***SOFTWARE PROJECT FINAL REPORT***

Meir Manevich, Munkaran Singh

5 December 2024

Table of Contents

[1. Introduction 4](#_Toc184232200)

[1.1. Purpose and Scope 4](#_Toc184232201)

[1.2. Product Overview (including capabilities, scenarios for using the product, etc.) 4](#_Toc184232202)

[1.3. Terms, Acronyms, and Abbreviations 4](#_Toc184232203)

[2. Project Management Plan 4](#_Toc184232204)

[2.1. Project Organization 4](#_Toc184232205)

[2.2. Lifecycle Model Used 5](#_Toc184232206)

[2.3. Risk Analysis 5](#_Toc184232207)

[2.4. Hardware and Software Resource Requirements 5](#_Toc184232208)

[2.5. Deliverables and schedule 6](#_Toc184232209)

[3. Requirement Specifications 6](#_Toc184232210)

[3.1. Stakeholders for the system 6](#_Toc184232211)

[3.2. Use cases 7](#_Toc184232212)

[3.2.1. Graphic use case model 7](#_Toc184232213)

[3.2.2. Textual Description for each use case 8](#_Toc184232214)

[3.3. Non-functional requirements 9](#_Toc184232215)

[4. Architecture 10](#_Toc184232216)

[4.1. Architectural style(s) used 10](#_Toc184232217)

[4.2. Architectural model (includes components and their interactions) 10](#_Toc184232218)

[4.3. Technology, software, and hardware used 11](#_Toc184232219)

[4.4. Rationale for your architectural style and model 11](#_Toc184232220)

[5. Design 11](#_Toc184232221)

[5.1. User Interface design 11](#_Toc184232222)

[5.2. Components design (static and dynamic models of each component) 12](#_Toc184232223)

[5.3. Database design 12](#_Toc184232224)

[5.4. Rationale for your detailed design models 12](#_Toc184232225)

[6. Test Management 12](#_Toc184232226)

[6.1. A complete list of system test cases 12](#_Toc184232227)

[6.2. Traceability of test cases to use cases 14](#_Toc184232228)

[6.3. Techniques used for test case generation 14](#_Toc184232229)

[6.4. Test results and assessments (how good are your test cases? How good is your software?) 14](#_Toc184232230)

[6.5. Defects reports 15](#_Toc184232231)

[7. Conclusions 15](#_Toc184232232)

[7.1. Outcomes of the project (are all goals achieved?) 15](#_Toc184232233)

[7.2. Lessons learned 15](#_Toc184232234)

[7.3. Future development 15](#_Toc184232235)

[References 15](#_Toc184232236)

List of Figures

[**Figure 1 7**](#_Toc184241120)

[**Figure 2 7**](#_Toc184241121)

[**Figure 3 7**](#_Toc184241122)

[**Figure 4 7**](#_Toc184241123)

[**Figure 5 10**](#_Toc184241124)

List of Tables

[**Table 1 12**](#_Toc184210606)

[**Table 2 12**](#_Toc184210607)

[**Table 3 13**](#_Toc184210608)

[**Table 4 13**](#_Toc184210609)

[**Table 5 13**](#_Toc184210610)

[**Table 6 14**](#_Toc184210611)

# 1. Introduction

## 1.1. Purpose and Scope

* Our purpose is to create a weather application that displays city-specific weather data to the user. We aim to replace the user's current go-to weather source to provide a more intuitive and clear application.
* Our application is built to appeal to any and all individuals seeking to gather current and up-to-date weather information.

## 1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

* The application contains the following features:
  + Weather search via city name or zip code.
  + Current weather data including temperature, precipitation, and more.
  + Future weather forecast including hourly and daily forecasts.
  + The option to save previously searched locations to quickly view them.
  + The ability to see past weather.
* The application is meant to be used in any scenarios for all people. Whether it be for personal or professional use, the application can provide useful weather metrics to the user.

