WEATHERING APPLICATION USING REACT JS

Prepared under the Guidance of

KnackForge Soft Solutions Private Limited

Chennai

By

Shalini K(513321104040)

B.E.Computer Science and Engineering

University College of Engineering, Arni

Submitted on

August 2024

DECLARATION

I, **SHALINI K** hereby declare that the project report entitled **WEATHER-APP using React-JS** done by me under the guidance of **SANGEETHA** is submitted in partial fulfillment of the requirements for the award of Internship in Front-end Development.

DATE:

PLACE: **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **KNACKFORGE** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I would like to express my sincere and deep sense of gratitude to my Project Guide **SANGEETHA** for her valuable guidance, suggestions, and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to staff members of the **KNACKFORGE** who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE

TABLE OF CONTENT

ABSTRACT

Weather is the state of the atmosphere at a given place and time, encompassing factors such as temperature, cloudiness, dryness, sunshine, wind, and rain. Among all geophysical phenomena, weather is the most significant in its impact on our daily lives.

Weather can vary greatly depending on climate, seasons, and various other factors. The primary objective of this work is to develop an application that provides the weather forecast for any city worldwide.

This paper focuses on creating a web application using the JavaScript framework React-JS, which includes features such as temperature, location, precipitation, wind speed, and humidity. The report offers a foundational understanding of the purpose and scope of weather forecasting, the basic principles involved, and the general models developed for predicting weather.

CHAPTER 1

INTRODUCTION

1.1 Objective of the Project

The primary objective of this project is to develop a web application that accurately displays the current state of the atmosphere.

1.1.1 Necessity

This website facilitates the understanding of weather conditions in various locations. It is user-friendly, operates smoothly, and provides accurate information. The application allows users to input a specific location to receive current weather conditions, aiding in travel and daily planning. Users can also check weather conditions in different zones and share this information with others. The application provides insights into temperature, humidity, and wind speed, helping users stay informed about changing weather patterns.

1.1.2 Layout of the Document

This documentation begins with a formal introduction. Following the introduction, the analysis and design of the project are discussed. The analysis and design sections cover various aspects such as project proposal, mission, goals, target audience, and environment. Subsequently, design details and table diagrams are presented. Use cases and test cases are elaborated in Chapter 2 and Chapter 3, respectively. The documentation concludes with the results and conclusion section.

1.2 Web Design Overview

The design process for the React-based website begins with a visual concept, incorporating a Higher-Order Component (HOC) that enhances another component by adding extra functionalities or properties. The design pattern includes stateless functions, render props, controlled components, conditional rendering, and React hooks. This overview covers commonly used logics and design patterns to achieve the desired website functionality.

CHAPTER 2

AIM AND SCOPE OF THE PRESENT INVESTIGATION

2.1 Project Proposal

The project proposal outlines the comprehensive plans and requirements for the project. It includes descriptions of the software functionality, steps for project completion, software requirements, and analysis. Additionally, the proposal addresses risk management, reward potential, and other project dependencies.

2.1.1 Mission

The mission of this project is to create an online web-based application that is accessible and user-friendly. This application is designed to provide weather information globally, making it convenient for users to obtain weather details for any city. By simply entering the name of a city, users can access information such as temperature, humidity, wind speed, and location. This web app aims to assist users in understanding weather conditions worldwide.

2.1.2 Goal

The primary goal of this project is to develop a web-based application that provides detailed weather information for specific locations. This application will help users stay informed about the climate and make informed decisions based on real-time weather data.

