Project Report on
Weather Application using Javascript and API

Submitted in partial fulfillment of the requirement for the award of the degree of

BSc. (Hons.) Computer Science

GALGOTIAS UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of Name of Supervisor:

Mr Ashwin Perti

Submitted by –

Dharmendra Bhadauria – 20SCSE1100004
Tej Pratap Singh – 20SCSE1100002
Prashant Kumar – 20SCSE1100020

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA**

16 May 2023

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in this Project Report entitled **"WEATHER APPLICATION USING JAVASCRIPT AND API"** is in partial fulfillment of the requirements for the award of the BSc. Computer science submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of 1 Februrary 2023 to 31 April 2023, under the supervision of Mr. Ashwin Perti, Department of Computer Science and Engineering of School of Computing Science and Engineering, Galgotias University, Greater Noida.

The matter presented in the project has not been submitted by us for the award of any other degree of this or any other place.

Prashant Kumar – 20SCSE1100020

Tej Pratap Singh – 20SCSE1100002

Dharmendra Bhadauria – 20SCSE1100004

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Mr. Ashwin Perti

CERTIFICATE

The Final Project Viva-Voce examination of

Prashant Kumar – 20SCSE1100020 Tej Pratap Singh – 20SCSE1100002 Dharmendra Bhadauria – 20SCSE1100004 has been held on 16 May 2023 and their work is recommended for the award of BSc (Hons) Computer Science

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Program Chair **Signature of Dean**

Date: May 2023

Place: Greater Noida

TABLE OF CONTENTS

Abstract

Weather forecasting refers to the process in which science and technology are applied to predict the conditions of the atmosphere for a given location and time.

Just like accurate weather forecasting is important, it is crucial to present that information in a way that is easy to understand. Any software or model is considered good only when it can give a good user experience, no matter how good or accurate a model is if it is giving information in a complex way then the user experience will be bad.

Most of the people start their day by checking the weather information so they can prepare themselves for the day for example if there are chances of rain they can carry an umbrella. So weather information is crucial for people as it helps them to make their plans, prepare themselves for the day, in their clothing these are just the things that are solved on a daily basis.

So in this project, we tried to solve the front or the problem that arises in the interaction between the application and the user i.e user interface. Our project is created using the front-end library of Javascript called React and the data or the forecasting is done by a third party, we are just using it assuming it is correct with the help of API calls. Our focus is to create an interface that gives information in a glance means all the information is easily shown and simple for the user to understand.

1. Introduction

1.1 Problem Statement

Weather forecasting plays a crucial role in our daily lives, helping us plan activities, make informed decisions, and stay prepared for changing weather conditions. While numerous organizations provide highly accurate weather information, the way this information is presented often falls short of user expectations. In this context, we have developed a weather forecasting application aimed at bridging the gap between accurate weather data and an intuitive user interface.

There are a lot of organizations that provide highly accurate information regarding forecasting but the way of their presentation is not up to the par. The reasons might be since they provide this information free of cost they depend heavily on the ads and keep their focus on the backend means on forecasting instead of presenting

- Many of the free weather software or websites programs have too many popups and unwanted software tied to them like weather bug
- Getting confusing information on weather warnings and watches from inaccurate sources
- Sometimes using these apps become complex due to lots of advertisement

1.2 Problem Statement

One of the primary challenges faced by users is the presentation of weather information. Many free weather software or websites depend heavily on advertisements, leading to an inundation of pop-ups and unwanted software. This detracts from the overall user experience and makes it difficult to access reliable weather forecasts. Additionally, inaccurate sources sometimes provide confusing information on weather warnings and watches, further complicating matters. The excessive advertisements in these apps also contribute to a complex and cluttered user interface, hampering usability.

1.3 Issue

During the development of our weather forecasting application, we identified several issues that required attention. These issues include a malfunctioning temperature conversion feature, an unresponsive background change according to temperature constraints, a need for UI improvements to enhance subtlety, and a requirement for enhanced dynamic properties to optimize user interactions.

- The conversion feature of changing temperature in Celsius and Fahrenheit is not working properly some error in API call
- The background change according to the temperature constraints
- Improving UI to make it more subtle
- Improving dynamic properties of the application

1.4 Solution

To address these issues and provide an improved user experience, we have devised a comprehensive solution strategy. Our approach includes changing the application's template, transitioning from basic HTML to React for enhanced flexibility and structure, and incorporating additional media queries to ensure responsiveness across different devices. We aim to resolve the temperature conversion issue by thoroughly reviewing the API call responsible for this functionality. Furthermore, we will revamp the user interface to make it more subtle and user-friendly, enhancing the dynamic properties for smoother interactions.

- Will try to change the template of the site
- We changed the layout of our initial template after switching over react from basic html
- we will add more media Queries
- By adding a constraint function to check the current temperature and updating the background accordingly
- Checking the fetching method to see the issue in units while calling the API

1.5 Tools and Technology

To implement our solutions, we will leverage various front-end and back-end technologies. The front-end will be developed using React, which enables efficient application structuring. We will utilize the Tailwind CSS framework to style the application and Google Font to incorporate diverse font choices. Additionally, Toastify will be employed to add loading animations and improve the user experience.

On the back-end, we will rely on the OpenWeatherMap API to retrieve the necessary weather data. The Luxon library will provide robust time and date functionalities, while the Fetch API will facilitate seamless data retrieval.

Front-end Technology

- React for creating structure of the application
- CSS framework Tailwind for styling the application
- Google Font – to use different fonts in our application
- Toastify – to add loading animations

Back-end Technology

- Openweathermap API to get the required data

- Luxon library for using the Time-date Functionalities Fetch API call to get the required data

1.6 Functionalities

We can search the city name we want to know the weather about

We can also directly access the current location by accessing the GPS information through the browser
We give information for +5 hours from the current time on an hourly basis

It also gives some additional information such as mentioned below when checking out today’s weather conditions

- Sunrise time
- Sunset time
- Wind speed
- Humidity
- Feels like

We can also check the average out temperature (means the average temperature of the day) for the coming 5 days
It also change the background of the application on the basis of the current temperature of that location to give a nice graphical touch

We have added an AQI option that fetch the AQI of the city along with that it has fetch some other information on the major pollution causing gases

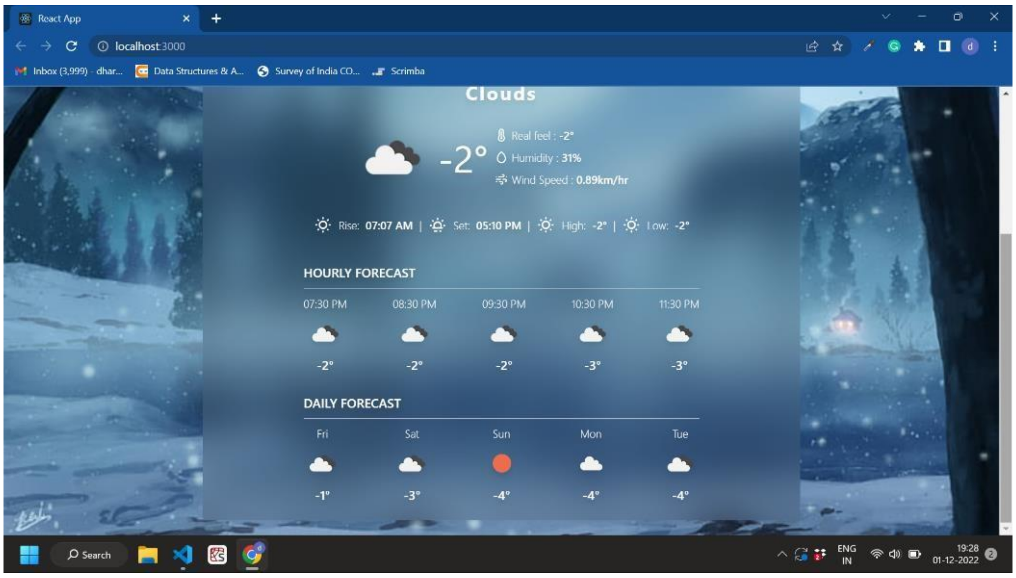
- CO
- SO2
- PM2
- PM10

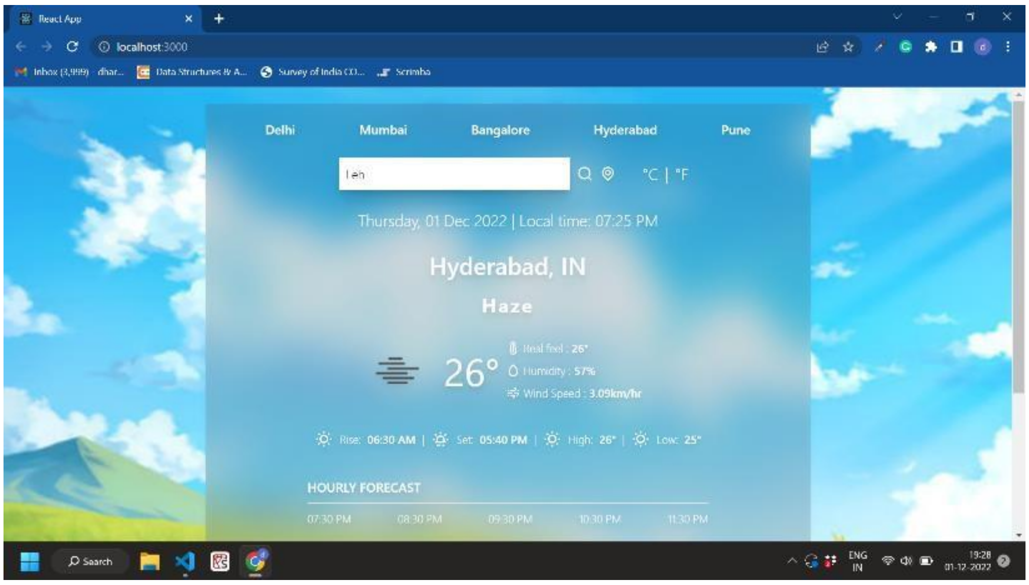
1.7 Future Plans

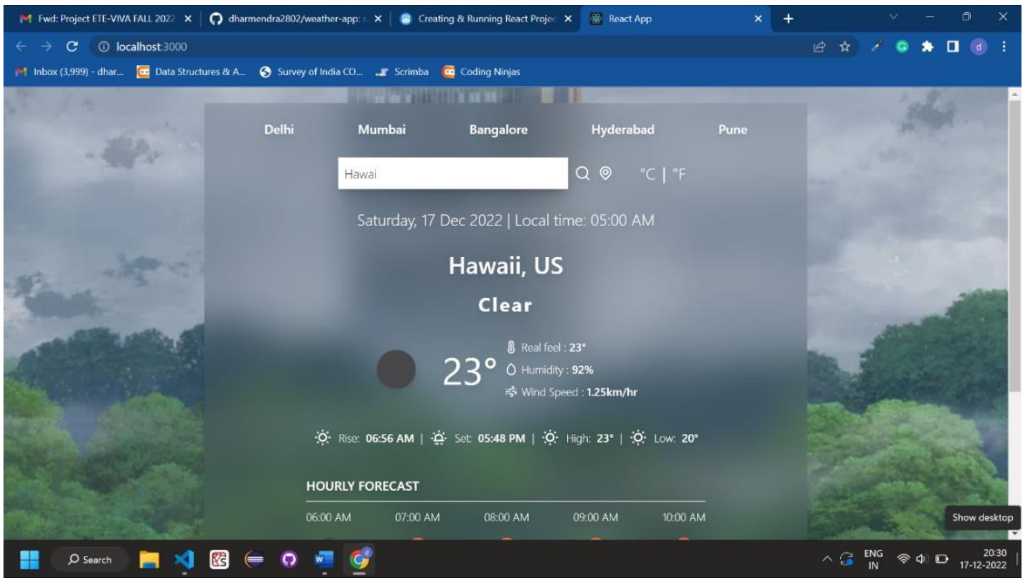
In our commitment to continuous improvement, we have outlined several future plans for the weather forecasting application. We aim to enhance the application's animations, making them more accurate and refining existing animations for smoother transitions. Moreover, we will focus on making the application more dynamic and responsive, ensuring optimal user interfaces across various mobile devices. Additional features in our pipeline include a toggle feature to display the loading process, a section to showcase recent searches, weather alerts for specific cities, a map feature displaying satellite images, time and condition-based background updates, and real-time traffic conditions.

