¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ 6f6129df19eaffd0b97d91e37f16789f914a8fba ¡¡¡¡¡¡¡ HEAD ======= ¿¿¿¿¿¿¿ 6f6129df19eaffd0b97d91e37f16789f914a8fba

Deliverable for UltraWeather

Andrew Newbill, Joshua Chamberlain, Aaron Alden

Abstract

This computer application uses a weather API to collect weather data for the current time as well as the forecast for the week. The target audience are driving adults who need to know the weather conditions to get to work or class on time. The application will also have emergency notifications, such as tornado warnings. Currently, the project is able to call the API and get the current weather. It also has switchable themes that can be changed at any time.

1. Introduction

Our goal is to deliver a product that is user friendly and well designed so that users will have an easy way to check the weather from their desktop. One advantage of this is that web browsers are often resource hungry and a small WPF app like ours will not be. This is crucial for low end systems where multitasking with a web-browser open slows everything to a crawl. We are targeting power-users who would like to use as few resources as possible to track the weather. It is often the case with weather apps that they use far more resources than necessary.

Another issue we plan to tackle with this project is privacy. Weather applications by companies almost certainly store your information on a server. Since most people keep not only the city they live in but the ones that they travel to pinned in these apps, companies know exactly where you live and where you go based off only the information in this application. We plan to store information client-side to give the user more security in how their data is stored. Closed ecosystems intentionally do not reveal exactly how they handle user information and this is not acceptable to everyone. We intend to be entirely transparent about how we handle your data since the project will be open-sourced once it is delivered. Even if our implementation ends up being imperfect or less secure than we hoped, at least we won't be hiding it.

1.1. Background

1.1.1. Terms to Know. The reader should be familiar with the general terms used to refer to various weather patterns. We plan to use a theme system for this project. A theme in the context of our program will be a change of icons that represent the current weather for a given day. The user should also be familiar with the imperial system, as our program will provide the requested data in this form.

1.1.2. Personal Connection. We came up with the idea for this project because most weather apps are integrated into operating systems and as such they have limited functionality for users who may travel between large areas or are simply interested in following the weather on a large scale. One strong inspiration was the recently unpredictable weather in Tennessee. It would be good for users working on the desktop to be able to check the weather as they work with a simple and fast app that does not run as a system service.

1.2. Impacts

We hope that our project we impact the safety of its users in a positive way. A user could be working at their computer, blissfully unaware of a coming storm. We hope that our user would happen to check the app and decide not to leave their home on that day. This common situation could also impact the safety of the general public since more people on the road during a dangerous storm leads to a higher chance of accidents. It could happen that a social get-together that was planned gets canceled once one user sees coming poor weather and notifies their friends. We also hope that working on this project impacts us positively by furthering our experience at proposing projects and working as a group.

1.3. Challenges

The first major challenge will be getting our C# WPF application to interact with the OpenWeather API. Once we can call the API and receive data the next major challenge will be organizing this data into something that is easy for the user to view and interpret. After that we will have to overcome the task of implementing "themes" that allow the user to organize more than one location's data. We plan to overcome the challenges by first reading the XML data from the API into variables that can be placed at various locations in the window. Getting the data organized into something that is pleasant to view and organize will be by far the biggest challenge of this project. The free tier of OpenWeather limits us to a certain amount of calls, and does not allow us access to data beyond current weather and forecast.

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Set location	Application user	Med	1
2	Add Favorites	Application user	Low	2
3	Search Location's weather	Application user	Med	3

TABLE 1. ULTRAWEATHER 1ST DRAFT USE CASE TABLE

2. Scope

¡¡¡¡¡¡¡ HEAD We understand that in 2022, most weather apps have the same features we come to expect. The main way we can make our app different is by focusing more on the aesthetics and graphics of the app. We want to take a minimalist, tile approach to our app. We also want to implement our radar. Whether we will have it dynamic (a looping map) or static is still to be decided. For our stretch goals, we want to implement severe weather alerts and also "pre-alerts". For the pre-alerts, the idea is to give users a heads-up alert up to a week before severe weather occurs. For example, if meteorologists are predicting a nasty storm front next week, the user should be alerted that there is the possibility for severe weather. Sometimes, just a tornado warning pushed to your phone as the storm is happening is too late to seek shelter. ======= We understand that in 2022, most weather apps have the same features such as temperature and humidity. The main way we can make our app different is by focusing more on the aesthetics and graphics of the app. We plan to do this by implementing themes that allow the user to change the look and feel of their application at any time. We also want to implement a radar system that can help users identify weather systems that may be approaching as a stretch goal. ¿¿¿¿¿¿¿ 6f6129df19eaffd0b97d91e37f16789f914a8fba

2.1. Requirements

The requirements were acquired with all of us talking in class about the function and non-functional requirements as well as further communication through discord about both types of requirements.