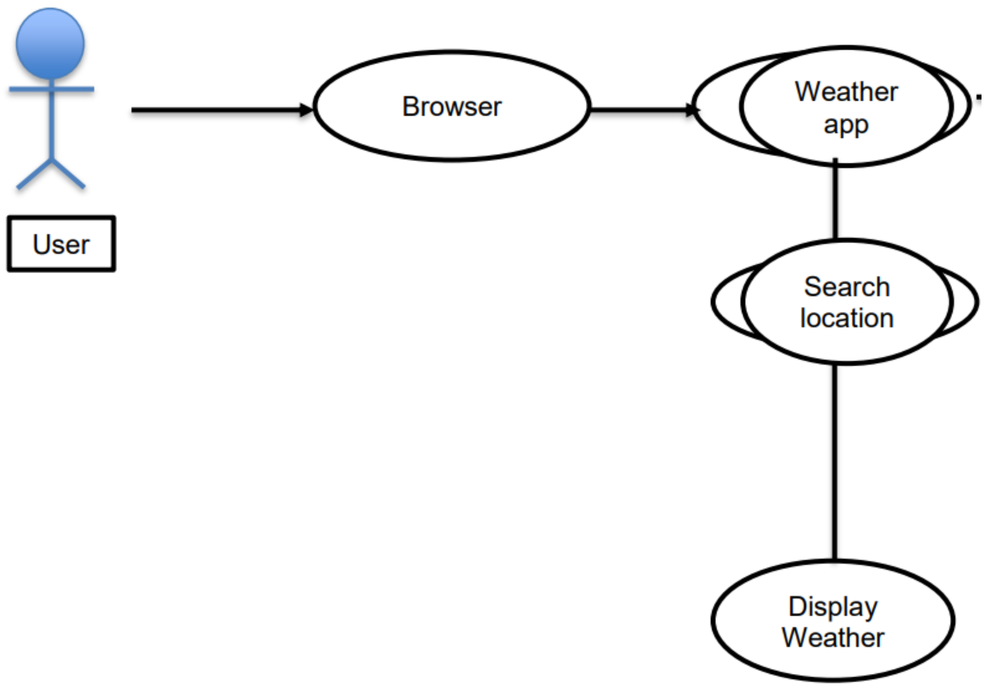
2.2 Scope of the Project

The scope of this project is to build a web-based application that is easily accessible to everyone in society. It will operate on a common server or browser, ensuring familiarity and ease of use. Customizations such as fonts and colors will not be alterable by users, allowing the focus to remain on the application's features and functionality. The goal is to create an application that supports effective weather management and provides valuable information to the community.

2.3 Overview of the Project

The project aims to provide a web-based application that users can easily access from any device and any location. By entering the name of a place, users can quickly obtain weather information, making the application a convenient tool for understanding global weather conditions.

2.4 FLOWCHART



CHAPTER 3
EXPERIMENTAL OR MATERIAL AND METHODS

3.1 System Study:

To develop this model we use new modern technology like React, JavaScript for overcoming our problem statement.

3.1.1 System requirement specifications:

Hardware requirements:

▪ Processor	:	Intel Pentium D 2.8 GHZ.
▪ RAM	:	6GB
▪ Hard Disk	:	80GB

Software requirements:

▪ OS	:	Windows.
▪ Framework	:	React, JavaScript.
▪ Web Browser	:	Chrome.
▪ Code editor	:	Visual Studio Code.

3.2 System Specifications:

JavaScript Overview:

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. JavaScript is a lightweight, interpreted programming language. JavaScript is the most popular programming language in the world and that makes it a programmer’s great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end software’s using different Javascript based frameworks like React, Node.JS etc. Javascript helps in manipulating

HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.

React Overview:

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It's 'V' in MVC. ReactJS is an open-source, component-based front-end library responsible only for the view layer of the application. It is maintained by Facebook.

React uses a declarative paradigm that makes it easier to reason about your application and aims to be both efficient and flexible. It designs simple views for each state in your application, and React will efficiently update and render just the right component when your data changes. The declarative view makes your code more predictable and easier to debug.

A React application is made of multiple components, each responsible for rendering a small, reusable piece of HTML. Components can be nested within other components to allow complex applications to be built out of simple building blocks. A component may also maintain an internal state – for example, a Tab List component may store a variable corresponding to the currently open tab.

3.3 Steps to Download & Install ReactJS:

First of all, you are going to need NPM (or Yarn, alternatively). Let's use NPM for this example.

If you don't have it installed on your system, then you need to head to the official Node.js website to download and install Node, which also includes NPM (Node Package Manager).

Select the "Recommended For Most Users" button and download the current version for your operating system.

After you download and install Node, start your terminal/command prompt and run node -v and npm -v to see which versions you have.

Your version of NPM should be at least 5.2.0 or newer because create-reactapp requires that we have NPX installed.

If you have an older version of NPM, run this command to update it:

npm install –g npm

3.3.1 What is create-react-app?

Since it is complicated and takes a lot of time, we don't want to configure React manually. create-react-app is a much easier way which does all the configuration and necessary package installations for us automatically and starts a new React app locally, ready for development.

Another advantage of using create-react-app is that you don't have to deal with Babel or Webpack configurations. All of the necessary configurations will be made by create-react-app for you.

How to Install Create-React-App

In order to install your app, first go to your workspace (desktop or a folder) and run the following command:

npx create-react-app my-app

The installation process may take a few minutes. After it is done, you should see a folder that appears in your workspace with the name you gave to your app.

Note:

If you're on Mac and receiving permission errors, don't forget to be a super user first with the sudo command.

How to Run the App You Created with Create-React-App

After the installation is completed, change to the directory where your app was installed:

cd my-app and finally run **npm start** to see your app live on localhost:

npm start

If you see something REACT logo in your browser, you are ready to work with React.

3.4 Packages to be Installed

- npm install react
- npm install luxon
- npm install tailwind

3.5 OpenWeatherMap API

The OpenWeatherMap API is designed to provide accurate and reliable weather data for a wide range of applications and use cases. Its straightforward API design ensures ease of use.