- We want to add more animation so it will become more accurate and also to brush up the current existing animations to make it more smooth
- Make it more dynamic so it can also provide a good UI on different mobile devices as well
- To add a toggle feature to show the loading process
- Show recent searches in upper city section Some additional information such as
  - Alert functionalities – if any warning is issued about the city in question
  - Map feature – to give a satellite image of the city
  - Time and condition-based background – update background considering both time and the weather conditions of that place
  - Traffic conditions

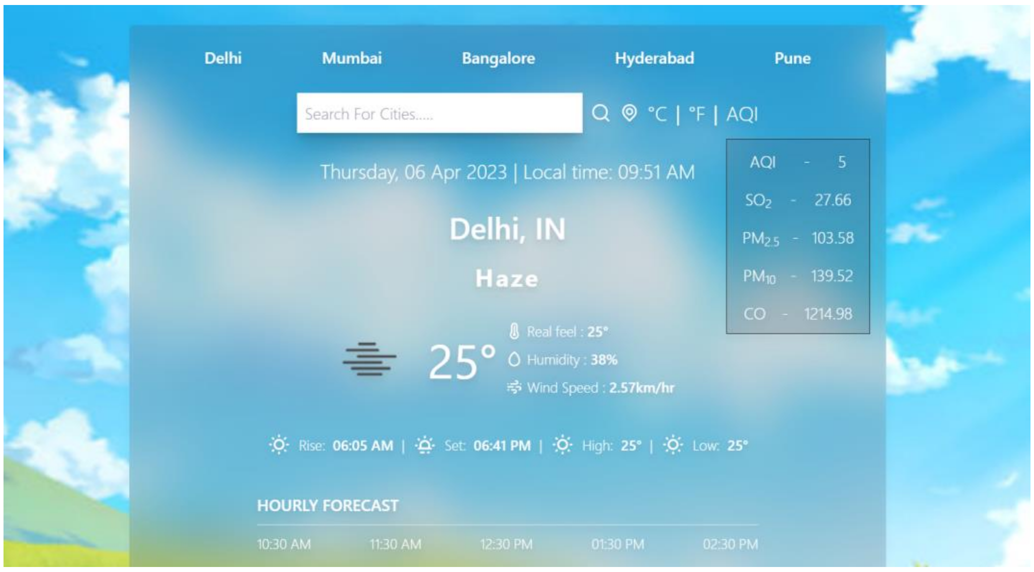
1.7 Output


React App
localhost:3000
Clouds
-2°
Real feel : -2°
Humidity : 31%
Wind Speed : 0.89km/hr
Rise: 07:07 AM | Set: 05:10 PM | High: -2° | Low: -2°
HOURLY FORECAST
07:30 PM -2°
08:30 PM -2°
09:30 PM -2°
10:30 PM -3°
11:30 PM -3°
DAILY FORECAST
Fri -1°
Sat -3°
Sun -4°
Mon -4°
Tue -4°

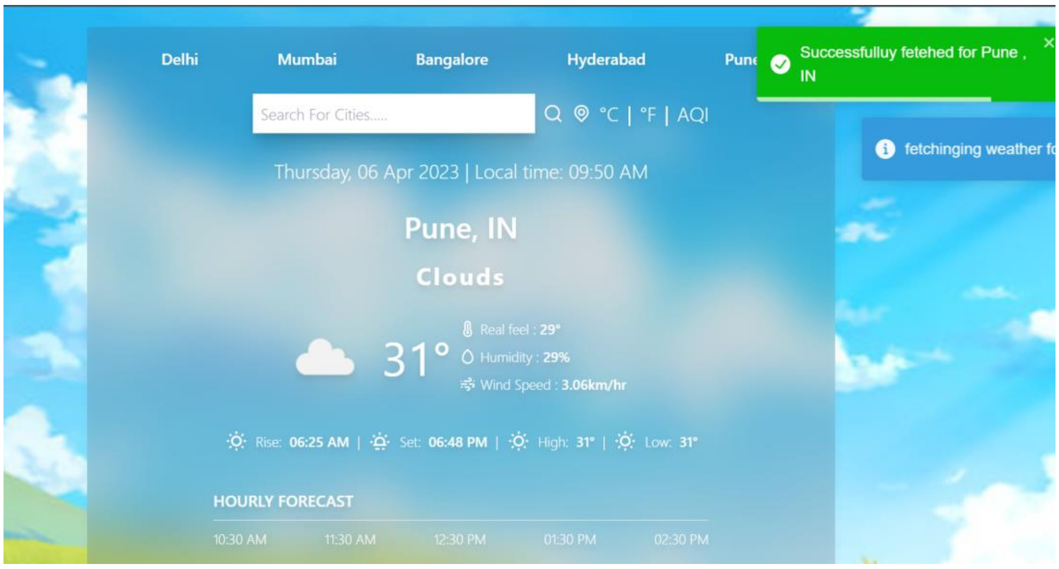

React App
localhost:3000
Delhi
Mumbai
Bangalore
Hyderabad
Pune
Leh
°C | °F
Thursday, 01 Dec 2022 | Local time: 07:25 PM
Hyderabad, IN
Haze
26°
Real feel : 26°
Humidity : 57%
Wind Speed : 3.09km/hr
Rise: 06:30 AM | Set: 05:40 PM | High: 26° | Low: 25°
HOURLY FORECAST
07:30 PM
08:30 PM
09:30 PM
10:30 PM
11:30 PM



1.7.1 Updated –



AQI	-	5
SO_2	-	27.66
$PM_{2.5}$	-	103.58
PM_{10}	-	139.52
CO	-	1214.98



Successfulluy fetehed for Pune , IN

fetchinging weather f

Thursday, 06 Apr 2023 | Local time: 09:50 AM

Pune, IN

Clouds

31°

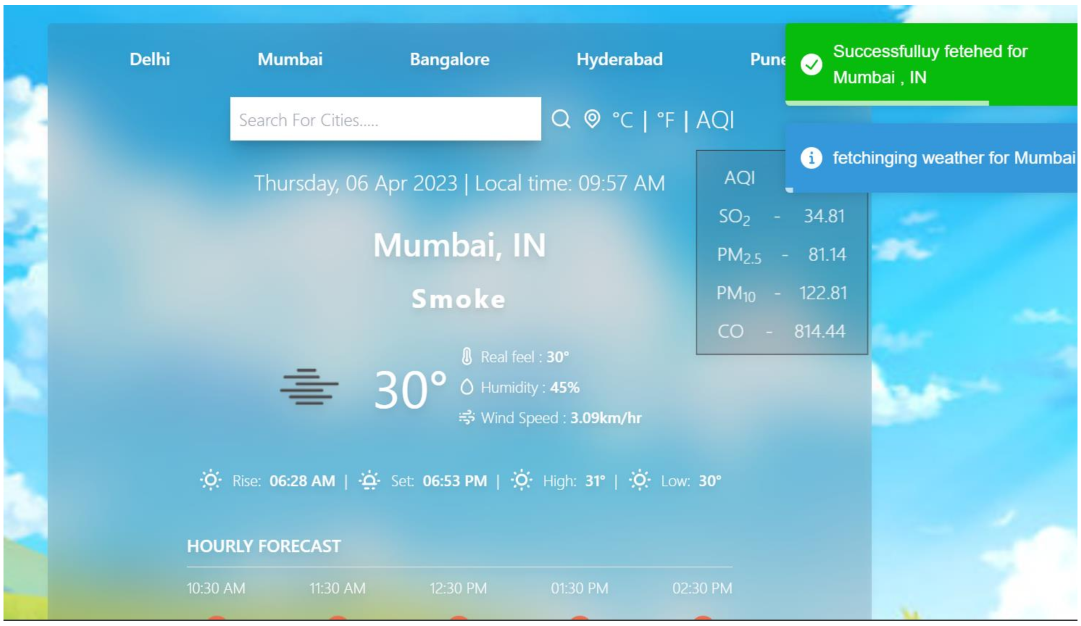
Real feel : 29°

Humidity : 29%

Wind Speed : 3.06km/hr

Rise: 06:25 AM | Set: 06:48 PM | High: 31° | Low: 31°

10:30 AM 11:30 AM 12:30 PM 01:30 PM 02:30 PM



Successfulluy fetehed for Mumbai , IN

fetchinging weather for Mumbai

Thursday, 06 Apr 2023 | Local time: 09:57 AM

Mumbai, IN

Smoke

30°

Real feel : 30°

Humidity : 45%

Wind Speed : 3.09km/hr

Rise: 06:28 AM | Set: 06:53 PM | High: 31° | Low: 30°

AQI		
SO_2	-	34.81
$PM_{2.5}$	-	81.14
PM_{10}	-	122.81
CO	-	814.44

10:30 AM 11:30 AM 12:30 PM 01:30 PM 02:30 PM

1.8 Literature Review

1.8.1.1 Weather Forecasting and the Need for Web Applications:
Weather forecasting plays a vital role in our daily lives, providing us with valuable information about the atmospheric conditions that influence our activities and decision-making. Accurate weather forecasts enable us to plan our outdoor activities, prepare for severe weather events, and make informed choices regarding travel and clothing. As technology continues to advance, the availability and accessibility of weather information have greatly improved. Web applications have emerged as a popular platform for delivering real-time and localized weather forecasts to users worldwide.

1.8.1.2 Overview of Weather Forecasting Web Applications:
Weather forecasting web applications leverage the power of the internet and modern web technologies to provide users with up-to-date and location-specific weather information. These applications utilize a combination of data sources, algorithms, and user interfaces to deliver forecasts in a user-friendly and interactive manner. They offer features such as current weather conditions, hourly and daily forecasts, radar and satellite imagery, and additional information such as wind speed, humidity, and precipitation levels. Weather forecasting web applications have become indispensable tools for individuals, businesses, and organizations relying on accurate weather information.

HTML, CSS, and JavaScript: The Foundation of Web Development:

1.8.2.1 HTML: Structure and Content:
HTML (Hypertext Markup Language) forms the foundation of web development. It provides the structure and content of web pages, defining the elements and their relationships. Weather forecasting web applications utilize

HTML to create the layout and structure of the application, including headings, paragraphs, tables, forms, and images. HTML tags and attributes define the semantic structure and presentation of weather information, ensuring proper rendering across different devices and browsers.