## 1.3. Terms, Acronyms, and Abbreviations

* API: Application programming interface; this enables communication between the application and weather sources. The API used is OpenMeteo.
* GUI: Graphical user interface.
* WMO: World Meteorological Organization.
* Gradle: An automation tool for building and running software projects. Our app comes in the form of a Gradle project to simplify the build and running sequence.

# 2. Project Management Plan

## 2.1. Project Organization

* The project development was split in two parts:
  + Back-End Development: This included all of the programming related to the API and communicating with it. This also involved taking in raw data and converting it to the proper data types required by the front-end to display and present it. The back-end development was done by M. Manevich.
  + Front-End Development: This included the execution of the data and the overall design of the app. Data received from the back end was used to update fields within the app in order for it to be displayed. This development side also included creating the images used in the application. The front-end development was done by M. Singh.

## 2.2. Lifecycle Model Used

* The waterfall model was used in the development of our application.
* We started by outlining our requirements, then we designed the software and system before testing and finalizing.

## 2.3. Risk Analysis

* As with any application, there are a number of risks involved in development. Below, we’ve listed a few key risks and how we tried to avoid them:
  + Risk: Weather data may be inaccurate, thus destroying app credibility.
    - We avoided this by using a well-established weather API that is trusted and used by many. This way, there is a lower chance of failure.
  + Risk: The development team may not be equipped with the necessary skills to develop an extensive app in such a small time period.
    - We avoided this by working smarter. Since our team of two only had roughly three months to complete this application, we split the work, with each member using their skills and interests to efficiently finish the project. The choice to use an established API also came as a result of this.
  + Risk: The application may not be cross-compatible among platforms.
    - To avoid this, our team used two of the most popular systems in personal computing: Windows and MacOS. This way, we could ensure that no issues came due to the software on which the app is running.

## 2.4. Hardware and Software Resource Requirements

* As mentioned previously, our application can be used across both Windows and MacOS. The java files come as a Gradle project.
* Gradle projects can be run across most IDE’s including Microsoft’s Visual Studio Code (VScode), and Apache’s Netbeans.

## 2.5. Deliverables and schedule

* Our project began in late September/early October of 2024, with a final due date of December 3, 2024.
* Preliminary planning documents such as our project plan, software specification, and design specification were completed in October preceding the initial development of our software.
* The initial draft was completed in early November and presented to our peers.
* Following the initial draft, a test plan was completed in late November.
* Our application was completed in late November and testing immediately began, stretching into early December.
* Following the testing, user’s and developer’s guides were created to complete.
* The final project presentation will happen on December 5th, 2024.

# 3. Requirement Specifications

## 3.1. Stakeholders for the system

* Everyone can be a stakeholder for our application. Examples of stakeholders and their roles include:
  + Users: The people who use the app for weather information.
  + Developers: The people who create the app and define its features.
  + Weather Providers (API): The controller of information available to the developers to then provide to the users.

## 3.2. Use cases

### 3.2.1. Graphic use case models

A blue circle with black text

Description automatically generated

Figure 1

A blue oval with black text

Description automatically generated

Figure 2

A blue oval with black text

Description automatically generated

Figure 3

A diagram of a search engine

Description automatically generated

Figure 4

### 3.2.2. Textual Description for each use case

|  |  |
| --- | --- |
| **M&M Weather: Enter City/Zip Code into Search Box and click Search Button** | |
| Actors | User, OpenMeteo Geocoding and Weather APIs |
| Description | A user can type a city name or zip code into a search bar and click to search button to retrieve weather data. |
| Data | OpenMeteo Locations Coordinates and Weather Databases. |
| Stimulus | User clicking the search button. |
| Response | Sending user’s search term to OpenMeteo APIs. |
| Comments | The program frontend should notify the user if their search term did not return any results. |
| **M&M Weather: View Current, Hourly, and Daily Weather** | |
| Actors | User, App Frontend |
| Description | A user can view the current, hourly and daily weather for the location they searched for, up to 6 hours for the hourly section and 7 days for the daily sections. |
| Data | JSON file with all weather information returned by the Weather API, parsed into text and displayed in textboxes. |
| Stimulus | App backend populates weather variables with textual representations of the data from the JSON file. |
| Response | App frontend populates all of its textboxes with the appropriates variables from the backend. |
| Comments | Past Weather feature could not be completed due to time constraints. |