To begin using the API, you must authenticate with your unique API key from your OpenWeatherMap account. Follow these steps to authenticate and make requests to the API:

Authentication: Use the base URL for OpenWeatherMap's API and pass your unique API key to the appid parameter.

Base URL: http://api.openweathermap.org/data/2.5/weather

The OpenWeatherMap API provides a range of weather data, including temperature, humidity, wind speed, and more. By utilizing this API, you can easily integrate weather information into your application.

3.6 Flex Grid

A grid system based on flex display property.

The grid should be displayed on the app to divided the app in to two parts.

Nav bar and the display bar which contains the information present in data under the console.

The data will be printed accordingly.

CHAPTER 4

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

4.1 Performance Analysis

Website performance optimization is a crucial aspect of technologically advanced web designs, serving as a primary factor for success in meeting modern daily needs. Slow-loading web pages frustrate visitors, driving them to seek alternatives and negatively impacting user engagement and satisfaction.

To highlight the importance of a fast-loading, responsive website, we created a comprehensive speed optimization guide, organized into six detailed chapters. Numerous research papers and benchmarks support the claim that optimizing website speed is one of the most cost-effective investments with high returns on investment (ROI).

A lightning-fast page load speed enhances visitor engagement and retention, leading to increased sales. Instantaneous website response times result in higher conversion rates. According to recent research by the Aberdeen Group, a delay of just one second in page load time can decrease customer satisfaction by 16%, reduce page views by 11%, and lower conversion rates by 7%.

4.2 Discussion

Discussions are valuable for exploration and discovery, though they can be anxiety-inducing due to their unpredictable nature and the necessity for instructors to relinquish some control over the flow of information. Effective discussions require careful planning to ensure they are engaging without being chaotic, and exploratory without losing focus.

When planning a discussion, it's important to consider various factors that can influence the productive exchange of ideas:

- **Cognitive Factors:** These involve the content and structure of the discussion, ensuring that the topics are intellectually stimulating and appropriately challenging.
- **Social/Emotional Factors:** Creating a supportive environment where participants feel comfortable sharing their ideas is essential for fostering open and meaningful discussions.
- **Physical Factors:** The physical setting can impact the flow of discussion, with considerations such as seating arrangements and the availability of necessary resources playing a role.

By addressing these factors, we can create discussions that are both lively and focused, facilitating a productive exchange of ideas and insights.

CHAPTER 5

SUMMARY AND CONCLUSIONS

5.1 Summary

This project objective is to deliver the online website that brings every useful application into one platform. This project is an attempt to provide the advantage of all small application in to one platform that help the user in time management. It helps the user in critical situations.

5.2 Conclusion

This website is helpful for people to identify tasks that are written in todo-list(notepad). So, no one will forget any important messages by seeing them and also it helps in calculating mathematical operations using the calculator and by using the BMI calculator helps in Gym for calculating your BMI directly. So, this Website is useful for time management and it is very easy to use.

Future Work

While the current weather application provides essential features for retrieving and displaying weather data, several enhancements and additional functionalities can be implemented to improve the user experience and extend the application's capabilities. Firstly, enhancing the user interface and experience through responsive design will ensure seamless operation across various devices, including tablets and mobile phones.

Introducing custom themes, such as light and dark modes, and adding subtle animations for transitions and data updates can make the application more visually appealing and engaging. Furthermore, expanding the weather information to include extended forecasts, such as 7-day or 14-day predictions, and hourly weather data would offer users a more comprehensive view of upcoming weather changes. Personalization features, such as allowing users to save their favorite locations for quick access and implementing user accounts for personalized settings, would significantly enhance user convenience.

Additionally, incorporating advanced weather data, including weather alert notifications for severe conditions and displaying the air quality index (AQI) for selected locations, would provide users with critical information to better prepare for and respond to various weather situations. These improvements will not only enhance the application's functionality but also provide a more tailored and informative experience for users.

