Weather App Reporting & Documentation

Introduction

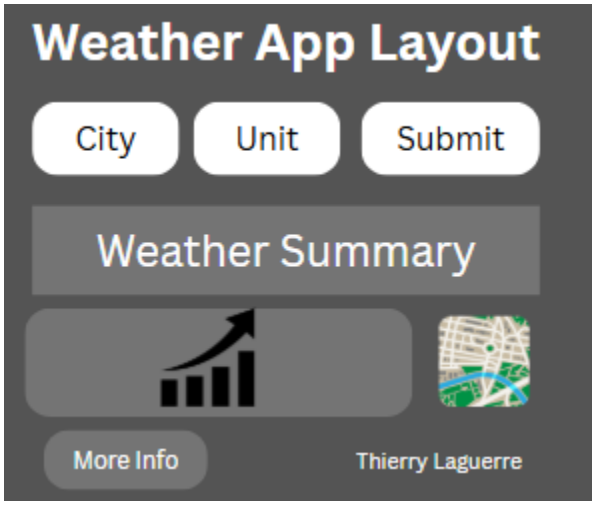
My application is an Asynchronous Weather Forecast Dashboard powered by the OpenWeather API. It is great for checking weather forecasts on a daily or weekly basis, as it allows you to view predictions up to four days ahead. This application is great for tourists, locals, drivers, outdoor workers, and adventurers. The app begins by prompting the user to enter a city to query. Next, the user has the option between Celsius (metric system) or Farenheight (imperial system), which are different standards for different municipalities. Moreover, for more ambitious users, I created a checkbox allowing them to view more detailed information such as wind speed, pressure, and humidity. Once submitted, the user can view tables, visualizations of charts and line graphs, and a map of their local area.

Usability Goals

Goal	How was it used?
Learnability	Intuitive Layout (nothing hidden), valid placeholders for input, and informative descriptions
Efficiency	One click to query weather Info, OpenWeather API information retrieval is fast, and visual cues such as loading message and charts.
Feedback	Instantaneous messages, success, warnings, loading, and errors.
Error Preventions	The user cannot progress until the city name field is no longer empty.

Minimalist	Consistent text sizes, fonts, and text color picker for customization

WireFrame



The initial wireframe focused on simply querying but did not tailor or personalize for users. Eventually, more options and features such as a color picker, sliders, and more visual aids were added to support user preferences.

API Integration

I used streamlit requests to fetch data and parse only the weather info. The first challenge I encountered was securely storing my API key, as it should not be shared, so I used a .env file. The next challenge was error handling for cities that do not exist, in which I return an error message and prompt the user to verify the city name input field.

Widget Explanation

Widget	Its purpose
st.text_input()	Text Input Field (Used for city name)
st.radio()	Selection of radio buttons (Used for

	units: Celsius or Fahrenheit
st.slider()	Slider widget (Use for the number of days to show)
st.checkbox()	Checkbox Widget (use for displaying more advanced tables and showing humidity, pressure, and wind speed bar chart)
st.button()	Button (used for submitting the city name as a parameter for fetching weather information
st.color_picker()	Color picker (Use for changing text color)
st.bar_chart()	Bar Chart Widget (Use for a bar chart of humidity, wind speed, and pressure)
st.map()	Area map (Used to specify location based on the city field)

HCI Principles

Principle	How was it used?
Visibility	Descriptive and clear language for sections such as Line Chart for temperature or Search Weather.
Feedback	Implemented using loading messages, warning messages, error messages, and quick response.
Consistency	Consistent styling, such as fonts and

	colors, across the board. Dataframes reused.
Flexibility	Allows users to filter advanced metrics such as wind speed, pressure, and more.
Error Prevention	Returns error message when city name is incorrect and warns user city field should not be empty.

Conclusion

Thus far, My Weather App adheres to the general guidelines of HCI principles and achieves its sole purpose of providing users with weather information based on city input. While my app works effectively, I feel a few changes can be made. I think that I could either store prior requests in localStorage or connect to some online database like Firebase. I would also like to maybe have a feature where users can turn on location, and the Weather App would be able to return forecasts without manual input.