1.8.2.2 CSS: Visual Presentation and Layout:

CSS (Cascading Style Sheets) is responsible for the visual presentation and layout of web pages. In weather forecasting web applications, CSS is used to style various elements, including fonts, colors, backgrounds, borders, and layouts. CSS frameworks, such as Tailwind CSS, provide pre-defined styles and utilities that enable developers to create visually appealing and responsive user interfaces. With CSS, weather forecasting web applications can achieve consistent and aesthetically pleasing designs, enhancing the user experience.

1.8.2.3 JavaScript: Interactivity and Dynamic Functionality:

JavaScript is a powerful scripting language that enables interactivity and dynamic functionality in weather forecasting web applications. It allows developers to create interactive elements, handle user input, and manipulate data in real-time. JavaScript frameworks like React provide a component-based architecture that facilitates the development of reusable and modular code. With JavaScript, weather forecasting web applications can update data dynamically, respond to user actions, and provide seamless user experiences.

React: Dynamic Components and State Management:

1.8.3.1 Component-Based Architecture:

React, or React.js, is a popular JavaScript library for building user interfaces. Its component-based architecture allows developers to create reusable and independent components that can be composed to form complex web applications. In weather forecasting web applications, React components can represent different sections of the interface, such as the current weather display, hourly forecasts, and interactive maps. This modular approach enhances code reusability, maintainability, and scalability.

1.8.3.2 Reusability and Modularity:

React promotes reusability and modularity through its component-based approach. Components encapsulate their logic, styling, and rendering, allowing them to be easily reused across different parts of the application. For example, a weather forecast card component can be reused to display hourly or daily forecasts. This reusability simplifies development, reduces code duplication, and ensures consistency throughout the application.

1.8.3.3 State Management in React :

React's state management allows weather forecasting web applications to handle dynamic data and respond to changes in real-time. The useState hook enables components to maintain and update their internal state, such as the selected location, temperature units, or forecast data. By updating the state, components can trigger re-rendering and reflect the latest changes in the user interface.

State management also plays a crucial role in handling user interactions and providing a seamless user experience. For example, when a user selects a different location or changes the temperature units from Celsius to Fahrenheit, the application can update the corresponding state variables and fetch the relevant weather data. This dynamic updating of state ensures that the displayed information is always accurate and up to date.

OpenWeatherMap API: Accessing Comprehensive Weather Data:

1.8.4.1 Introduction to OpenWeatherMap API:

Weather forecasting web applications rely on external data sources to retrieve accurate and comprehensive weather information. One such popular data source is the OpenWeatherMap API. OpenWeatherMap provides a range of APIs that offer minute-by-minute, hourly, and daily forecasts, historical weather data, national weather alerts, and more. These APIs allow developers to access a wealth of weather data and integrate it seamlessly into their applications.

1.8.4.2 Minute-by-Minute, Hourly, and Daily Forecasts:

The OpenWeatherMap API provides minute-by-minute, hourly, and daily forecasts, enabling weather forecasting web applications to deliver detailed and precise weather information. Minute-by-minute forecasts offer real-time updates on weather conditions within a short time frame, while hourly forecasts provide a more extended outlook. Daily forecasts give users an overview of the expected weather conditions for the upcoming days. By leveraging these forecasts, weather forecasting web applications empower users to plan their activities with accuracy.

1.8.4.3 Historical Weather Data:

In addition to current and future forecasts, the OpenWeatherMap API offers access to historical weather data. This historical data allows weather forecasting web applications to provide users with insights into past weather patterns and trends. It can be used for research purposes, trend analysis, or even for personal interests. By incorporating historical weather data, these applications enhance their informational value and provide a more comprehensive understanding of weather patterns.

1.8.4.4 Integration and Implementation in Web Applications:

Integrating the OpenWeatherMap API into weather forecasting web applications involves making HTTP requests to retrieve the desired weather data. This data can be fetched using technologies like JavaScript's Fetch API, which allows applications to send network requests and receive responses asynchronously. The API response can then be parsed and displayed in the user interface, providing users with accurate and relevant weather information.

Fetching weather data from the OpenWeatherMap API requires specifying the location, desired forecast parameters (e.g., temperature, humidity, wind speed), and appropriate API endpoints. By leveraging the power of the API, weather forecasting web applications can access a vast repository of weather data and present it to users in a user-friendly and visually appealing manner.

Fetch API: Seamless Network Communication:

1.8.5.1 Asynchronous Requests:

The Fetch API, available in modern web browsers, enables weather forecasting web applications to communicate with external APIs and retrieve data asynchronously. Asynchronous requests prevent blocking the main thread, ensuring a smooth user experience even when fetching data from remote sources. This asynchronous nature allows the application to continue executing other tasks while waiting for the response from the API.

1.8.5.2 Retrieving Data from External Sources:

With the Fetch API, weather forecasting web applications can send HTTP requests to external APIs, such as the OpenWeatherMap API, and retrieve the requested data. These requests can include parameters such as the desired location, forecast duration, or specific weather parameters to fetch the relevant information. The API response, typically in JSON format, can then be processed and displayed in the application's user interface.

1.8.5.3 Integrating Fetch API with Weather Forecasting Web Applications:

Integrating the Fetch API with weather forecasting web applications involves a series of steps to handle network requests and process the received data.

Firstly, the application needs to construct the API request URL by specifying the necessary parameters, such as the location and forecast type. This URL is then passed to the Fetch API's fetch() function to initiate the request.

Once the request is sent, the Fetch API returns a Promise that resolves to the server's response. The application can use JavaScript's Promise-based syntax, including the .then() method, to handle the response.

In the .then() block, the received data is typically in JSON format. The application can parse this data using JSON.parse() to convert it into a usable JavaScript object. This object can then be further processed and extracted to retrieve the desired weather information.

After extracting the necessary weather data, the application can update the state or modify the user interface accordingly. For example, it can display the current temperature, humidity, or weather conditions in a visually appealing format.

Error handling is also a crucial aspect of integrating the Fetch API. The application can use the .catch() method to capture any network errors or failed requests. Proper error handling ensures that the application gracefully handles unexpected situations and provides a user-friendly experience even when network connectivity or API responses encounter issues.

By effectively integrating the Fetch API, weather forecasting web applications can seamlessly retrieve weather data from external sources, such as the OpenWeatherMap API, and present it to users in a reliable and timely manner.

Luxon Library: Efficient Date and Time Manipulation:

1.8.6.1 Introduction to Luxon Library:

In weather forecasting web applications, accurate manipulation and formatting of dates and times are essential for displaying time-sensitive weather information. The Luxon library is a powerful tool that simplifies working with dates and times in JavaScript. It provides a comprehensive interface for performing various operations on dates and times, such as addition, subtraction, formatting, parsing, and comparison.

1.8.6.2 Manipulating, Formatting, and Parsing Dates and Times:

Luxon offers a wide range of methods and utilities to manipulate dates and times. It enables weather forecasting web applications to add or subtract time intervals, extract specific components (year, month, day, hour, etc.), and perform complex temporal calculations. Luxon also supports formatting and parsing dates and times in different formats, allowing applications to present the information according to user preferences or localization requirements.

1.8.6.3 Simplifying Temporal Calculations:

Weather forecasting often involves complex temporal calculations, such as calculating the average temperature over a specific period or determining the time of sunrise or sunset. Luxon simplifies such calculations by providing intuitive methods and utilities. It handles time zone conversions, daylight saving time adjustments, and other intricacies, ensuring accurate and reliable results.

1.8.6.4 Enhancing Accuracy in Weather Forecasting Applications:

Accurate date and time manipulation are crucial for precise weather forecasting. Luxon's robust features and APIs enable weather forecasting web applications to handle time-related operations effectively. By leveraging Luxon, these applications can ensure accurate temporal calculations, present time-sensitive data in a user-friendly format, and deliver reliable and precise weather forecasts to users.

Tailwind CSS Framework: Styling with Efficiency and Customization:

1.8.7.1 Introduction to Tailwind CSS:

Tailwind CSS is a popular utility-first CSS framework used in web development, including weather forecasting web applications. It provides a comprehensive set of pre-defined utility classes that allow developers to style and customize their applications efficiently and effectively.

1.8.7.2 Utility-First Approach:

Unlike traditional CSS frameworks, Tailwind CSS follows a utility-first approach. It provides a vast collection of utility classes that directly apply specific CSS properties and values to elements. Developers can easily combine these classes to create complex styles without writing custom CSS

1.8.7.3 Rapid Customization and Styling Options:

Tailwind CSS offers extensive customization options, allowing developers to tailor the appearance of weather forecasting web applications to their specific needs. By leveraging configuration files, developers can customize colors, fonts, spacing, breakpoints, and more. This flexibility empowers developers to create unique and visually appealing designs that align with the application's branding and user experience requirements.

Additionally, Tailwind CSS provides a wide range of pre-designed components and layout options that can be easily integrated into weather forecasting web applications. These components offer ready-to-use styling for common UI elements, such as buttons, cards, grids, and navigation menus. By leveraging these components, developers can save time and effort in designing and implementing the user interface.

1.8.7.4 Responsive Design and User Interface Enhancement:

Responsive design is crucial for weather forecasting web applications, as they need to adapt to various screen sizes and devices. Tailwind CSS provides built-in utilities and responsive classes that enable developers to create responsive layouts effortlessly. These utilities allow elements to be displayed, hidden, or repositioned based on different screen sizes, ensuring a consistent and user-friendly experience across devices.

Furthermore, Tailwind CSS promotes the use of responsive design patterns, such as flexbox and grid, to create flexible and adaptive user interfaces. These patterns allow for seamless content alignment, stacking, and reordering, enhancing the readability and usability of weather forecasting web applications on different devices.

Recap of Technologies and Frameworks:

In this literature review, we explored the technologies and frameworks used in weather forecasting web applications. HTML, CSS, and JavaScript form the foundation of web development, providing structure, styling, and interactivity. React enhances the development process with its component-based architecture and state management capabilities. The OpenWeatherMap API offers comprehensive weather data, while the Fetch API facilitates seamless network communication. Luxon simplifies date and time manipulation, and Tailwind CSS provides efficient styling and customization options.

Significance of Accurate Weather Forecasts:

Accurate weather forecasts play a crucial role in various aspects of our lives, including outdoor activities, travel, agriculture, and disaster preparedness. Weather forecasting web applications bridge the gap between users and weather information, providing real-time and location-specific forecasts that enable informed decision-making. The technologies and frameworks discussed in this literature review contribute to the dvelopment of robust and user-friendly weather forecasting web applications, enhancing the accessibility and usability of weather data.

Future Trends and Developments in Weather Forecasting Web Applications:

Weather forecasting web applications will continue to evolve and incorporate new technologies and advancements. Machine learning and artificial intelligence techniques are being employed to improve forecast accuracy and provide more detailed insights. Integration with Internet of Things (IoT) devices, such as weather sensors and smart home systems, allows for personalized and localized forecasts. Additionally, advancements in data visualization techniques enable more intuitive and interactive representations of weather information.

As technology progresses, weather forecasting web applications will become increasingly sophisticated, delivering more accurate and personalized forecasts to users worldwide. These applications will continue to empower individuals, businesses, and organizations with valuable weather insights, enabling them to make informed decisions and adapt to changing weather conditions effectively.

2. Methodology

Agile Project Management:

We employed the agile project management methodology for the development of our weather forecasting web application. Agile is known for its iterative and collaborative approach, which was well-suited to our project. By embracing agile principles, we aimed to enhance the application's functionality through continuous feedback, adaptability, and regular incremental improvements.