REFERENCES

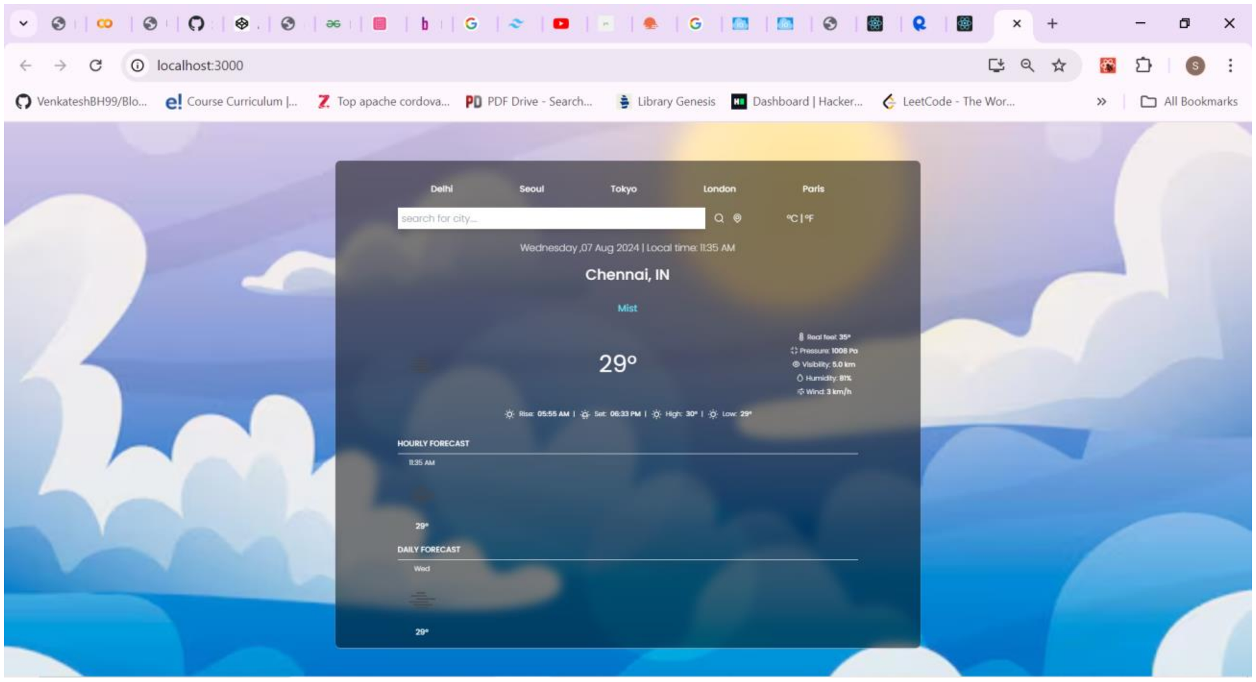
- https://github.com
- https://stackoverflow.com
- https://www.geeksforgeeks.org
- https://openweathermap.org

APPENDIX

A. SCREENSHOTS

5.1 Default Weather Display for Chennai

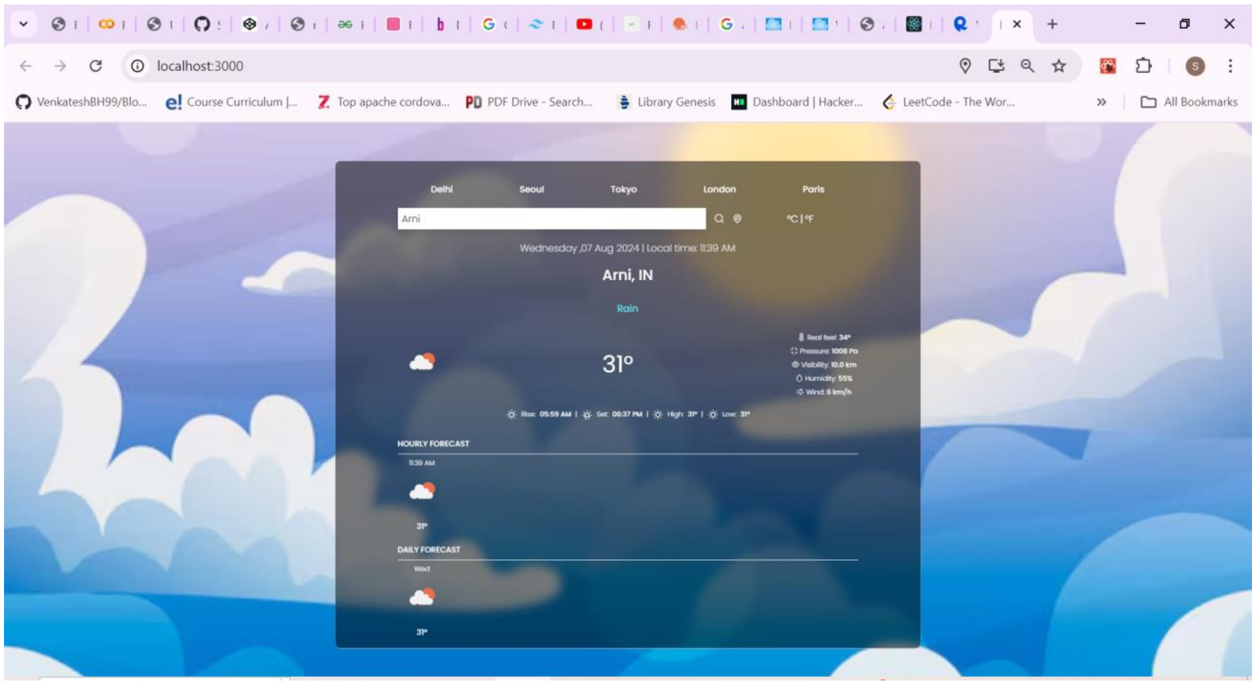
Upon loading the application, the weather data for Chennai is displayed by default. This ensures that users are immediately presented with relevant weather information without needing to perform an initial search.



