**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | Cross-platform Development |
| **Assessment Title:** | Analysis of Information Systems |
| **Lecturer Name:** | David Gonzalez |
| **Student Full Name:** | Alejo Santos, Telmuun Dunia, Luis Ramirez |
| **Student Number:** | 2023197, 2023306, 2023169 |
| **Assessment Due Date:** | TBC |
| **Date of Submission:** |  |

**Declaration**

|  |
| --- |
| By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution. |

Contents

[Introduction. 1](#_Toc198567728)

[Weather app Alutech. 2](#_Toc198567729)

[Requirements. 2](#_Toc198567730)

[Installation steps. 3](#_Toc198567731)

[Design Decisions. 4](#_Toc198567732)

[Application usage. 5](#_Toc198567733)

[APIs used. 5](#_Toc198567734)

[Challenges Encountered and Solutions Implemented. 5](#_Toc198567735)

[Reflections on the Development Process. 7](#_Toc198567736)

[Potential future improvements. 7](#_Toc198567737)

[INTERACTIVE APPLICATION DEVELOPMENT. 8](#_Toc198567738)

[Part 1: Software Development Life Cycle (SDLC) – Weather App in React. 9](#_Toc198567739)

[Phase 1: Requirements Analysis. 9](#_Toc198567740)

[Phase 2: System Design. 10](#_Toc198567741)

[Phase 3: Implementation. 10](#_Toc198567742)

[Phase 4: Testing. 11](#_Toc198567743)

[Phase 5: Deployment. 12](#_Toc198567744)

[Phase 6: Maintenance. 12](#_Toc198567745)

[Part 2: Software Development Life Cycle. 13](#_Toc198567746)

# Introduction.

This project is a modern, responsive, and user-friendly weather application developed using the React framework. Designed with simplicity and efficiency in mind, the app offers a clean and intuitive interface for users to check real-time weather conditions in any city around the world.

The core functionality includes fetching live weather data from a public API, displaying key information such as temperature, weather conditions, humidity, wind speed, and a 5-day forecast. Users can search for any location, with results updating dynamically based on their input.

To enhance the overall user experience, the app also includes features such as location-based weather detection, animated weather icons, and customizable theme options (e.g., light/dark mode). Altogether, this project showcases a balance between functionality and design, demonstrating how modern front-end technologies like React can be used to build practical and visually engaging applications.

# Weather app Alutech.

This weather app was designed to be a simple and user-friendly application.

## Requirements.

The basic requirements to run this project are the same as every React project, for example:

Operating System:

* Windows
* macOS
* Linux

Node.js

* v14 or higher

**Dependencies.**

These are typical dependencies required to run the app

* React – Front-end framework (react, react-dom, react-scripts)
* Weather API – Example: OpenWeatherMap API key (you’ll need to register and get your own key)
* dotenv (optional) – For managing API keys securely

Once you have completed all these requirements, you are ready to download the application from GitHub or follow these steps.

## Installation steps.

1. Clone repository:

<https://github.com/TelmuunDn/weather-app-ca2-aoe.git>

1. Install dependencies:

*npm install*

1. Install Expo CLI(if not installed)

*npm install -g expo-cli*

1. Install required Expo packages

*npx expo install expo-location expo-font expo-linear-gradient*

1. Install additional packages

*npm install @react-native-async-storage/async-storage*

*npm install lodash*

*npm install react-native-vector-icons*

1. Start the Expo app

*npx expo start –tunnel*

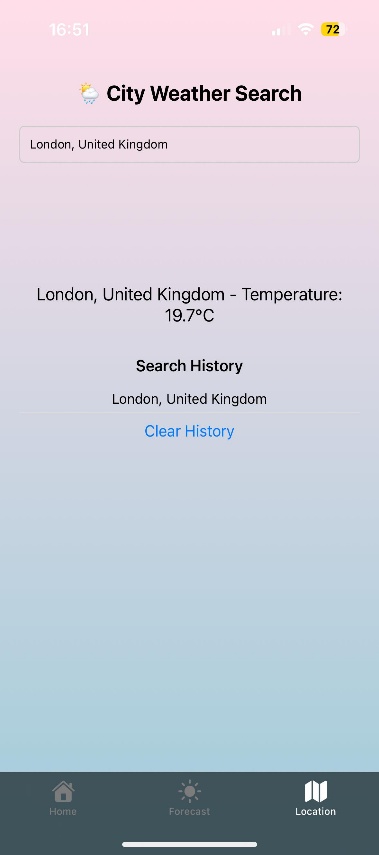
A screenshot of a computer

AI-generated content may be incorrect.Open the QR code in the Expo Go app on your phone or run it on an emulator.

