APRIL 2, 2022

Weatherly

PROJECT REPORT

Github: https://github.com/ctan24/Group-4-Project

Christopher Tan
Hector Onato
Joon Hyung Hahn

CPSC 2350 – Group 4

Table of Contents

Project Overview 1

SDLC 1

High-Level Features 2

APIs 2

Unit / Integration Tests 3

CI/CD Infrastructure 4

High-Level DFD 4

Takeaway & Challenges 4

Project Overview

Weatherly is a web application that suits all your needs regarding the weather. It lets the user select a city and display the cities' weather while showing daily, hourly, and current forecasts, accurate to the minute. This information will range from the high and low temperatures, wind speed, visibility, and the weather. Aside from forecasts, the application can also serve as a way to consume your weather reports, as the homepage displays weather related news of the entered city from the previous weeks. All of this information can be accessed in a single click, simply by entering the city name to the field provided.

The application can be accessed through the following link:
https://group4-weatherly-app.herokuapp.com/

SDLC

Model: Prototyping

Prototyping is an information-gathering technique useful in seeking user reactions, suggestions, innovations, and revision plans. The main goal of this SDLC is to create a working model that has all the features to allow the users to interact with the system. The specific prototype model we chose is a selected features prototype. This model aims to implement the features one at a time, and the following features will be built on top of the created prototype. Successful prototypes created with this model in mind are not just a mock-up that is used to test its features, but will be incorporated into the final system, reducing the workload of interfacing.

We chose to use the prototyping model because we valued having a clear representation of our project before moving on to the next step in development. By using the selected features prototype, we can prioritize features create a prototype that ensure we have our most important features working. At the start of the project, there were other features that we wanted to implement, such as weather maps. Since our SDLC model allowed us to create a working prototype for our most important features first, we managed to determine that we won’t have time to implement the weather maps.

A problem we faced with our SDLC model is the time consumed. While it was beneficial in that we were able to redetermine the requirements for our project, waiting for the prototype meant that we had lest time to implement other things like tests. Overall, the prototyping model helped us greatly because this is the first time some of our members are implementing APIs with an unfamiliar language in the form of Node.js. We were able learn more and build a solid foundation prior to creating the first prototype, allowing us to redefine the direction and requirements for our project after we have a better understanding on the technology stack we were using.

High-Level Features

- Allows users to look up the forecasts of a specified city by entering the location into the text field provided
- Displays minute, hourly, and daily forecasts when a valid city is entered
- Displays weather related news from the day before and some from previous weeks

APIs

One Call API:
Makes one API call to get current forecasts and historical weather data in a JSON format

- Minute forecasts for current hour, which includes temperature, weather, wind speed, and visibility
- Hourly forecast for the next 3 hours, which includes temperature and weather
- Daily forecast for the three upcoming days, which includes temperature and weather

News API:
Locate articles and breaking headlines from news sources and blogs across the web

- Daily news reports, displays news headlines from the previous day
- Weekly news reports, displays news headlines from one and two weeks prior

Geocoding API:
Converts addresses represented by a string into geographic coordinates

Heroku:
Allows the GitHub repository to connect to Heroku and deploy the application

Unit / Integration Tests

7 Unit tests:

Jest was used for unit tests, making sure that functions used in this project is working as intended. Some functions had to be isolated since the functions being tested had dependencies on other functions that made tests failed. Eliminating dependencies while testing helped reducing redundant codes and errors.

We used mock object for unit tests. Making a mock variable with a value of our intention, we could have tested our functions are working properly or not.

Creating invalid inputs, empty arrays and throwing error messages were checked in the unit tests.

4 Integration tests

Jest was used for integration tests. Some functions and parts are combined and tested as in one unit/ group for integration test purpose. We wanted to test whether the data are well sent and received from one to another, so focused on pipeline tests. We treated checking port is working properly as one integration test as it serves as one flow. The other tests are asynchronous functions so we implemented “async” and “await” callbacks in the integration tests.