2.1 Analysis:

To kickstart the development process, we conducted a comprehensive analysis of existing weather forecasting applications. Our objective was to understand their strengths, weaknesses, and the challenges faced by users. This analysis provided valuable insights into common pitfalls and opportunities for innovation. By studying competitor applications, we gained a deeper understanding of user expectations and industry trends, enabling us to differentiate our application.

2.2 Planning:

In the planning phase, we meticulously outlined the various components and steps required to build our weather forecasting web application. This involved detailed discussions and brainstorming sessions to identify the project's scope, objectives, and key deliverables. We established clear goals, defined user requirements, and prioritized features based on their impact and feasibility.

2.2.1 Data Requirement:

To ensure accurate weather forecasts, we carefully evaluated the data requirements of our application. We explored multiple weather data providers and APIs, considering factors such as data quality, availability, and cost. After thorough evaluation, we selected the OpenWeatherMap OneCall3.0 API as our primary data source. This API provided a rich set of weather data, including hourly and daily forecasts, historical data, and additional information such as humidity, wind speed, and sunset times.

2.2.2 Data Retrieval:

To retrieve weather data from the OpenWeatherMap API, we leveraged the JavaScript fetch method. This method allowed us to make asynchronous requests to the API endpoints and fetch the required data in a seamless manner. We implemented error handling mechanisms and utilized best practices for efficient data retrieval, ensuring smooth integration of weather data into our application.

2.2.3 Data Filtering:

Given the comprehensive nature of the OneCall3.0 API, which provided a wealth of weather information, we implemented a data filtering mechanism. We carefully extracted and stored only the relevant data required for our application's functionalities. This process involved parsing the API response, selecting the necessary data fields, and structuring it in a way that optimized storage and retrieval operations. By filtering the data, we minimized redundancy and improved the overall performance of our application.

2.2.4 Technology Used:

For building the structure and user interface of our application, we chose the React framework. React's component-based architecture, virtual DOM, and efficient rendering mechanism made it an ideal choice for developing a responsive and interactive web application. The use of React components facilitated code reusability, modularity, and easier maintenance. We followed industry best practices and design patterns to ensure clean code structure and scalability.

In terms of styling, we adopted the Tailwind CSS framework. Tailwind CSS's utility-first approach and extensive set of pre-defined classes expedited the styling process. It allowed us to rapidly prototype and customize the application's visual elements with minimal CSS code. The framework's responsive design capabilities ensured that our application looked visually appealing and performed well across different devices and screen sizes.

2.2.5 Information Fetching:

To fetch weather data from the server, we utilized JavaScript's fetch API. This powerful API enabled us to make HTTP requests to the OpenWeatherMap API endpoints, retrieve weather data, and handle the responses asynchronously. We implemented error handling, data parsing, and transformation logic to ensure the accuracy and reliability of the fetched data.

2.3 Designing:

During the designing phase, we focused on creating a user-friendly and intuitive interface that provided a seamless weather forecasting experience. Adhering to the principles of user-centered design, we emphasized simplicity, clarity, and ease of use. We conducted user research and incorporated user feedback to continuously refine the design. Our goal was to create a visually appealing and accessible interface that allowed users to effortlessly access and interpret weather information.

2.3.1 Layout and Navigation:

We designed a clean and intuitive layout that prominently displayed the current weather conditions while providing easy access to additional forecast details. We employed a consistent and logical navigation structure, ensuring that users could effortlessly navigate between different sections of the application, such as hourly forecasts, daily forecasts, and historical data.

2.3.2 Visual Presentation:

To enhance the visual appeal and usability of the application, we employed appropriate color schemes, typography, and iconography. We used visual cues, such as icons and animations, to convey weather conditions and create a more engaging user experience. Additionally, we ensured that the application's visual elements were responsive and optimized for different screen sizes, enabling users to access the application seamlessly across various devices.

2.3.3 User Interaction and Feedback:

We implemented interactive elements to enable users to personalize their weather forecasts and tailor the information according to their preferences. Users could input their location, select measurement units, and customize the display of weather data. We also incorporated feedback mechanisms to gather user input and suggestions, allowing us to continuously improve the application based on user needs and expectations.

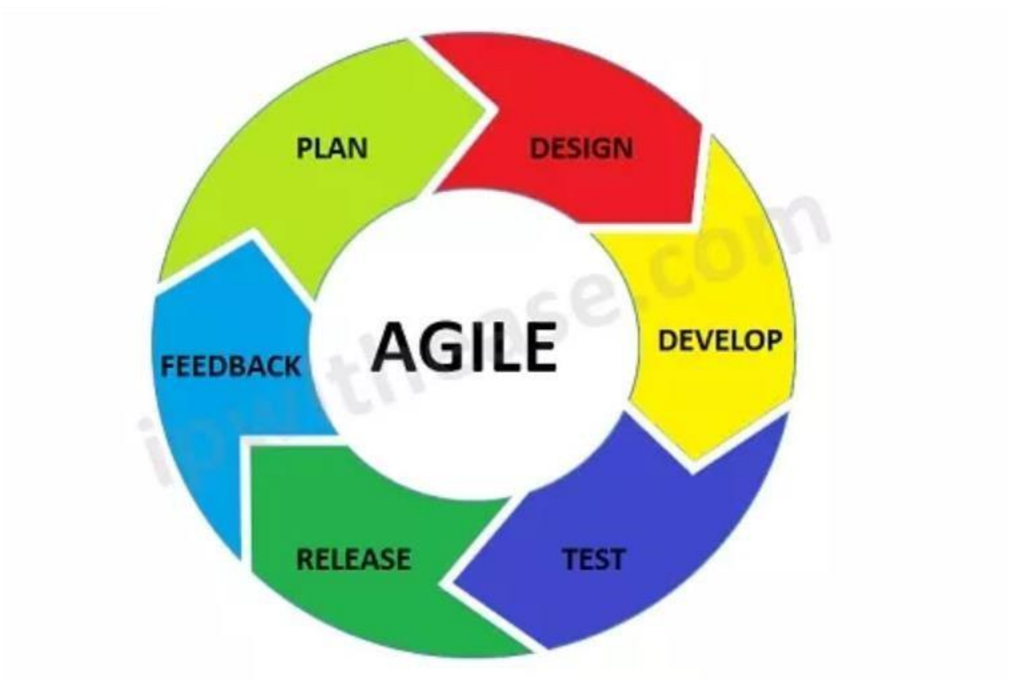
2.4 Test and Change:

Testing played a crucial role in ensuring the quality and functionality of our weather forecasting web application. We

conducted rigorous testing at each development stage, including unit testing, integration testing, and user acceptance testing. We identified and addressed any bugs, usability issues, or performance bottlenecks through thorough testing and continuous feedback loops.
Based on user feedback and the results of testing, we iteratively made changes and refinements to enhance the application's performance, usability, and overall user experience. This iterative approach allowed us to continually improve the application, adding new features, optimizing existing functionalities, and addressing any user pain points.

Throughout the development process, we emphasized open communication and collaboration within the development team. Regular meetings, discussions, and agile ceremonies such as daily stand-ups, sprint planning, and retrospectives enabled effective coordination, efficient task allocation, and timely issue resolution.

In conclusion, our methodology for developing the weather forecasting web application followed an agile approach, prioritizing user feedback and continuous improvement. By analyzing competitor applications, carefully planning our data requirements, utilizing appropriate technologies, implementing intuitive design, and conducting thorough testing, we aimed to deliver a high-quality and user-centric weather forecasting experience. The iterative nature of our development process allowed us to adapt to evolving user needs and incorporate new features to make our application more robust and comprehensive.

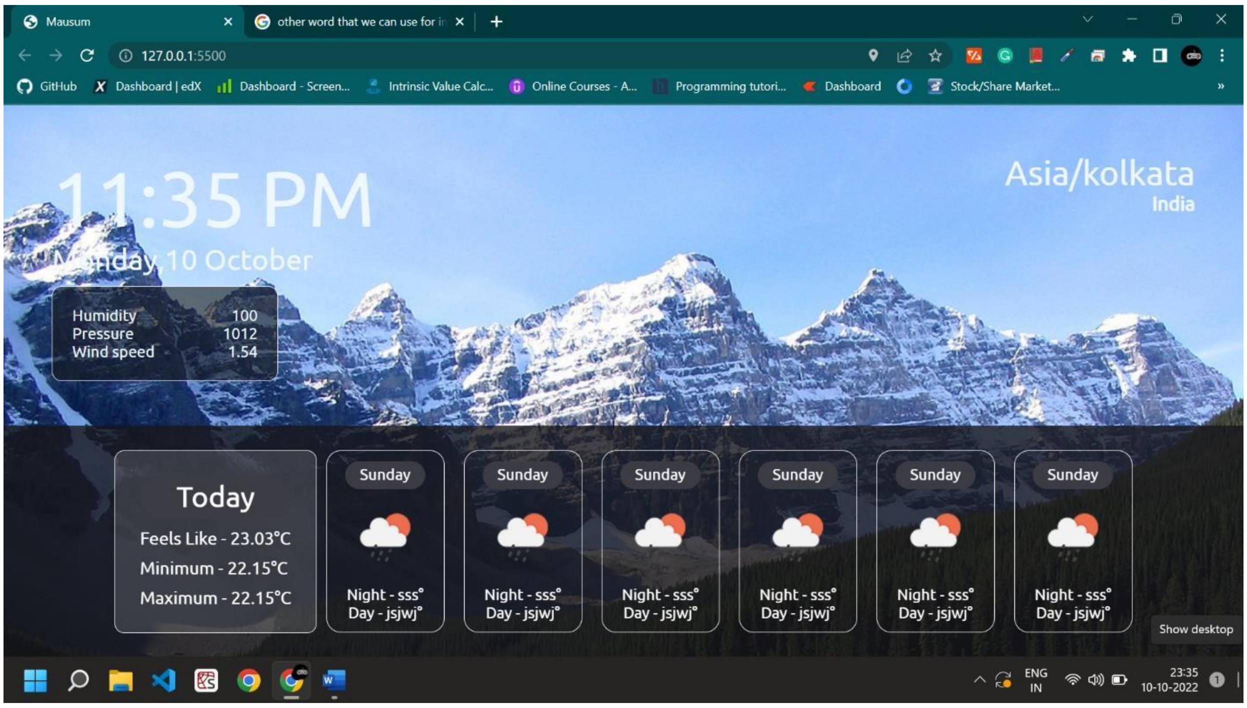


Initial Build

First, we created our application with simple HTML, CSS, and javascript, It supported only limited Functionalities like

 Showing weather information of just the current location

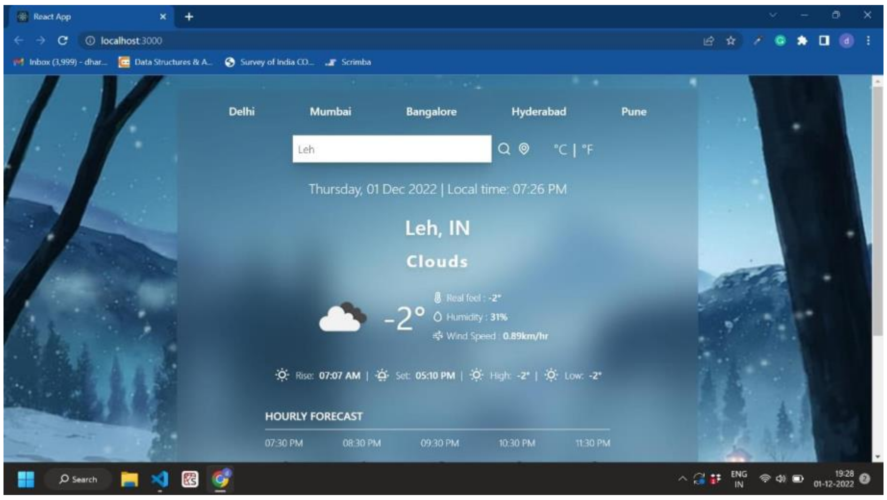
 Showing only the weather information of current day