## Design Decisions.

Here is a brief explanation of some features that you will find inside this application:

1. The application is built in tabs: the Home, Forecast, and Location tabs.



The Home page has a simple design showing the temperature, emoji, location with hour and date, and three features like rain, humidity and wind. We kept the user interface with this easy way to navigate through the application because one of the main purposes is to keep it simple and aesthetic.

The Forecast page shows only 7 days to avoid saturating the page with more information, and the style makes it easy to read and understand the data.

The Location page shows the find bar and the search history, as well as the main page, we kept this in a simple style to leave the users to find a location that they are looking for instead of showing places that are not of interest.

## Application usage.

1. Every tab contains different information.
2. On the Home tab, you will see the current temperature in your location, and the toggle button to change between Fahrenheit and Celsius temperatures.
3. Forecast tab, you will see the next five days' forecast with the date, max and low temperature, and the probability of rain.
4. On the Location tab, you will find a search bar to check the temperature in the place that you desire.

## APIs used.

1. IPAPI (for location by IP).

* Fetches the current city/country by the user's public IP
* Fallback to this method when GPS is unavailable

1. Meteomatics API.

* Primary source for weather data
* Requires API key (base64 encoded username: password)
* Set your credentials directly in the Authorization header in the request

Authorization: "Basic " + btoa("your\_username:your\_password")

## Challenges Encountered and Solutions Implemented.

***Integrating a weather API.***

Solution implemented:

* Researched and selected a suitable weather API (e.g., OpenWeatherMap, WeatherAPI).
* Registered for an API key and handled authentication.

***Handling user input (City Search).***

Solution Implemented:

* Created a search bar with controlled components (useState) for city input.
* On form submission, the app triggered a new fetch for the specified city’s weather data.

***Displaying weather data nicely.***

Solution Implemented:

* Parsed JSON responses and formatted dates, temperatures, and icons.
* Used conditionals to show different icons or backgrounds based on weather type

## Reflections on the Development Process.

Learning how React works and applying it to a real-world weather app project was a great learning experience. It was rewarding to see how everything we studied (like components, hooks, and API calls) came together to build something useful.

Using GitHub for version control made it easier to keep track of changes and work as a team. We also had regular discussions about how to improve the app and fix bugs, which helped us stay organized and move forward smoothly.

We gained hands-on experience with building a full React app, using external APIs (like a weather API), and managing state with hooks like useState and useEffect. It also helped us better understand error handling, async data fetching, and how to make an interface that updates based on real-time data.

One thing we could have done differently was planning the app’s structure more clearly from the beginning. We didn’t have a full idea of what the final product should look like, so we had to go back and change things a few times. With better early planning, we could have avoided some of the refactoring and feature changes later on.

## Potential future improvements.

* ***Add dynamic backgrounds or icons.***

Change the background or weather icons based on conditions (e.g., sunny, rainy, snowy).

Why? Makes the app more visually engaging and informative.

* ***Improve mobile responsiveness.***

Make sure the layout looks clean and functions smoothly on all screen sizes.

* ***Add timezone and local time display.***

Show the local time of the searched city alongside weather data.

* ***Support for multiple languages.***

Add a language selection feature using i18n libraries (like react-i18next).

* ***Weather alerts or notifications.***

Show warnings for extreme weather like storms, heat waves, etc.

* ***Save favourite locations.***

Let users bookmark cities they frequently check.

# INTERACTIVE APPLICATION DEVELOPMENT.

## Part 1: Software Development Life Cycle (SDLC) – Weather App in React.

The Waterfall methodology provided a clear and structured workflow for developing our weather application. Because the core features—such as city search, API integration, real-time weather display, and a responsive UI—were well-defined from the beginning, the linear nature of Waterfall helped us execute the project phase by phase with minimal revisions. Each stage (Requirements, Design, Implementation, Testing, Deployment, and Maintenance) was completed sequentially, allowing us to focus on delivering a clean and functional application on time.