CI/CD Infrastructure

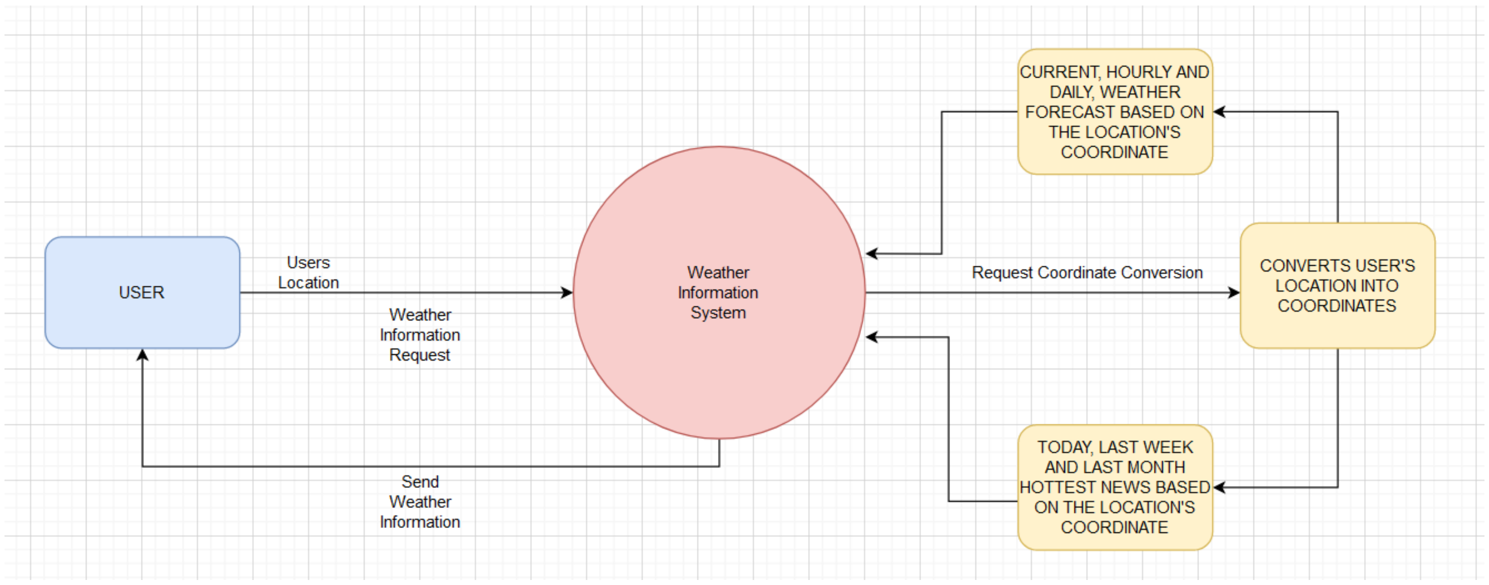
Our CI/CD pipeline consists of using GitHub Actions that runs a workflow file whenever there is a push or pull request made on the main branch. We used a template of a Node.js workflow from the GitHub Actions Library and built upon it to incorporate continuous deployment.

We created a workflow that creates a clean installation of node dependencies which will then build the source code and run the unit and integration tests defined above, across different versions of node (12.x, 14.x, 16.x).

Passing all these tests are a prerequisite for the workflow before running its next job, which is deploying our website to Heroku. Upon passing all the prior jobs, the workflow will establish a connection to the Heroku API and push the master branch into Heroku. Heroku’s built in feature to connect with GitHub’s repository will then allow us to deploy the application accessible through the following link:

https://group4-weatherly-app.herokuapp.com/

High-Level DFD


USER
Users Location
Weather Information Request
Weather Information System
Send Weather Information
Request Coordinate Conversion
CONVERTS USER'S LOCATION INTO COORDINATES
CURRENT, HOURLY AND DAILY, WEATHER FORECAST BASED ON THE LOCATION'S COORDINATE
TODAY, LAST WEEK AND LAST MONTH HOTTEST NEWS BASED ON THE LOCATION'S COORDINATE

Takeaway & Challenges

The development of Weatherly has taken around 2.5 months, and it was an undertaking not free from challenges. It was our first time planning out a project of this scale implementing a lot of the features, thus there are many lessons to be learnt and aspects of the project that can be approved upon.

