CCT College Dublin

Assessment Cover Page

|  |  |
| --- | --- |
| Module Title: | Cross Platform Development |
| Assessment Title: | group |
| Lecturer Name: | David González |
| Student Full Name: | Youssef sabry, Daniel Bobuh, Louis Esvenn Canilang, Terence Teche |
| Student Number: | 2023318, 2023468, 2023429, 2023303 |
| Assessment Due Date: | 28/05/2025 |
| Date of Submission: | 28/05/2025 |

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. |

Documentation

[Documentation: 4](#_Toc199273773)

[Component Documentation 4](#_Toc199273774)

[HomeScreen (Main Component) 4](#_Toc199273775)

[searchCityWeather() 4](#_Toc199273776)

[useCurrentLocation() 4](#_Toc199273777)

[loadHistory(), saveToHistory(), and clearHistory() 4](#_Toc199273778)

[City Search Field (TextInput) 4](#_Toc199273779)

[Hourly Forecast Scroll 4](#_Toc199273780)

[Seven-Day Forecast 5](#_Toc199273781)

[Weather Display Card 5](#_Toc199273782)

[Temperature Unit Toggle 5](#_Toc199273783)

[Theme Toggle (Dark and Light Mode) 5](#_Toc199273784)

[Toast Notifications 5](#_Toc199273785)

[Offline Detection 5](#_Toc199273786)

[Responsive Design 5](#_Toc199273787)

[API Integration 6](#_Toc199273788)

[Geocoding API – OpenStreetMap (Nominatim) 6](#_Toc199273789)

[Weather API – Open-Meteo 6](#_Toc199273790)

[Error Handling and API Reliability 7](#_Toc199273791)

# 

# Documentation:

## Component Documentation

### HomeScreen (Main Component)

This is the main functional component located in the index.tsx file. It manages the app's layout, user interactions, and all core logic. It includes the search input, weather display, and user preferences such as temperature unit and theme. All state, hooks, and helper functions are implemented within this component.

### searchCityWeather()

This function is triggered when the user searches for a city. It first checks for internet connectivity, then sends a request to the OpenStreetMap API to get the city's geographic coordinates. Using those coordinates, it fetches current and forecasted weather data from the Open-Meteo API. If the city is found and the data is retrieved successfully, it updates the weather state and saves the city to local search history. If the city is not found or an error occurs, it displays an error using a toast notification.

### useCurrentLocation()

This function requests the user’s permission to access device location using the expo-location library. If permission is granted, it retrieves the device's latitude and longitude and fetches corresponding weather data from the Open-Meteo API. If permission is denied or location fails, an error toast is displayed. The results are labeled as "Current Location" in the UI and stored in search history.

### loadHistory(), saveToHistory(), and clearHistory()

These functions manage city search history using AsyncStorage. loadHistory retrieves previously saved cities when the app starts. saveToHistory stores the most recent city searches, maintaining a maximum of five entries. clearHistory removes all saved history from storage when triggered by the user.

### City Search Field (TextInput)

This is the input field where users type the name of a city. As the user types, a filtered list of suggestions appears below based on previous searches. Once the user presses the search icon, the searchCityWeather function is executed.

### Hourly Forecast Scroll

This component displays a horizontally scrollable list of weather conditions for the next twelve hours. Each item shows the hour, the expected temperature, and a weather icon based on the Open-Meteo code.

### Seven-Day Forecast

This section shows a vertical list of the upcoming seven days. For each day, it displays the date label, a weather condition indicator, and the high and low temperatures. The data is pulled from the daily forecast values returned by the weather API.

### Weather Display Card

This section displays the current weather information, including city name, country, temperature, weather condition, wind speed, and humidity. It also includes animations that fade the content into view when data is loaded.

### Temperature Unit Toggle

This toggle allows the user to switch between Celsius and Fahrenheit. It updates the state to convert all temperature values accordingly, both in the current weather and the forecast.

### Theme Toggle (Dark and Light Mode)

This toggle switches the app’s appearance between light and dark themes. It adjusts background color, text color, and icon appearance. It also includes an animated rotation effect when toggled.

### Toast Notifications

The app uses toast messages to inform users about various issues such as no internet connection, location errors, and unsuccessful searches. This improves user feedback during interaction.

### Offline Detection

The app uses the @react-native-community/netinfo library to detect whether the device has an internet connection. If no connection is found, the app prevents data requests and displays an error message.

### Responsive Design

The app uses the useWindowDimensions and Platform APIs to adjust its layout based on screen size and device type. It adapts fonts, padding, and layout to support both phones and tablets, as well as portrait and landscape modes.

## API Integration

### Geocoding API – OpenStreetMap (Nominatim)

The app uses the OpenStreetMap Nominatim API to convert user-entered city names into geographic coordinates (latitude and longitude). This step is required because the weather API requires coordinates rather than city names.

The API endpoint used is a formatted URL like the following:

“https://nominatim.openstreetmap.org/search?format=json&addressdetails=1&q={city}”

The request is sent using a standard fetch call. The response contains an array of location results. The app extracts the first result, including its latitude, longitude, city name, and country, which are used to request weather data.

If no matching location is found, the app displays an error message using a toast notification. If the API call fails or returns an empty array, the app informs the user that the city could not be found.

The geocoding logic is located in the searchCityWeather function inside index.tsx.

### Weather API – Open-Meteo

The app fetches weather data using the Open-Meteo API. This service provides current weather conditions, hourly forecasts, and daily forecasts based on geographic coordinates.

The API endpoint used is structured as follows:

“https://api.open-meteo.com/v1/forecast?latitude={lat}&longitude={lon}&current\_weather=true&hourly=temperature\_2m&daily=temperature\_2m\_max,temperature\_2m\_min,weathercode&timezone=auto”

The URL parameters include:

* latitude and longitude: provided by the geocoding or GPS location functions.
* current\_weather=true: enables current weather data.
* hourly=temperature\_2m: includes hourly temperature data for forecasting.
* daily=temperature\_2m\_max,temperature\_2m\_min,weathercode: provides the daily high and low temperatures and general weather condition codes.
* timezone=auto: ensures dates and times are returned in the local timezone of the coordinates.

Once data is retrieved, the app updates its state variables to display the information. It extracts and uses:

* Current temperature
* Hourly temperature values
* Seven-day forecasts
* Weather condition codes to select visual indicators

The weather data is displayed in both the main weather card and the forecast sections. If the API call fails or returns an error, the app shows a relevant toast notification and does not attempt to update the screen.

### Error Handling and API Reliability

Both API calls are wrapped in try-catch blocks. In case of failure (e.g., network error, invalid response, or server downtime), the app handles these gracefully by:

* Showing toast notifications
* Preventing state updates
* Notifying the user to try again or check their internet connection

Additionally, the app uses the NetInfo library to check if the device is online before attempting API requests. This helps prevent unnecessary fetch attempts when the device has no internet connection.