This structure made it easy to split tasks, track milestones, and maintain documentation throughout development.

## Phase 1: Requirements Analysis.

In this phase, we defined the scope and expected functionality of our weather application.

**Core Functional Requirements:**

* **City Search:** Users should be able to search for any city to view its current weather conditions.
* **Real-Time Weather Data:** Fetch current temperature, condition (e.g., sunny, rainy), humidity, wind speed, and weather icons using a public weather API (e.g., OpenWeatherMap).
* **Error Handling:** Show friendly messages for failed searches, API issues, or empty inputs.
* **Responsive UI:** The layout should adjust for mobile and desktop screens.
* **Loading & State Management:** Indicate loading status and handle data cleanly using React hooks.

## Phase 2: System Design.

In this phase, we planned the architecture and identified the tools and components we would use.

**Architecture and Tools:**

* **Frontend Framework:** React with functional components and hooks.
* **Data Fetching:** axios for calling the weather API.
* **State Management:** useState, useEffect for component-level state.
* **Error and Loading States:** Simple conditional rendering.
* **Styling:** Custom CSS for responsiveness and theming.
* **Version Control:** Git and GitHub for code management.

**Component Design:**

* **SearchBar** – Input field for city name.
* **WeatherDisplay** – Shows temperature, weather description, humidity, wind, etc.
* **Loader/ErrorDisplay** – Manages feedback during fetch or failure.
* **App Container** – Coordinates state, renders children.

## Phase 3: Implementation.

The code was developed in incremental steps aligned with the planned design.

**Key Development Milestones:**

* **App Setup:**
  + Initialized with create-react-app.
  + Created main component structure (App.js, SearchBar.js, WeatherDisplay.js).
* **API Integration:**
  + Connected to OpenWeatherMap via axios.
  + Handled API key securely and structured the data fetching function.
* **UI Logic:**
  + Controlled form input for city.
  + Managed loading, success, and error states with hooks.
* **Data Handling:**
  + Parsed and displayed temperature, weather icons, and metadata.
  + Handled fallback for missing data.
* **Styling:**
  + Designed layout for both mobile and desktop views.

## Phase 4: Testing.

The app was tested to ensure proper functionality and visual consistency.

**Testing Strategies:**

* **Unit Tests (Basic):**
  + Checked rendering of components based on props.
  + Validated API fetch logic (mocked API responses).
* **Manual Testing:**
  + Tested city searches with valid and invalid inputs.
* **Cross-Browser Compatibility:**
  + Verified UI across major browsers (Chrome, Firefox, Edge).
* **Edge Case Handling:**
  + Empty input → prompt user to enter a city.
  + City not found → show error message.
  + Invalid API response → fallback UI with retry option.

## Phase 5: Deployment.

The weather app was prepared and deployed for public access.

**Deployment Steps:**

* **Build Optimization:** Used npm run build to create a production build.
* **Hosting:** Deployed to GitHub Page for public access.
* **Repository:** Full project hosted on GitHub with:
  + README.md (setup instructions)
  + Code documentation
  + Screenshots of UI

## Phase 6: Maintenance.

Although the app is complete, we have plans for future support and enhancements.

**Planned Maintenance:**

* **Monitor for Bugs:** GitHub Issues tracking minor bugs or UI quirks.
* **Feature Enhancements (Future Roadmap):**
  + Add geolocation-based weather detection.
  + Display 5-day forecast.
  + Toggle between °C and °F.
* **Dependency Management:**
  + Regular updates for React, Axios, and other libraries.

# Part 2: Software Development Life Cycle.

The project was developed in the GitHub environment, therefore, the SDLC breakdown and individual contributions are well documented and available at:

<https://github.com/TelmuunDn/weather-app-ca2-aoe.git>

**Project overview.**

The goal of this project was to create a weather application using React that allows users to search for any city and view real-time weather information. The app shows key details such as temperature, weather condition, humidity, and wind speed. It fetches data from a public weather API and presents it in a clean, user-friendly interface.

This project helped us practice using React features like components, hooks (useState, useEffect), and working with external APIs. We used GitHub for version control and collaboration, which made it easier to keep track of our code and work as a team. The project was a great way to apply what we’ve learned in a real-world setting and build something useful.