|  |  |
| --- | --- |
| **M&M Weather: Save Search** | |
| Actors | User, App Backend |
| Description | A user can choose to save their current search term to a list for quick access in the future. |
| Data | List of saved strings. |
| Stimulus | User clicking the save button. |
| Response | Current search term is saved to the Saved List. |
| Comments | If the user tries to save an incorrect search term, the program notifies them and prompts them to try again. |
| **M&M Weather: Click on Saved Search from Saved List** | |
| Actors | User, OpenMeteo Geocoding and Weather APIs |
| Description | A user can double click a saved search term to retrieve weather data. |
| Data | OpenMeteo Locations Coordinates and Weather Databases. |
| Stimulus | User clicking the saved search term. |
| Response | Sending user’s search term to OpenMeteo APIs. |
| Comments |  |

## 3.3. Non-functional requirements

* Application must run using the Java Virtual Machine.
* The speed of the application is dependent on the internet speed of the user and response time of the APIs.
* User cannot save more than 10 terms to the Saved List.

# 4. Architecture

## 4.1. Architectural style(s) used

* The pipe and filter architecture was used in our app.

## 4.2. Architectural model (includes components and their interactions)

* A search box takes a string from the user as input, which is sent as the input parameter of the search() function.
* The search() function returns a JSON object, which is sent to the weatherParse() function to be parsed and split up into variables to be displayed by the GUI.
* An error message will be displayed beneath the search box if the search term is invalid.

A close-up of a diagram

Description automatically generated

Figure 5

## 

## 4.3. Technology, software, and hardware used

* Two API’s (one for location and one for weather) are used for the generation of necessary information.
* Data parsing is used to gather the information from the API’s, it is then converted and made accessible to the user.

## 4.4. Rationale for your architectural style and model

* This structure worked best for us since it involves data transformations that process inputs in order to produce outputs. In a very simplified form, we are processing a user’s input and producing output.

# 5. Design

## 5.1. User Interface design

* The UI contains the following:
  + A full screen background image corresponding to the current weather conditions.
  + A search/save panel on the far right of the screen, which contains:
    - A search bar to enter location (city name/zip code).
    - A search button to submit the location entered.
    - A list of saved locations.
    - A button to save the current location into the saved locations.
  + The remaining screen real estate is split into thirds vertically.
  + The leftmost third contains the following current weather conditions:
    - Relative Humidity.
    - Apparent (feels like) temperature.
    - Precipitation.
    - Wind speed.
    - Wind direction.
  + The central third contains the following:
    - Location.
    - Current temperature.
    - Current conditions.
    - 6-hour temperature forecast.
    - 6-hour precipitation forecast.
  + The rightmost third contains the following 7-day weather forecast elements, separated by day:
    - Maximum Temperature.
    - Minimum Temperature.
    - Sunrise.
    - Sunset.
    - Precipitation.

## 5.2. Components design (static and dynamic models of each component)

* Almost every text field is dynamic, controlled by the program to adjust according to the data provided for each searched location.

## 5.3. Database design

* We did not use a database for this program, although we rely on OpenMeteo’s extensive databases of location coordinates and weather data to run our program.

## 5.4. Rationale for your detailed design models

* Our design model draws the user in to the center of the screen. A large open space provides for the user to see the main element of the background image. Here, we also introduce the main, current conditions. This allows the user to view vital information at a glance.