Upon initialization, the application makes an API call to OpenWeatherMap using the coordinates or city name for Chennai to fetch and display the weather data.

5.2 Weather Search by City Name

Users can search for the weather in any city by entering the city name into the search bar. The application retrieves the relevant weather data for the specified city and displays it on the screen. This feature allows users to obtain weather information for any location worldwide.

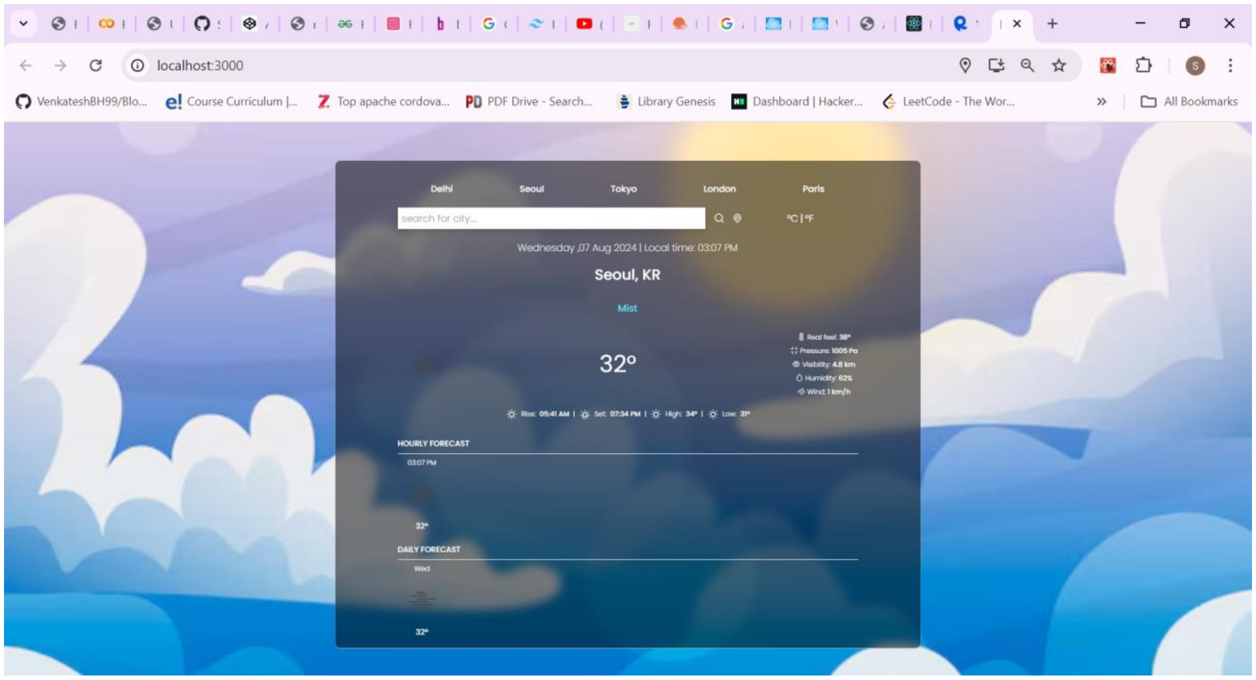


Search Functionality

1. **Input Field:** Users enter the name of a city in the search input field.
2. **API Call:** The application constructs a query URL with the entered city name and sends a request to the OpenWeatherMap API.
3. **Data Display:** Upon receiving the data, the application updates the UI to display the weather information for the searched city.

5.3 Quick Access to Weather Data for Select Cities

The application includes buttons at the top of the interface that provide quick access to weather information for a few predefined cities. By clicking on these buttons, users can instantly view the weather data for those specific locations without needing to perform a search.