2.1.1. Functional.

- The User should have the ability to pick what location they want to get the weather from – allows the User to get particular information for their use.
- The User will be able to see all weather information provided by OpenWeatherAPI.
- The User will be alerted when the national weather service has alerts for the searched location.
- The system will connect to the OpenWeatherAPI.
- The User will be able to put locations into folders.
- The User will be able to search for a location.

2.1.2. Non-Functional.

- Privacy - The user's search locations will be protected and not given to any third parties.
- Reliability - The app will work with a proper internet connection.
- Capacity - The app will have the same capacity as the storage on the computer i.e. The user will be able to have as many locations as their phone or computer can fit. Also the API calls are limited to 1,000,000 calls per month.

2.2. Use Cases

Use Case Number: 1
Use Case Name: Searching for current weather and forecast for a location
Description: Upon opening the application, the user will have the option to enter a city and state to get the weather for that location.

1) If the user wishes the application to find the weather for a location they can first enter the name of the city.
2) After entering the name of the city, the user can select which state the city is located in.
3) Once they have entered both the city and the state, the user can press the Go button to retrieve the information.

Termination Outcome: The application will display the current weather as well as the forecast for the week for that location.
Use Case Number: 2
Use Case Name: Entering a location that is not found

Description: A message box appears that lets the user know that the city was not found. No information will change in the window.

1) The user enters the name of the city for which they wish to get weather information.
2) The user selects the state of the city they wish to gather weather information for.

Termination Outcome: A message box appears that let's the user know the city was not found. None of the weather data displayed in the window changes.

Use Case Number: 3

Use Case Name: Changing the theme

Description: The user will have the option to change the theme of the application. This will not change the functionality.

1) The user wishes to change their theme. They select a theme from the list of available themes.

Termination Outcome: The icons change for the application to reflect the theme that the user has selected.

2.3. Interface Mockups

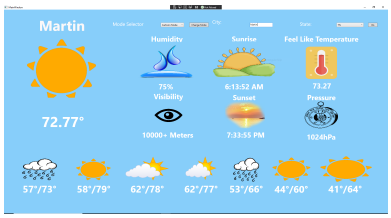


Figure 1. Illustration of Use Case 1. Here the user has entered a city and state and retrieved the weather information for said location.

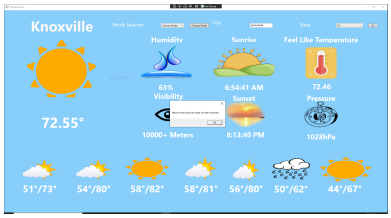


Figure 2. Illustration of Use Case 2. Here the has entered a city that could not be found in their chosen state.

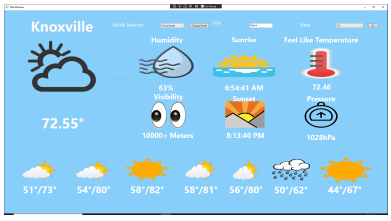


Figure 3. Illustration of Use Case 3. Here the user has changed the theme.

3. Project Timeline

Milestones: Feburary 5th: The requirements stage has been discussed and decided. March 1st: All designs are finished, shown through our use cases. March 29 - 31: Implementation starts with Andrew coding the system that connects to the API. April 5 - 12: Implementing the user interface designs shown through the use cases and implementing the location search function. April 14 - 21: Implementing other functions such as the tiling system and GPS optimal route searching functions If time is available: April 19 - 26: Work on radar stretch goal. April 28: Verfication and testing of all different themes and functionality of the program. Maintance will be taking place in the future as well as during development as problems arise.

