly, and visually appealing, providing a better experience for users across all devices.

4o

Step 4: AI Integration - Add chatbot for user interaction.

Step 5: Testing - Validate features with sample data.

## 6. LiiNK Guidelines Integration

The app will translate LiiNK weather guidelines into programmatic logic, such as temperature thresholds and exposure durations.

## 7. Challenges and Solutions

Challenge: Accurate weather data retrieval.  
Solution: Use Tomorrow.io API.

Challenge: Adhering to guidelines.  
Solution: Thorough guideline mapping and testing.

NOTES and FEEDBACK:

To test the app:

To send someone the link to your page for testing, follow these steps:

**1. Run Your Flask App Locally**

* Make sure your Flask app is running on your local machine. You can start your app by running this command in the terminal:

bash

Copy code

python app.py

* This will start the Flask development server, typically accessible on http://127.0.0.1:5000/ by default.

**2. Expose Your Local Flask App to the Internet**

Since the Flask server is by default only accessible on your local machine, you'll need to expose it to the internet for someone else to access. One popular way to do this is by using a tool called **Ngrok**. Here's how to use it:

* Download Ngrok from https://ngrok.com/download and follow the installation instructions.
* Once installed, open a terminal in the directory where app.py is located.
* Run the following command to create a tunnel to your local Flask app:

bash

Copy code

ngrok http 5000

This will give you a public URL like:

arduino

Copy code

http://xxxxxxxx.ngrok.io

(where xxxxxxxx will be a unique identifier for your session).

**3. Send the Ngrok URL**

* Once Ngrok generates the public URL, send that link (e.g., http://xxxxxxxx.ngrok.io) to the person you want to test the app.
* They can open that URL in their browser to access and test the app.

**4. Testing and Feedback**

* After testing, they can provide feedback or report any issues they encounter.
* You can make changes locally to your app and Ngrok will automatically reflect those changes in the public URL.

**5. Keep Ngrok Running**

* Keep the Ngrok terminal session open while the person is testing the app, as Ngrok only works while it's running. If you stop Ngrok, the public URL will no longer be valid.

**Optional: Deploy to a Web Hosting Service**

If you'd like to share the link permanently or make the app available to others without having to use Ngrok every time, you can deploy your app to a cloud service like **Heroku**, **Vercel**, or **PythonAnywhere**. These platforms will give you a permanent URL for your app.

Let me know if you need more details on any of these steps!

4o mini