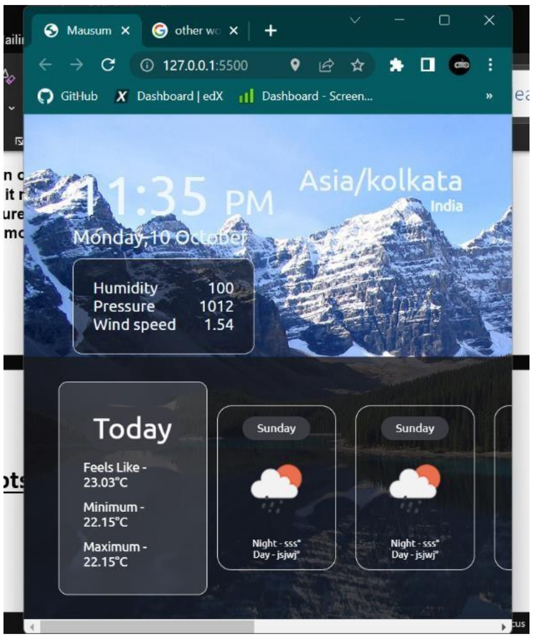
Updated build

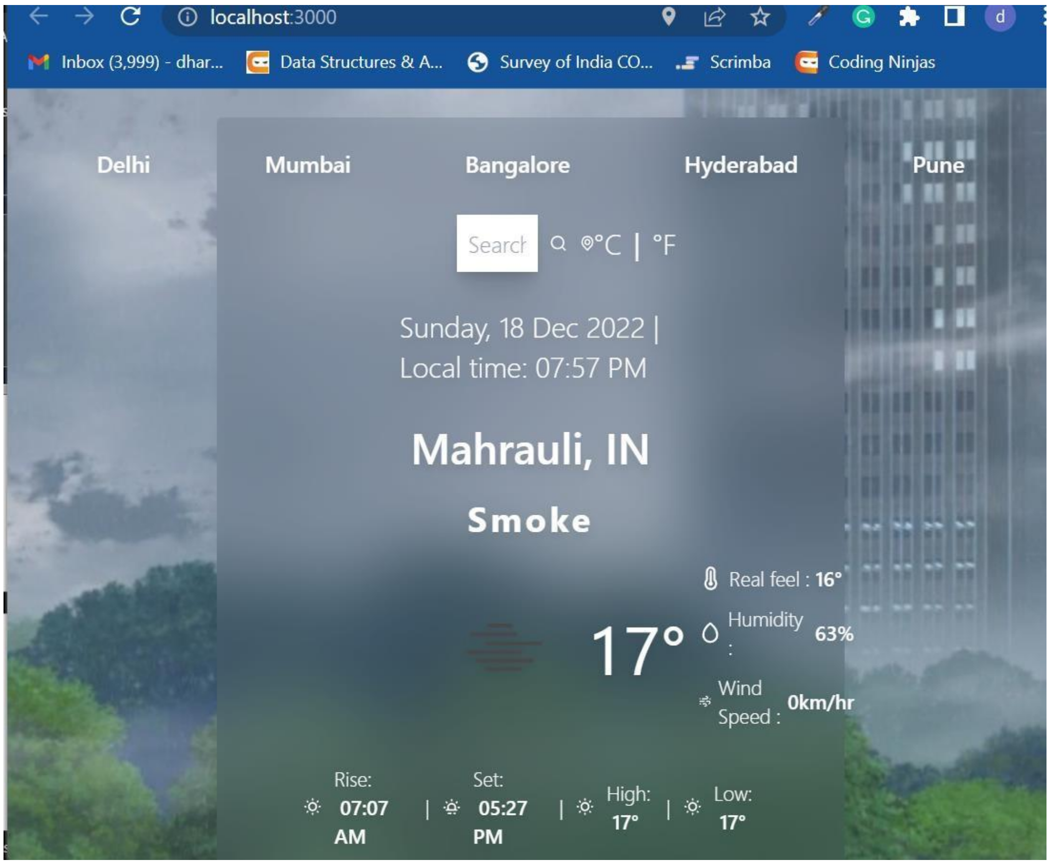
After getting the first feedback from the reviewer we switch to React and also tried to solve all the issues that we were having in our initial build Like –

- Search bar
- Hourly information
- Daily information
- Units Switch



However, the initial build has much more good dynamic capabilities compared to current build





Sunday, 18 Dec 2022 | Local time: 07:57 PM

Mahrauli, IN

Smoke

Real feel : 16°
17°
Humidity : 63%
Wind Speed : 0km/hr

Rise: 07:07 AM | Set: 05:27 PM | High: 17° | Low: 17°

3.Code

3.1 - TopButtons.jsx – it is one of the components in the react application that we have created to show the upper section of the app which shows five major cities of India

```
import React from 'react'
 export default function TopButtons({setQuery})
{   const cities = [
    {
id:1,
      title: "Delhi"
    },
{
id:2,
      title: "Mumbai"
    },     {
id:3,        title:
"Bangalore"
    },     {
id:4,        title:
"Hyderabad"
    },
{
id:5,
      title: "Pune"
    }
  ]
  return (
    <div className='flex items-center justify-around '>
      {cities.map((city) => (
        <button key={city.id} className='text-white text-md font-medium mx-10
hover:scale-125 transition ease-in-out'
        onClick={() => setQuery({q: city.title})}>
          {city.title}
        </button>
      ))}
    </div>





  )
}
```

<u>3.2 - Input.jsx</u> – this is the second component it hold a searchbar to search any custom city also it provide functionalities ti search current location. Along with this it has two options to show the data either in standard or imperial format

```
import React, { useState } from 'react'
import {UilSearch , UilLocationPoint} from "@iconscout/react-unicons"
import { ToastContainer,toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { latitude,longitude,API_KEY } from '../services/weatherService';


function Input({setQuery , units , setUnits}) {

  const [city,setCity] = useState("");


  const [aqi, setAqi] = useState(null);
  const [components, setComponents] = useState("");

  const handleSearch = () => {
    if(city !== '') setQuery({q: city})
  }

  const handleloco = () => {
    if(navigator.geolocation){
      toast.info('Fetching Location');
      navigator.geolocation.getCurrentPosition((position) => {
        toast.success("Location Fetched");
        let lat = position.coords.latitude;
        let lon = position.coords.longitude;

        setQuery({
          lat,
          lon,
        });
      });
    }
  }

  const handleUnit = (a) => {
    const selectedUnit = a.currentTarget.name;
    // console.log(selectedUnit);
    // console.log(units);

    if(units !== selectedUnit) setUnits({selectedUnit});

    // console.log(units)
  }

  window.onload = function(){

}

  const handleAQI = () =>{


      // Fetch the AQI data from the API
      fetch(`http://api.openweathermap.org/data/2.5/air_pollution/forecast?lat
=${latitude}&lon=${longitude}&appid=${API_KEY}`)
        .then(response => response.json())
        .then(data => {
          // Set the AQI and SO2 values
          setAqi(data.list[0].main.aqi);

          // Create an object that contains all the pollutant measurements in
the components object
          const componentsObj = {
            co: data.list[0].components.co,
            no: data.list[0].components.no,
            no2: data.list[0].components.no2,
            o3: data.list[0].components.o3,
            so2: data.list[0].components.so2,
            pm2_5: data.list[0].components.pm2_5,
            pm10: data.list[0].components.pm10,
            nh3: data.list[0].components.nh3
          };
          // Set the components object
          setComponents(componentsObj);
          const targetDiv = document.getElementById("aqi-block");
          const btn = document.getElementById("aqi-button");
          btn.onclick = function() {
          if (targetDiv.style.display !== "none") {
            targetDiv.style.display = "none";
          } else {
            targetDiv.style.display = "block";
          }
        };

        })
        .catch(error => console.error(error));
    };



  return (
    <div className='flex flex-row justify-center my-6'>
        <div className='ml-10 flex flex-row w-3/4 items-center justify-center
space-x-2'>
            <input
            value= {city}
            onChange={(e) => setCity(e.currentTarget.value)}
            type="text"
            className="text-md font-light p-2 w-full shadow-xl focus:outline-
none capitalize"
            placeholder='Search for cities..... '
            />
            <UilSearch size={25} className="text-white cursor-pointer
transition ease-out  hover:scale-125"
            onClick={handleSearch} />
            <UilLocationPoint size={25} className="text-white cursor-pointer
transition ease-out hover:scale-125"
            onClick={handleloco}/>
        </div>

        <div className='flex flex-row w-1/4 items-center justify-center'>
            <button name="metric"
            className='ml-2 text-xl text-white font-light hover:scale-110'
            onClick={handleUnit}>
               &deg;C
            </button>
            <p className='text-xl ml-2 mr-2 text-white mx-1'>|</p>
            <button name="imperial"
            className='text-xl text-white font-light hover:scale-110'
            onClick={handleUnit}>
               &deg;F
            </button>
            <p className='text-xl ml-2 mr-2 text-white mx-1'>|</p>

            <button name="aqi" id='aqi-button'
            className='text-xl text-white font-light hover:scale-110'
            onClick={handleAQI}>
               AQI
              <div id='aqi-block' className='absolute mt-3 w-36 flex flex-col
gap-1.5 border p-[5px] border-solid
                backdrop-blur-sm border-[#363636]' >

                <div className='w-full flex flex-row justify-around text-base
p-1'>
                  <span>AQI</span><span>-</span> <span>{aqi}</span>
                </div>

                <div className='w-full flex flex-row justify-around text-base
p-1'>
                  <span>SO<sub>2</sub></span> <span>-
</span><span>{components.so2}</span>
                </div>
                <div className='w-full flex flex-row justify-around text-base
p-1'>
                  <span >PM<sub>2.5</sub></span><span>-</span>
<span>{components.pm2_5}</span>
                </div>
                <div className='w-full flex flex-row justify-around text-base
p-1'>
                  <span>PM<sub>10</sub></span><span>-</span>
<span>{components.pm10}</span>
```

```
                </div>
                <div className='w-full flex flex-row justify-around text-base 
p-1'>
                  <span>CO</span> <span>-</span><span>{components.co}</span>
                </div>

              </div>
            </button>
        </div>

    </div>
  )
}

export default Input
```