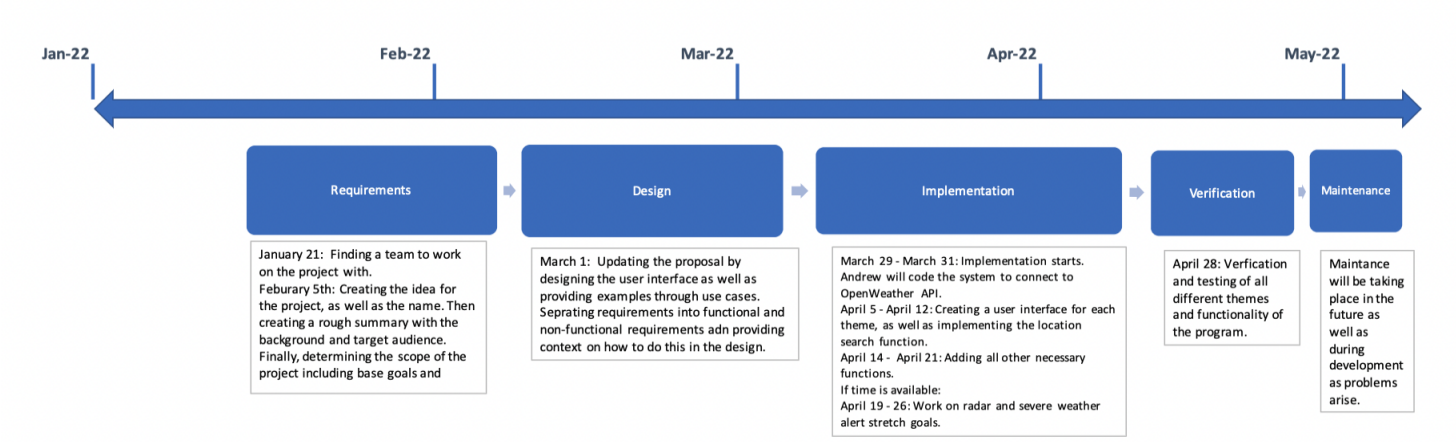
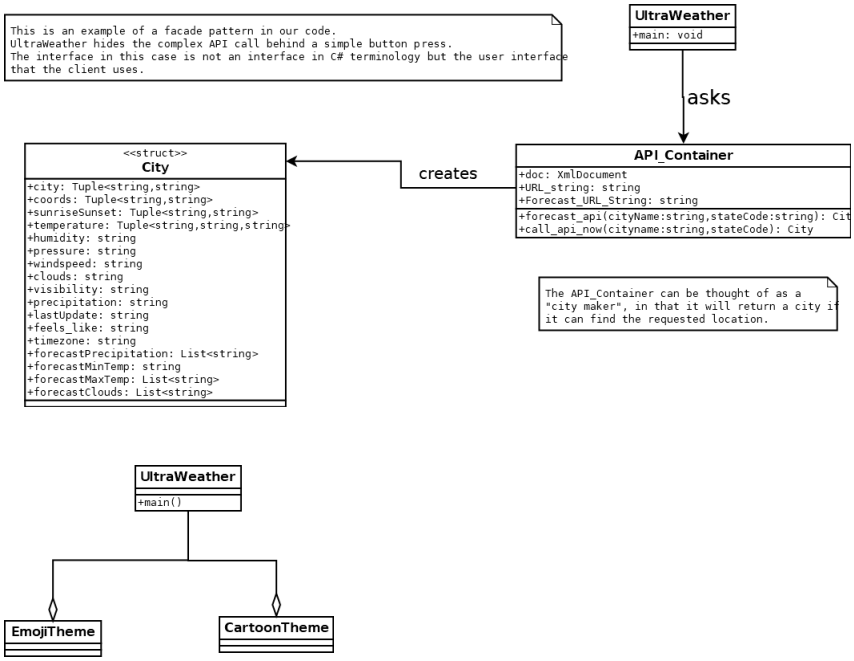


Figure 4. Full Project Timeline

3.1. UML Outline



4. Project Structure

The program is structured around the API Container. The Go button acts as a facade for the complex API call that happens once the user enters their city and selects their state. The program uses part of the visitor pattern for themes. All of the data is contained in a temporary City struct that will be destroyed once the data has been given to the relevant fields. This ensures a minimum usage of resources. The MainWindow class contains the functions for every button and list in our program window, which it uses mainly to access the API Container's call api functions.

4.1. Design Patterns Used

We used a facade pattern and a visitor pattern. The facade pattern comes with the user interface. The user interface is set up in a way that the user can provide information to the system, but they cannot access anything in the backend. The

visitor pattern is used when checking the mode/theme of the program. The program visits each data point (the two themes in this case) and does different actions based on which one it is. A full visitor pattern is not currently implemented, but we plan to do so before release.

5. Results

The UltraWeather collects data from OpenWeather API and displays it to the screen in a user friendly way. UltraWeather allows the person to type in a city and checks whether the city is valid. It provides an error MessageBox when it is not valid. The pictures will change when the weather is different, and the forecast will be as accurate as the API.

5.1. Future Work

Where are you going next with your project? For early deliverables, what are your next steps? (HINT: you will typically want to look back at your timeline and evaluate: did you meet your expected goals? Are you ahead of schedule? Did you decide to shift gears and implement a new feature?) By the end, what do you plan on doing with this project? Will you try to sell it? Set it on fire? Link to it on your resume and forget it exists?

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.