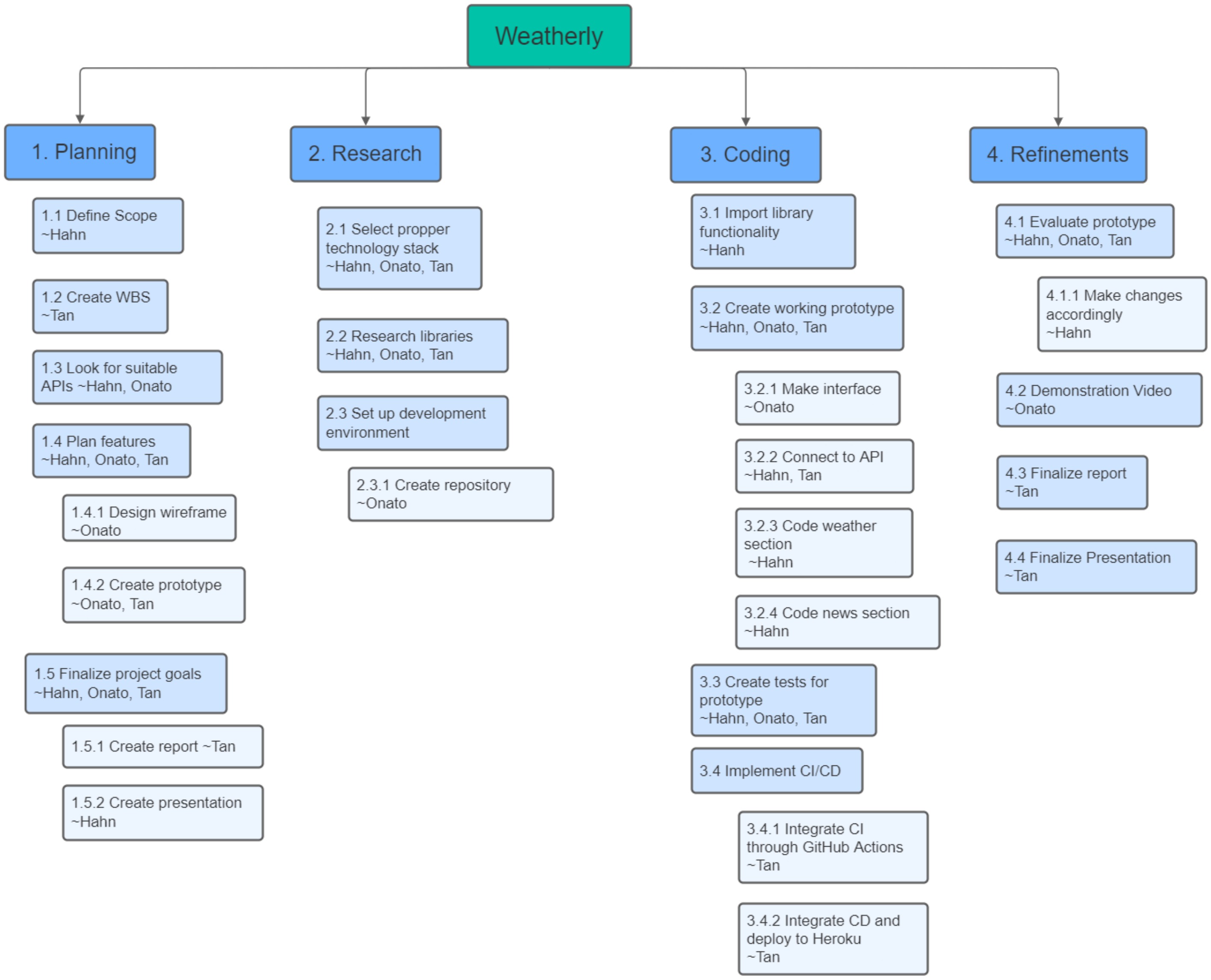
Since the scope of our project was rather small, the prototyping model we chose was working as expected. For a project with a larger scale, it would be more appropriate to use agile to accommodate for changes more flexibly. The prototyping model we chose was very similar to how agile worked, and it was part of the reason as to why it worked so well. We can see how when using agile, managing sprints would allow the project to go more smoothly and perhaps even address the time constraints we had the first prototype taking so much of our time.

Another challenge we faced was the overpromise of features. Despite knowing that we were not well versed in Node and that we were down a member, we planned too many features which didn’t end up being implemented. Luckily, our SDLC model made us identify and prioritize a particular feature first, meaning the time spent on the dropped features weren’t much.

We had a bit of trouble implementing the CI/CD pipeline because of our unfamiliarity with the technology stack we chose. Since 2 of the members in our group were unfamiliar with the language, we had to wait for the first prototype to be finished before we can learn from said code and create the unit / integration tests. This held up our CI/CD pipeline, missing the implementation date for the check-in on March 19. Thorough research should have been done during the planning phase so that the unit / integration tests could be implemented sooner.

The current problem that remains unsolved revolves around the new API. The API we chose was free, but it only allows up to 100 calls a day. While this was okay for individual tests and uses, it becomes a problem when we have tests that needs to make calls to the API whenever it runs and when multiple people have access to the application. This means if we were to push too many times in a day, the application can stop working altogether because the news API can no longer be accessed.

A lot can be improved on our project, but the key takeaway would be to clearly define the project scope and choose an appropriate SDLC model that accommodates the situation. The work done by each member is represented by the WBS below:


Weatherly
1. Planning
1.1 Define Scope ~Hahn
1.2 Create WBS ~Tan
1.3 Look for suitable APIs ~Hahn, Onato
1.4 Plan features ~Hahn, Onato, Tan
1.4.1 Design wireframe ~Onato
1.4.2 Create prototype ~Onato, Tan
1.5 Finalize project goals ~Hahn, Onato, Tan
1.5.1 Create report ~Tan
1.5.2 Create presentation ~Hahn
2. Research
2.1 Select propper technology stack ~Hahn, Onato, Tan
2.2 Research libraries ~Hahn, Onato, Tan
2.3 Set up development environment
2.3.1 Create repository ~Onato
3. Coding
3.1 Import library functionality ~Hanh
3.2 Create working prototype ~Hahn, Onato, Tan
3.2.1 Make interface ~Onato
3.2.2 Connect to API ~Hahn, Tan
3.2.3 Code weather section ~Hahn
3.2.4 Code news section ~Hahn
3.3 Create tests for prototype ~Hahn, Onato, Tan
3.4 Implement CI/CD
3.4.1 Integrate CI through GitHub Actions ~Tan
3.4.2 Integrate CD and deploy to Heroku ~Tan
4. Refinements
4.1 Evaluate prototype ~Hahn, Onato, Tan
4.1.1 Make changes accordingly ~Hahn
4.2 Demonstration Video ~Onato
4.3 Finalize report ~Tan
4.4 Finalize Presentation ~Tan