3.3 - TempAndDetails.jsx – In this component, all the required information is shown about the city like its temperature, humidity, wind speed, and other information.

```
import React from
'react' import {
UilArrowUp,
    UilArrowDown,
    UilTemperature,
    UilWind,
    UilTear,
    UilSun,
    UilSunset
} from "@iconscout/react-unicons" import { formatLocalTime, iconUrlFromCode
} from '../services/weatherService'


function TempAndDetail({weather:
{     details,     icon,
temp,temp_min,temp_max,
sunrise,sunset,
speed,humidity,
feels_like,timezone
    }  })
{
return (


    <div className='flex flex-col justify-center items-center '>
        <div className='flex items-center justify-center py-1 text-2xl font-
bold tracking-widest text-white'>             <p>{details}</p>
        </div>


        <div className='flex flex-row items-center space-around  text-white w-
9/12 py-3  w-3/5'>            <img
src={iconUrlFromCode(icon)}
alt="icon"              className='w-
22'/>


            <p className='text-5xl mx-2'> {`${temp.toFixed()}`}&deg;</p>
            <div className='flex flex-col space-y-2 items-start'>
                <div className='flex font-light text-sm items-center
justifycenter'>
                    <UilTemperature size={18} className="mr-1" />
                    Real feel :
                    <span className='font-medium ml-
1'>{`${feels_like.toFixed()}`}&deg;</span>
                </div>
                <div className='flex font-light text-sm items-center
justifycenter'>
                    <UilTear size={18} className="mr-1" />
                    Humidity :
                    <span className='font-medium ml-1'>{`${humidity}%`}</span>
</div>
                <div className='flex font-light text-sm items-center
justifycenter'>
                    <UilWind size={18} className="mr-1" />
                    Wind Speed :
                    <span className='font-medium ml-1'>{`${speed}km/hr`}</span>
                </div>
            </div>
        </div>


        <div  className='flex  flex-row  justify-center  items-center  space-x-2
textwhite text-sm py-3'>


            <UilSun />
            <p className='font-light'>
                Rise: <span className='font-medium ml-
1'>{formatLocalTime(sunrise,timezone,"hh:mm a")}</span>
</p>
            <p className='font-light'>|</p>


            <UilSunset />
            <p className='font-light'>
                Set: <span className='font-medium ml-
1'>{formatLocalTime(sunset,timezone,"hh:mm a")}</span>
            </p>
            <p className='font-light'>|</p>
            <UilSun />
            <p className='font-light'>
                High: <span className='font-medium ml-
1'>{`${temp_max.toFixed()}`}&deg;</span>
            </p>
            <p className='font-light'>|</p>
            <UilSun />
            <p className='font-light'>
                Low: <span className='font-medium ml-
1'>{`${temp_min.toFixed()}`}&deg;</span>
            </p>


        </div>


    </div>
  )
}  export default
TempAndDetail
```

<u>3.4 - TimeLocation.jsx</u> – This component is used to show the current time , date, and time format of that city

```
import React from 'react' import { formatLocalTime } from
'../services/weatherService'
 function TimeLocation({weather: {dt,timezone,name,country}})
{      return (
        <div>
            <div className='flex item-center justify-center my-6'>
                <p className='text-white text-xl font-extralight'>
                    {formatLocalTime(dt,timezone)}
                </p>


            </div>
            <div className='flex items-center justify-center my-3'>
                <p className='text-white text-3xl font-medium'>
                    {`${name}, ${country}`}
                </p>
            </div>
        </div>
```

```
    )
}  export default
TimeLocation
```

<u>3.5 - Forecast.jsx</u> – This component is used to get and show both hourly forecasts of the location for the coming 5 hours and forecast of the coming 5 days.

```
import React from 'react' import { iconUrlFromCode } from
'../services/weatherService'
 function Forecast({ title,items })
{   return (
    <div>
        <div className="flex items-center justify-start mt-6">
            <p className="text-white font-medium uppercase">
                {title}
            </p>
        </div>
        <hr className='my-2'></hr>

        <div className="flex items-center justify-between text-white">
           {items.map((item) =>(
            <div className="flex flex-col items-center justify-center">
                <p className="font-light text-sm">
                    {item.title}
                </p>                 <img
className='w-12 my-1'
src={iconUrlFromCode(item.icon)} alt=""
                />
                <p className="font-medium">{`${item.temp.toFixed()}`}&deg;</p>


            </div>
           ))}
        </div>
    </div>
  )
}  export default
Forecast
```

3.6 – App.js

```
import logo from './logo.svg';
import './App.css';
import UilReact from '@iconscout/react-unicons/icons/uil-react'
import TopButtons from './components/TopButtons';
import Input from './components/Input';
import TimeLocation from './components/TimeLocation';
import TempAndDetail from './components/TempAndDetail';
import Forecast from './components/Forecast';
import getFormattedWeatherData from './services/weatherService';
import { useEffect, useState } from 'react';
import { ToastContainer,toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
function App() {

  const [query, setQuery] = useState({ q: 'delhi' });
  const [units, setUnits] = useState('metric');
  const [weather, setWeather] = useState(null);


  useEffect(() => {

    const fetchWeather = async () => {
      const msg = query.q ? query.q : 'current location';
      toast.info('fetchinging weather for '+ msg);
      await getFormattedWeatherData({...query,units }).then(
        (data) => {

          toast.success(`Successfulluy fetehed for ${data.name} ,
${data.country}`);
          console.log(data);
          setWeather(data);
        }
      );
    };

    fetchWeather();

  }, [query, units])


  const bgChange = () => {
    if (!weather) return 'bg-main';
    if(weather.temp< 5 )
      return 'bg-snow';
    if(weather.temp>25)
      return 'bg-sunny';
    return 'bg-haze';

  }
  return (
    <div className={`w-full h-full ${bgChange()} bg-cover , bg-no-repeat py-
5`}>
      <div className='mx-auto  w-3/5 py-5 px-32
      bg-white bg-opacity-20
      backdrop-blur-xl rounded drop-shadow-lg
      bg-slate-600' >
        <TopButtons setQuery={setQuery}/>
        <Input setQuery={setQuery} units={units} setUnits={setUnits} />
        {weather && (
          <div>
            <TimeLocation weather={weather} />
            <TempAndDetail weather={weather} />

            <Forecast title="hourly forecast" items={weather.hourly}/>
            <Forecast title="daily forecast" items={weather.daily}/>
          </div>
        )}
      </div>

    <ToastContainer autoClose={3000} theme="colored" newestOnTop={true}/>
    </div>
  );
}

export default App;
```

3.7 – weatherService.js – it contain javascript code that is used to make every API calls

```
import { DateTime } from "luxon";
import { useState } from "react";

const API_KEY = "d8ab493aa80cf34ae8123f2490d4887b";
const Base_URL = "https://api.openweathermap.org/data/2.5";
let latitude,longitude;

const getWeatherData = (infoType , searchPara) =>
{
    const url = new URL(Base_URL + "/" + infoType);

    url.search = new URLSearchParams({...searchPara,appid:API_KEY}
        );
   // console.log(url);
    return fetch(url).then((res) => res.json())
};

const formatCurrentWeather = (data) =>
{
    const {
        coord: {lon,lat},
        main:{uvi,temp, feels_like , temp_min , temp_max , humidity},
        name,
        dt,
        sys : {country , sunrise , sunset},
        weather,
        wind:{speed}
    } = data
    const {main:details , icon} = weather[0]

    return {lat,lon,temp,feels_like,temp_min,temp_max,uvi
    ,humidity,name,dt,country,sunrise,sunset,weather,speed,details,icon}
}
```

```
const formatForecastWeather = (data) => {
    let { timezone, daily , hourly } = data;
    daily = daily.slice(1,6).map(d => {
        return {
            title: formatLocalTime(d.dt,timezone,'ccc'),
            temp: d.temp.day,
            icon:d.weather[0].icon
        }
    });

    hourly = hourly.slice(1,6).map(d => {
        return {
            title: formatLocalTime(d.dt,timezone,'hh:mm a'),
            temp: d.temp,
            icon:d.weather[0].icon
        }
    });

    return { timezone , hourly , daily};
};

const formatLocalTime = (secs , zone , format = "cccc, dd LLL yyyy' | Local
time: 'hh:mm a") =>DateTime.fromSeconds(secs).setZone(zone).toFormat(format);

const getFormattedWeatherData = async (searchPara) => {
    const formattedCurrentWeather = await getWeatherData("weather",searchPara)
    .then(formatCurrentWeather)

    const {lat,lon} = formattedCurrentWeather;
    latitude=lat;
    longitude=lon;
    const formattedForecastWeather = await getWeatherData('onecall',{
        lat , lon, exclude: 'current,minutely,alerts',
        units: searchPara.units
    }).then(formatForecastWeather);

  return {...formattedCurrentWeather, ...formattedForecastWeather};
};

const iconUrlFromCode = (code) =>
`http://openweathermap.org/img/wn/${code}@2x.png`;



export {formatLocalTime, iconUrlFromCode};
export default getFormattedWeatherData;
export {longitude,latitude,API_KEY} ;
```