# 6. Test Management

## 6.1. A complete list of system test cases

Table 1

|  |  |
| --- | --- |
| ID | 1 |
| Test Input | “Cleveland” |
| Expected Output | The program should display accurate, current weather information for the city of Cleveland. |
| Description | This is used to test the search function for the weather application with a city name as input. |

Table 2

|  |  |
| --- | --- |
| ID | 2 |
| Test Input | “44114” |
| Expected Output | The program should display accurate, current weather information for the city of Cleveland. |
| Description | This is used to test the search function for the weather application with a zip code as input. |

Table 3

|  |  |
| --- | --- |
| ID | 3 |
| Test Input | “Cl3veland” and “402I9” |
| Expected Output | The program should display an error for an invalid location entry. |
| Description | This is used to test the search function for the weather application with an invalid location as input. |

Table 4

|  |  |
| --- | --- |
| ID | 4 |
| Test Input | Save city |
| Expected Output | The program should save the current location to the saved list. |
| Description | This is used to test the save function for the weather application. |

Table 5

|  |  |
| --- | --- |
| ID | 5 |
| Test Input | Load city |
| Expected Output | The program should display accurate, current weather information for the saved city. |
| Description | This is used to test the save function for the weather application. |

Table 6

|  |  |
| --- | --- |
| ID | 6 |
| Test Input | Background Image |
| Expected Output | The background image should successfully update according to the weather and time at the location. |
| Description | This is used to test the image selection methods for the weather application. |

## 6.2. Traceability of test cases to use cases

* The test-cases involve the elements of our use cases. We need to be able to display accurate information, all of our test cases involve this.

## 6.3. Techniques used for test case generation

* To develop the test cases, we simply looked at every feature of our app. If our app does something, it must be tested.

## 6.4. Test results and assessments (how good are your test cases? How good is your software?)

* Test 1: City Search.
  + Our program runs flawlessly with searching cities. If a valid city is entered, it will display weather information for that city, every time.
* Test 2: Zip code Search.
  + Our program also runs flawlessly with searching zip codes. If a valid code is entered, it will display weather information for that city, every time.
* Test 3: Invalid Search.
  + In the event an invalid city or zip code is entered, the app will display a notification to the user, prompting them to reenter a valid location.
* Test 4: Location save.
  + The app will save a location once you search it with no problems.
* Test 5: Location load:
  + The app will load a saved location once you double click it from the list.
* Test 6: Background image.
  + The background updates dynamically to each and every city, corresponding with their weather conditions. Currently, eight images are available.
* Our test cases were very good. They helped us test every feature and function of our app. It is important to note that the actual test inputs went beyond just what we listed. Random locations all over the world were searched and the app provided great results.

## 6.5. Defects reports

* Currently, the application has no defects. Every described feature works and works well.

# 7. Conclusions

## 7.1. Outcomes of the project (are all goals achieved?)

* Almost all of our goals were achieved. We successfully created a weather application that can display accurate and up-to-date weather information. One goal, however, was not achieved: we were unable to implement past weather to our application. There were no issues gathering or displaying this information, we just did not have enough time to implement those features. Instead, we focused on ensuring our app runs well and reliably.

## 7.2. Lessons learned

* One of the biggest lessons came with learning the importance of organization. By staying organized in the traditional sense, where we keep everything neat and tidy, we efficiently built our application. Moreover, we were organized in terms of being clear and having good communication. There was rarely a time where we were unsure of what the other person would think. We had a clear route and if there was any question in what we should do, we immediately contacted each other.
* Another lesson we learned was the importance of setting aside more time than we initially thought for development. A small thing to work on could quickly balloon into a major issue and then three hours later you are still working on it. Ensuring we began development right away enabled us to receive such curveballs and still complete the project in a timely manner.

## 7.3. Future development

* In the future, we would like to add the past weather data. This is a feature that is available to us and would help set our app apart from other big name weather applications.

# References

Software Engineering by Ian Sommerville