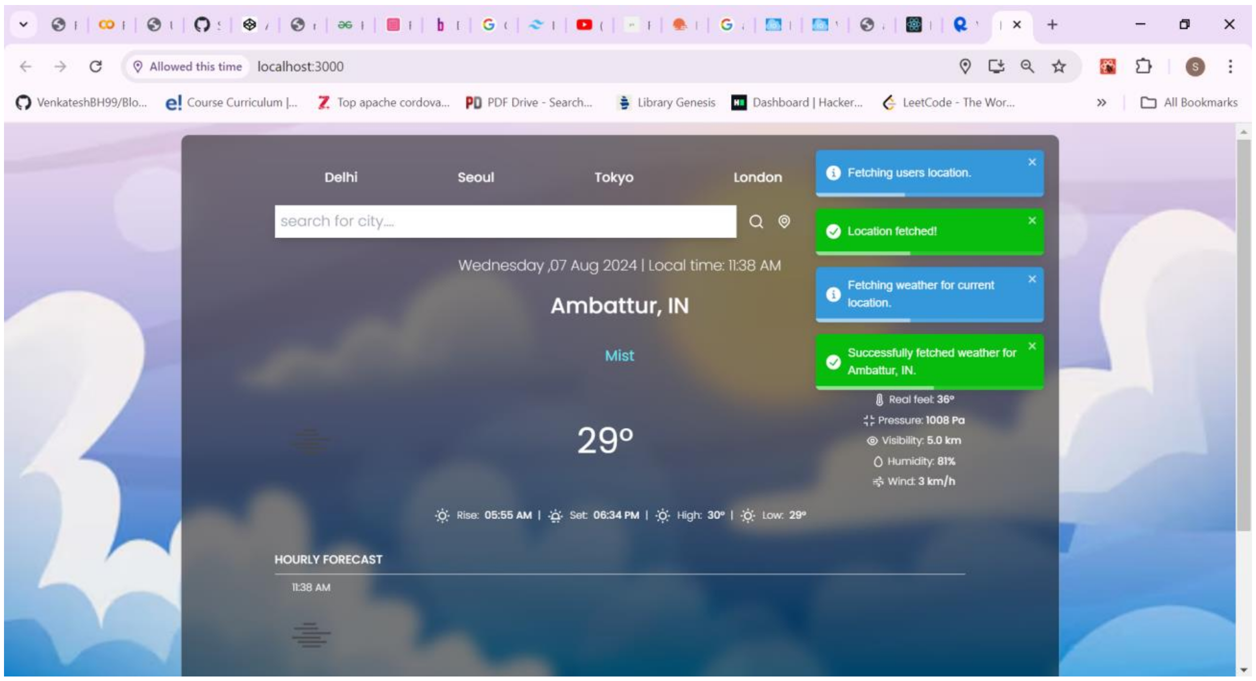


Quick Access Buttons

1. **Predefined Cities:** The application has a set of buttons, each labeled with the name of a predefined city.
2. **Button Click Event:** When a button is clicked, the application triggers an API call to fetch the weather data for the corresponding city.
3. **Data Display:** The weather information for the selected city is displayed on the screen.

5.4 Weather Data for Current Location

The application also allows users to view the weather for their current location. When users grant the application access to their device's location, the app retrieves the current geographical coordinates and fetches the weather data for that location. This feature is particularly useful for users who want to know the weather conditions in their immediate vicinity.



Fetching users location.

Location fetched!

Fetching weather for current location.

Successfully fetched weather for Ambattur, IN.

Wednesday ,07 Aug 2024 | Local time: 11:38 AM

Ambattur, IN

Mist

29°

Real feel: 36°
Pressure: 1008 Pa
Visibility: 5.0 km
Humidity: 81%
Wind: 3 km/h

Rise: 05:55 AM | Set: 06:34 PM | High: 30° | Low: 29°

11:38 AM

Current Location Weather

1. **Location Permission:** When the user opts to see the weather for their current location, the application requests permission to access the device's location.
2. **Geolocation API:** Upon granting permission, the application uses the Geolocation API to obtain the user's current coordinates (latitude and longitude).
3. **API Call:** The coordinates are then used to construct a query URL for the OpenWeatherMap API.
4. **Data Display:** The fetched weather data is displayed, providing the user with real-time weather information for their current location.

B.SOURCE CODE:

Public:

public\index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

src:

src\index.js

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

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

src\index.css

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300
;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,
800;1,900&display=swap');

@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base{
    html{
        font-family:'poppins',sans-serif;
    }
}
```

src\App.js

```
import './App.css';
import TopButtons from './components/TopButtons';
import Inputs from './components/Inputs';
import TimeAndLocation from './components/TimeAndLocation';
import TemperatureAndDetails from './components/TemperatureAndDetails';
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: "chennai" });
  const [units, setUnits] = useState("metric");
  const [weather, setWeather] = useState(null);

  useEffect(() => {
    const fetchWeather = async () => {
      const message = query.q ? query.q : "current location.";

      toast.info("Fetching weather for " + message);

      await getFormattedWeatherData({ ...query, units }).then((data) => {
        toast.success(
          `Successfully fetched weather for ${data.name}, ${data.country}.`
        );

        setWeather(data);
      });
    };

    fetchWeather();
  }, [query, units]);

  return (
    <div className="app-container">
      <div className="content-wrapper mx-auto mt-4 py-5 px-32 h-fit shadow-xl
shadow-gray-400">
        <TopButtons setQuery={setQuery} />
        <Inputs setQuery={setQuery} units={units} setUnits={setUnits} />