3.8 – index.js – main index file of the react application

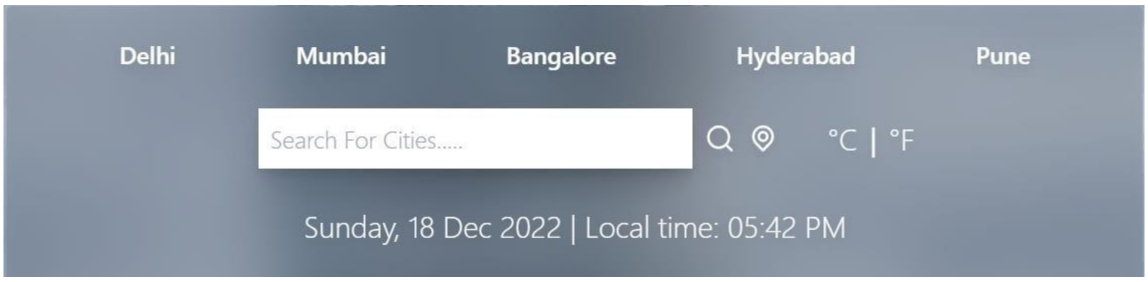
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <App />
);
```

4.Result

4.1 - Upper section –

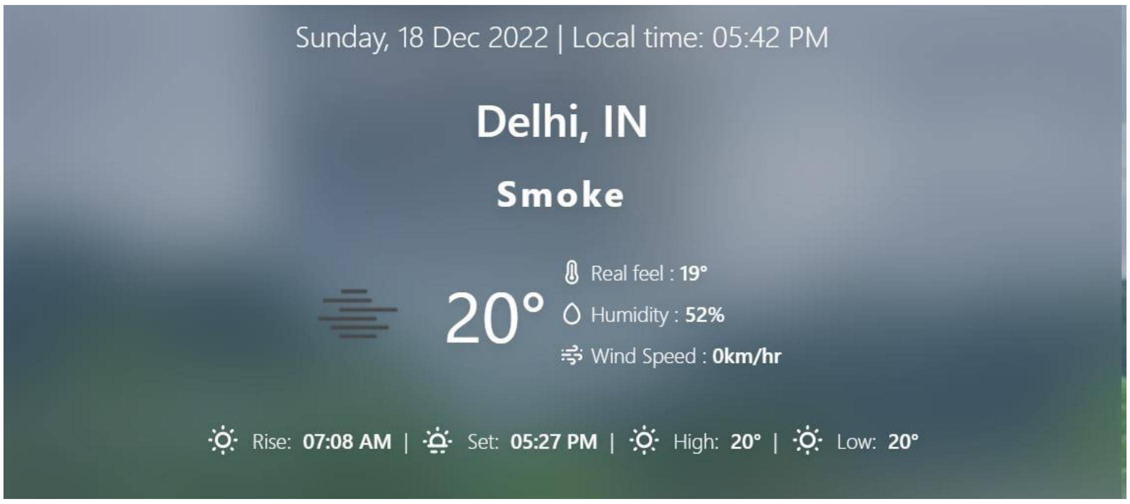
- City block - It shows some major cities of India – in future, we will try to replace it with recent searches
- Search bar – to search city and location icon to search the current location
- Convert Icon – to fetch results either in standard or imperial value • Basic Info - Date and time zone information



Sunday, 18 Dec 2022 | Local time: 05:42 PM

4.2 - Middle section –

- City info – shows city name and its conditions like rainy or sunny
- Temperature – temperature along with some additional information like humidty , Sunset , sunrise etc



Sunday, 18 Dec 2022 | Local time: 05:42 PM

Delhi, IN

Smoke

20°

Real feel : 19°

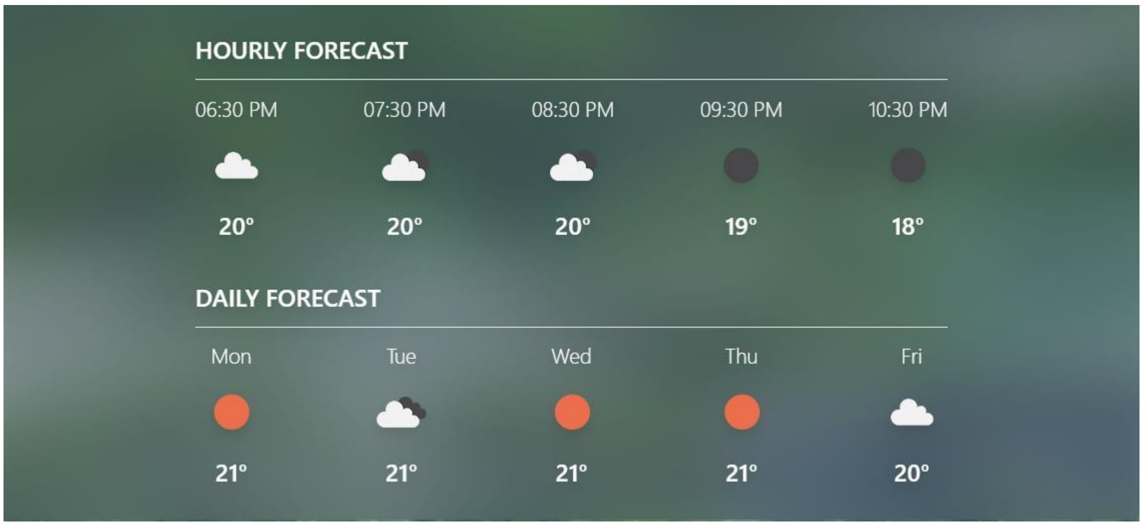
Humidity : 52%

Wind Speed : 0km/hr

Rise: 07:08 AM | Set: 05:27 PM | High: 20° | Low: 20°

4.3 - Lower Section –

- <u>Hourly Forecast</u> – it shows weather info in 1 hour interval upto 5 hours
- <u>Daily forecast</u> – it shows weather info for coming 5 days



HOURLY FORECAST

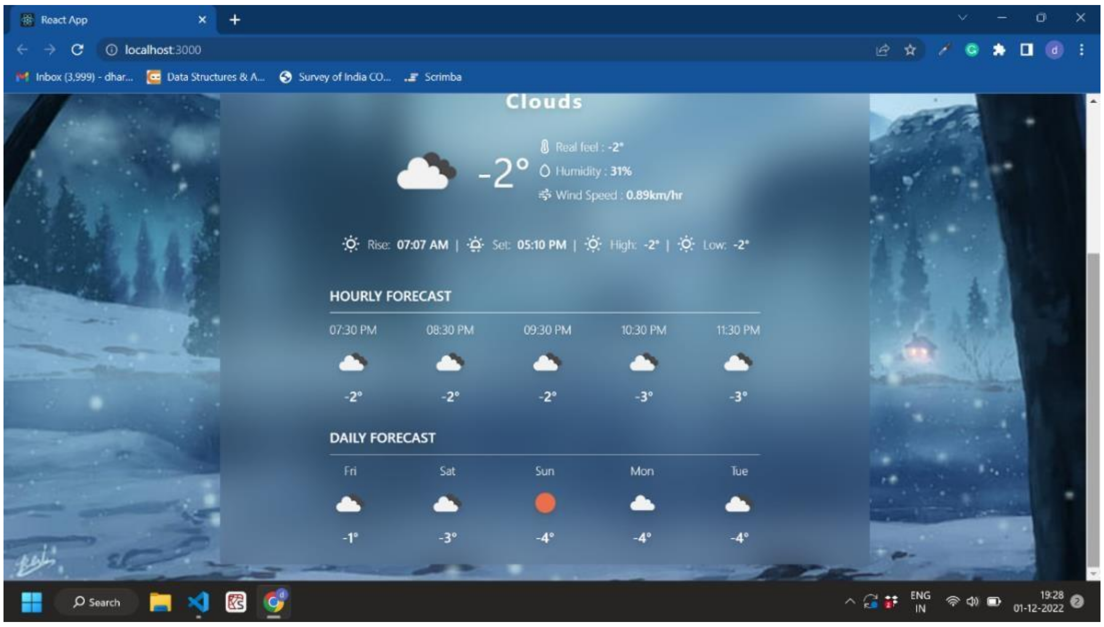
06:30 PM	07:30 PM	08:30 PM	09:30 PM	10:30 PM
20°	20°	20°	19°	18°

DAILY FORECAST

Mon	Tue	Wed	Thu	Fri
21°	21°	21°	21°	20°

4.4 - Dynamic Background –

Change background according to the conditions of that city like rainy or cloudy



Clouds

-2°

Real feel : -2°
Humidity : 31%
Wind Speed : 0.89km/hr

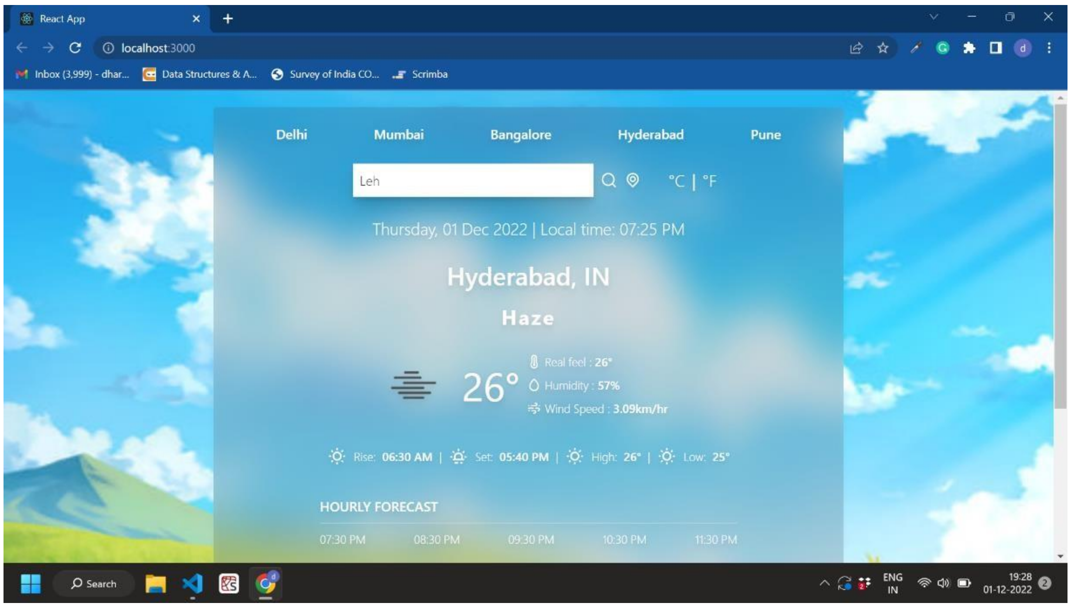
Rise: 07:07 AM | Set: 05:10 PM | High: -2° | Low: -2°

HOURLY FORECAST

```
07:30 PM   08:30 PM   09:30 PM   10:30 PM   11:30 PM
  -2°        -2°        -2°        -3°        -3°
```

DAILY FORECAST

```
Fri   Sat   Sun   Mon   Tue
-1°   -3°   -4°   -4°   -4°
```



Leh

Thursday, 01 Dec 2022 | Local time: 07:25 PM

Hyderabad, IN

Haze

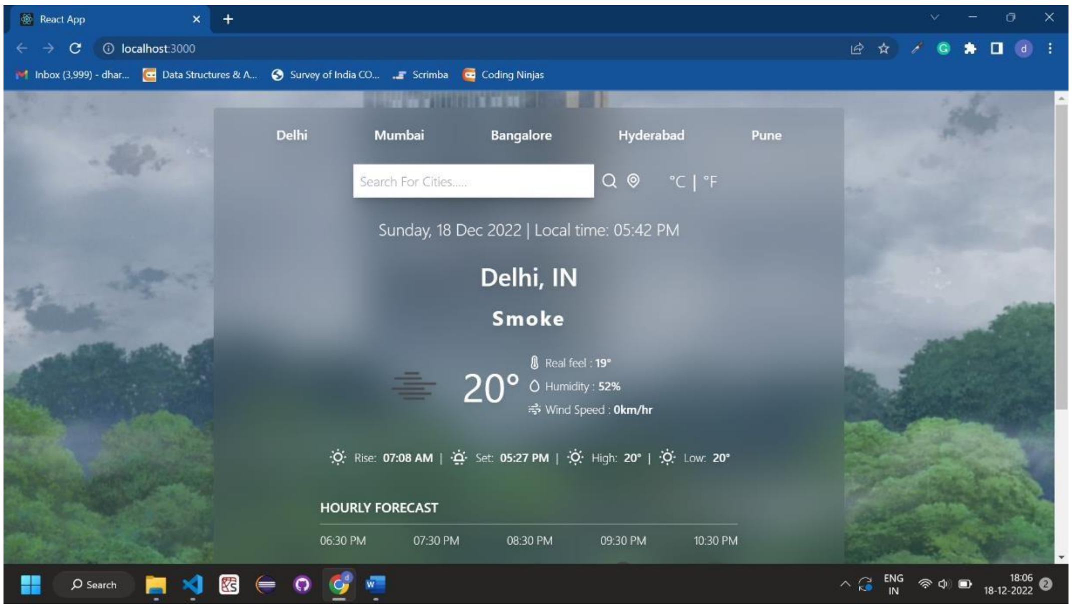
26°

Real feel : 26°
Humidity : 57%
Wind Speed : 3.09km/hr

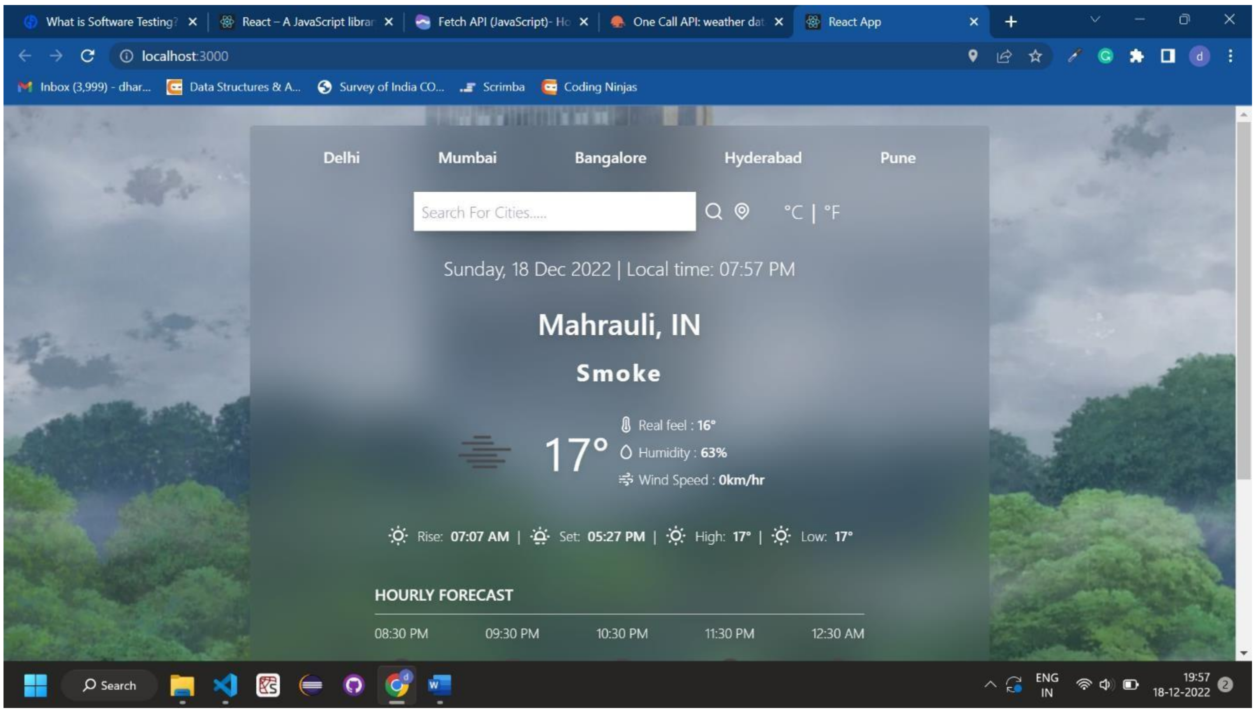
Rise: 06:30 AM | Set: 05:40 PM | High: 26° | Low: 25°

HOURLY FORECAST