        {weather && (
          <div>
            <TimeAndLocation weather={weather} />
            <TemperatureAndDetails weather={weather} />

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

        <ToastContainer autoClose={5000} theme="colored" newestOnTop={true} />
      </div>
    </div>
  );
}

export default App;
```

src\App.css

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

.app-container {
  position: relative;
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
}

.content-wrapper {
  position: relative;
  z-index: 1;
  background-color: rgba(0, 0, 0, 0.5);
  border-radius: 10px;
  padding: 20px;
  width: 100%;
  max-width: 1200px;
  box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0,
0.05);
  backdrop-filter: blur(10px);
}

.app-container::before {
  content: "";
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: url('https://img.freepik.com/free-vector/sky-background-video-
conferencing_23-2148639325.jpg') no-repeat center center/cover; /* Add your
background image URL here */
  z-index: 0;
}

.mx-auto {
  max-width: 1200px;
}

.mt-4 {
  margin-top: 1rem;
}
```

```css
.py-5 {
  padding-top: 1.25rem;
  padding-bottom: 1.25rem;
}

.px-32 {
  padding-left: 8rem;
  padding-right: 8rem;
}

.h-fit {
  height: fit-content;
}

.shadow-xl {
  box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0,
0.05);
}

.shadow-gray-400 {
  --tw-shadow-color: rgba(156, 163, 175, 1);
  box-shadow: 0 1px 2px 0 var(--tw-shadow-color);
}
```

src\components\Forecast.jsx

```jsx
import React from "react";
import { iconUrlFromCode } from "../Services/WeatherService";

function Forecast({ title, items }) {
  let data = [];
  data.push(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" />

      <div className="flex flex-row items-center justify-between text-white">

        {data.map((item, index) => (
          <div
            key={index}
            className="flex flex-col items-center justify-center"
          >
            <p className="font-light text-sm">{item.title}</p>
            <img
              src={iconUrlFromCode(item.icon)}
              className="w-25 my-1"
              alt=""
            />
            <p className="font-medium">{`${item.temp.toFixed()}°`}</p>
          </div>
        ))}
      </div>
    </div>
  );
}

export default Forecast;
```

src\components\Inputs.jsx

```
import React, { useState } from "react";
import { UilSearch, UilLocationPoint } from "@iconscout/react-unicons";
import { toast } from "react-toastify";

function Inputs({ setQuery, units, setUnits }) {
  const [city, setCity] = useState("");

  const handleUnitsChange = (e) => {
    const selectedUnit = e.currentTarget.name;
    if (units !== selectedUnit) setUnits(selectedUnit);
  };

  const handleSearchClick = () => {
    if (city !== "") setQuery({ q: city });
  };

  const handleLocationClick = () => {
    if (navigator.geolocation) {
      toast.info("Fetching users location.");
      navigator.geolocation.getCurrentPosition((position) => {
        toast.success("Location fetched!");
        let lat = position.coords.latitude;
        let lon = position.coords.longitude;

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

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

      <div className="flex flex-row w-1/4 items-center justify-center">
        <button
          name="metric"
          className="text-xl text-white font-light transition ease-out
hover:scale-125"
          onClick={handleUnitsChange}
        >
          °C
        </button>
        <p className="text-xl text-white mx-1">|</p>
        <button
          name="imperial"
          className="text-xl text-white font-light transition ease-out
hover:scale-125"
```

```
          onClick={handleUnitsChange}
        >
          °F
        </button>
      </div>
    </div>
  );
}

export default Inputs;
```