```
07:30 PM   08:30 PM   09:30 PM   10:30 PM   11:30 PM
```



4.6 - Current Location


Delhi Mumbai Bangalore Hyderabad Pune

Search For Cities..... °C | °F

Sunday, 18 Dec 2022 | Local time: 07:57 PM

Mahrauli, IN

Smoke

17°

Real feel : 16°

Humidity : 63%

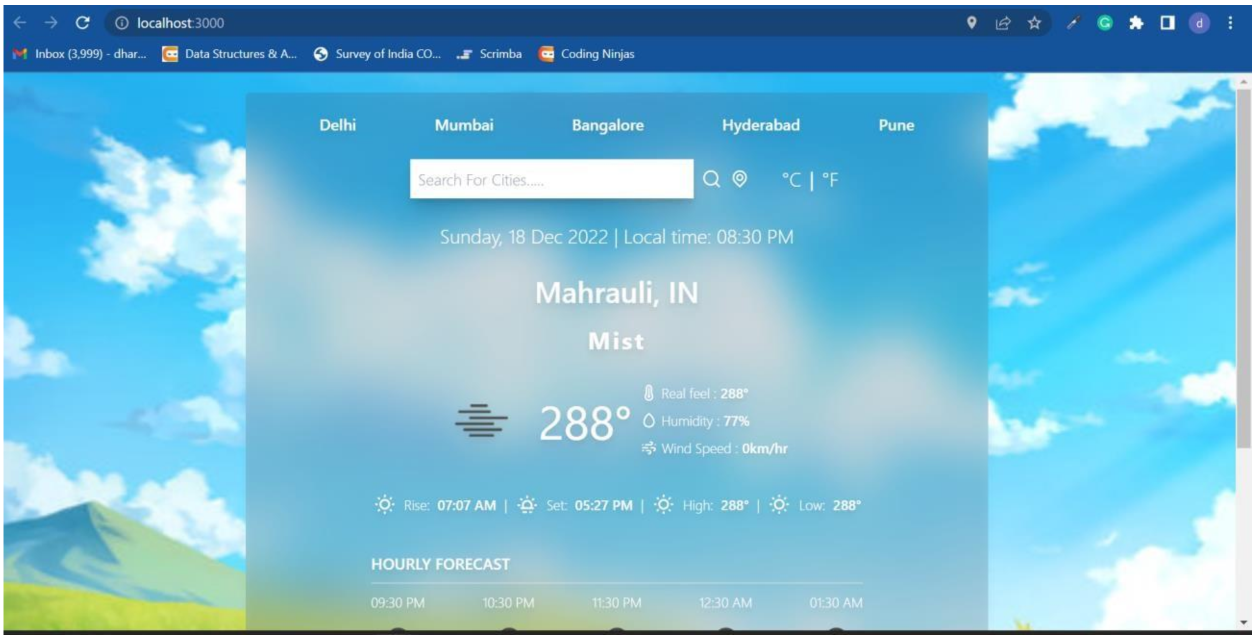
Wind Speed : 0km/hr

Rise: 07:07 AM | Set: 05:27 PM | High: 17° | Low: 17°

HOURLY FORECAST

08:30 PM 09:30 PM 10:30 PM 11:30 PM 12:30 AM

4.7 - Imperial Call


Delhi Mumbai Bangalore Hyderabad Pune

Search For Cities..... °C | °F

Sunday, 18 Dec 2022 | Local time: 08:30 PM

Mahrauli, IN

Mist

288°

Real feel : 288°

Humidity : 77%

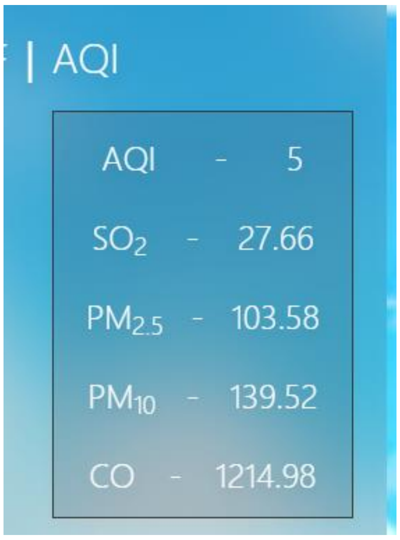
Wind Speed : 0km/hr

Rise: 07:07 AM | Set: 05:27 PM | High: 288° | Low: 288°

HOURLY FORECAST

09:30 PM 10:30 PM 11:30 PM 12:30 AM 01:30 AM

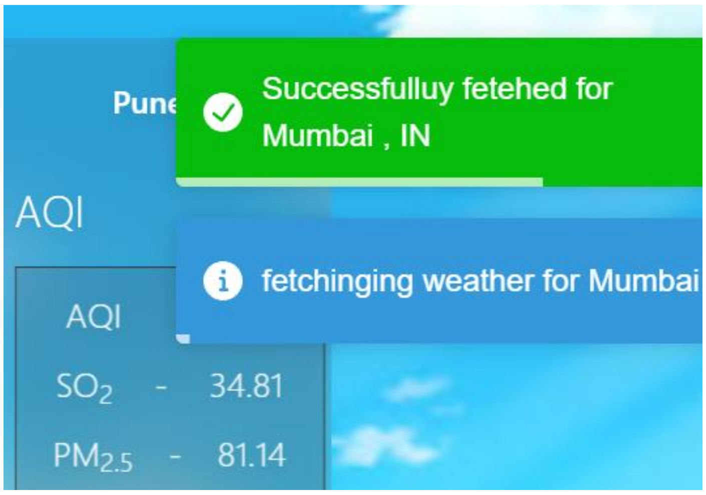
4.8 – AQI



| AQI

AQI	-	5
SO_2	-	27.66
$PM_{2.5}$	-	103.58
PM_{10}	-	139.52
CO	-	1214.98

4.9 – Togglefy



Successfulluy fetehed for Mumbai , IN

fetchinging weather for Mumbai

5. Conclusion and Future Prospect

5.1 Conclusion

In conclusion, the developed weather application stands out as a comprehensive and user-friendly solution with several notable features:

- ✓ Extensive Weather Details: The application provides users with a wide range of weather information, including UV index, sunrise and sunset times, wind speed, humidity, and more. This ensures users have access to comprehensive and accurate data for their location of interest.
- ✓ Unit Conversion Convenience: The ability to toggle between standard and metric units caters to a global audience, allowing users to view weather data in their preferred measurement system.
- ✓ Forecasting and Planning: The inclusion of a 7-day forecast empowers users to plan ahead effectively, whether it's for personal activities, outdoor events, or travel arrangements.
- ✓ Engaging Visual Experience: The dynamic background feature, changing based on the temperature of the searched location, adds a visually captivating element to the application, enhancing user engagement and immersion.

The weather application's scalability potential, economic viability, and contribution to environmental sustainability through promoting eco-conscious behaviors make it a valuable and relevant solution in today's market. It addresses the increasing demand for reliable weather information while offering a user-friendly and feature-rich experience for individuals and businesses alike.

5.2 Future Scope

The weather application has a promising future scope with opportunities for further growth and enhancement. Some potential areas of expansion and development include:

- Integration with Smart Devices: The application can be integrated with smart devices such as smartwatches, voice assistants, and IoT devices, allowing users to access weather information seamlessly from their connected devices.
- Personalization and Customization: Adding personalized features like user-defined preferences, customizable dashboards, and personalized notifications can enhance the user experience and provide tailored weather information based on individual needs.
- Advanced Weather Analytics: Incorporating advanced weather analytics and data visualization techniques can provide users

with deeper insights into weather patterns, trends, and historical data, empowering them to make informed decisions.

- Social and Community Features: Introducing social and community features, such as user-generated content, weather-related discussions, and sharing options, can foster a sense of community and engagement among users.
- Expansion to New Markets: The application can explore expanding to new geographical markets, offering localized weather information, language support, and region-specific features to cater to a broader user base.

Overall, the future scope of the weather application is promising, with ample opportunities for innovation, expansion, and delivering an enhanced user experience in the dynamic and evolving field of weather forecasting and information.

6. References

- https://www.geeksforgeeks.org/reactjs-tutorials/Freecodecamp.org - for learning about react and its syntax
- https://scrimba.com/learn/learnreact – Scrimba for learning more implementation of React and Javascript
- https://reactjs.org/docs/getting-started.html
    - React official documentation – for understanding reactjs
- https://nodejs.org/en/docs/ - Nodejs documentation for API reference documentation and about node
- https://nodejs.org/en/ - to install and learn the implementation of nodejs
- https://tailwindcss.com/docs/installation - to learn about Tailwind and how to install and use it
- https://tailwindcss.com/docs/installation - to find out the tailwind syntax for the required normal CSS property
- https://openweathermap.org/api/one-call-api - to learn about how to use and access the OpenWeatherMap Onecall API
- https://www.npmjs.com/package/luxon - for understanding luxon package
- https://www.w3schools.com/jsref/api_fetch.asp - for understand about the javascript fetch API

- https://openweathermap.org/api - to get the information the AQI api
- https://unschooler.me/tutorials/http-understanding-12582- to learn more about the API
- https://www.youtube.com/watch?v=FAnuh0_BU4c&embeds_euri=https%3A%2F%2Funschooler.me%2F&embeds_origin=https%3A%2F%2Funschooler.me&source_ve_path=OTY3MTQ&feature=emb_imp_woyt – understanding about the API
- https://www.youtube.com/watch?v=I75zkl37JcQ&embeds_euri=https%3A%2F%2Funschooler.me%2F&embeds_origin=https%3A%2F%2Funschooler.me&source_ve_path=OTY3MTQ&feature=emb_imp_woyt – to learn about the REST API