src\components\TemperatureAndDetails.jsx

```
import React from "react";
import {
  UilTemperature,
  UilTear,
  UilWind,
  UilSun,
  UilSunset,
  UilEye,
  UilCompress,
} from "@iconscout/react-unicons";
import { formatToLocalTime, iconUrlFromCode } from "../Services/WeatherService";

function TemperatureAndDetails({
  weather: {
    details,
    icon,
    temp,
    temp_min,
    temp_max,
    sunrise,
    sunset,
    speed,
    humidity,
    pressure,
    visibility,
    feels_like,
    timezone,
  },
}) {
  return (
    <div>
```

```
      <div className="flex items-center justify-center py-6 text-xl text-cyan-
300">
        <p>{details}</p>
      </div>

      <div className="flex flex-row items-center justify-between text-white py-
3">
        <img src={iconUrlFromCode(icon)} alt="" className="w-25" />
        <p className="text-5xl">{`${temp.toFixed()}°`}</p>
        <div className="flex flex-col space-y-2">
          <div className="flex font-light text-sm items-center justify-center">
            <UilTemperature size={18} className="mr-1" />
            Real feel:
            <span className="font-medium ml-
1">{`${feels_like.toFixed()}°`}</span>
          </div>
          <div className="flex font-light text-sm items-center justify-center">
            <UilCompress size={18} className="mr-1" />
            Pressure:
            <span className="font-medium ml-1">{`${pressure} Pa`}</span>
          </div>
          <div className="flex font-light text-sm items-center justify-center">
            <UilEye size={18} className="mr-1" />
            Visibility:
            <span className="font-medium ml-1">{`${(visibility /
1000).toFixed(1)} km`}</span>
          </div>
          <div className="flex font-light text-sm items-center justify-center">
            <UilTear size={18} className="mr-1" />
            Humidity:
            <span className="font-medium ml-1">{`${humidity.toFixed()}%`}</span>
          </div>
          <div className="flex font-light text-sm items-center justify-center">
            <UilWind size={18} className="mr-1" />
            Wind:
            <span className="font-medium ml-1">{`${speed.toFixed()} km/h`}</span>
          </div>
        </div>
      </div>

      <div className="flex flex-row items-center justify-center space-x-2 text-
white text-sm py-3">
        <UilSun />
        <p className="font-light">
          Rise:{" "}
          <span className="font-medium ml-1">
            {formatToLocalTime(sunrise, timezone, "hh:mm a")}
          </span>
        </p>
        <p className="font-light">|</p>

        <UilSunset />
        <p className="font-light">
          Set:{" "}
          <span className="font-medium ml-1">
            {formatToLocalTime(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()}°`}</span>
        </p>
        <p className="font-light">|</p>

        <UilSun />
        <p className="font-light">
          Low:{" "}
          <span className="font-medium ml-1">{`${temp_min.toFixed()}°`}</span>
        </p>
      </div>
    </div>
  );
}

export default TemperatureAndDetails;
```

src\components\TimeAndLocation.jsx

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300
;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,
800;1,900&display=swap');

@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
@layer base{
    html{
        font-family:'poppins',sans-serif;
    }
}
```

src\components\TopButtons.jsx

```
import React from "react";

function TopButtons({ setQuery }) {
  const cities = [
    {
      id: 1,
      title: "Delhi",
    },
    {
      id: 2,
      title: "Seoul",
    },
    {
      id: 3,
      title: "Tokyo",
    },
    {
      id: 4,
      title: "London",
    },
    {
      id: 5,
      title: "Paris",
    },
  ];

  return (
    <div className="flex items-center justify-around my-6">
      {cities.map((city) => (
        <button
          key={city.id}
          className="text-white text-lg font-medium"
          onClick={() => setQuery({ q: city.title })}
        >
          {city.title}
        </button>
      ))}
    </div>
  );
}

export default TopButtons;
```

src\Services\WeatherService.js

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

const API_KEY = "5b549cc0bc75da52a689b3aa7d5282b0"
const BASE_URL = "https://api.openweathermap.org/data/2.5/"

const getWeatherData = (infoType, searchParams) => {
  const url = new URL(BASE_URL + infoType);
  url.search = new URLSearchParams({ ...searchParams, appid: API_KEY });

  return fetch(url).then((res) => res.json());
};

const formatCurrentWeather = (data) => {
  const {
    coord: { lat, lon },
    main: { temp, feels_like, temp_min, temp_max, humidity, pressure },
    visibility,
    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,
    humidity,
    pressure,
    visibility,
    name,
    dt,
    country,
    sunrise,
    sunset,
    details,
    icon,
    speed,
  };
};

const formatForecastWeather = (data) => {
  console.log('---->',data)
  let { timezone, main, dt, weather } = data;
  let daily = {
      title: formatToLocalTime(dt, timezone, "ccc"),
      temp: main.temp,
      icon: weather[0].icon,
    };

  let hourly = {
      title: formatToLocalTime(dt, timezone, "hh:mm a"),
      temp: main.temp,
      icon: weather[0].icon,
    };

  return { timezone, daily, hourly };
};

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

  const { lat, lon } = formattedCurrentWeather;

  const formattedForecastWeather = await getWeatherData("weather", {
    lat,
    lon,
    exclude: "current,minutely,alerts",
    units: searchParams.units,
  }).then(formatForecastWeather);

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

```
const formatToLocalTime = (
  secs,
  zone,
  format = "cccc ,dd MMM yyyy' | Local time: 'hh:mm a"
) =>{

  let d = DateTime.fromSeconds(secs).setZone(zone/60).toFormat(format);
  console.log(secs,zone,format,DateTime.fromSeconds(secs), d);
  return d
}
console.log("formatToLocalTime: ", formatToLocalTime);

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

export default getFormattedWeatherData;

export { formatToLocalTime, iconUrlFromCode };
```