

**PROJECT REPORT ON**

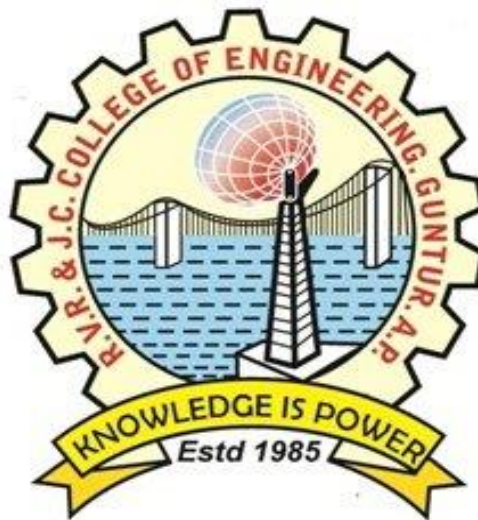
# **WEATHER FORECASTING WEBSITE**

**Submitted under partial fulfillment of the requirements to**

**IT 353 – PROJECT 1**

By

N. SRI HARSHA (Y21IT087)  
P. MANI KANTA SWAMY (Y21IT102)  
SK. ALTHAF ALI MUBARAK (Y21IT109)  
T. SAI KUMAR REDDY (Y21IT122)



**OCTOBER 2023**

**Department of Information Technology**

**R.V.R. & J.C. College of Engineering**

**(Affiliated to Acharya Nagarjuna University, Guntur)**

# **R.V.R. & J.C. COLLEGE OF ENGINEERING**

## **DEPARTMENT OF INFORMATION TECHNOLOGY**

### **BONAFIDE CERTIFICATE**

This is to certify that this project work titled WEATHER FORECASTING WEBSITE is the bonafide work of (N. Sri Harsha (Y21IT087), P. Mani Kanta Reddy (Y21IT102), Sk. Althaf Ali Mubarak (Y21IT109), T. Sai Kuma Reddy (Y21IT122)) who have carried out the work under my supervision, and submitted in partial fulfillment of the requirements to **IT-353, PROJECT 1** during the year 2023-2024.

**Dr. A. Yaswanth Kumar**

Associate Professor

**Dr. A. Srikrishna**

Prof. & HOD, Dept. of IT

## ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without proper suggestions, guidance, and environment. The combination of these three factors acts as the backbone of our Project “**WEATHER FORECASTING WEBSITE**”.

We would like to express my gratitude to the Management of **R.V.R. & J.C. COLLEGE OF ENGINEERING** for providing me with a pleasant environment and excellent lab facility.

We are very much thankful to our Principal, **Dr. Kolla Srinivas**, for providing support and a stimulating environment.

We express our sincere thanks to **Dr. A. Srikrishna**, Professor, and Head of the Department of Information Technology, for her valuable suggestions during the mini project.

We are very glad to express our special thanks to our guide **Dr. A. Yaswanth Kumar**, Associate Professor in the Department of Information Technology for his timely, guidance and for providing us with the most essential materials required for the completion of this project.

Finally, we express our sincere thanks to all the **Teaching** and **Non-Teaching** staff of the **IT Department** who have contributed to the successful completion of this report.

N. SRI HARSHA (Y21IT087)

P. MANI KANTA SWAMY (Y21IT102)

SK. ALTHAF ALI MUBARAK (Y21IT109)

T. SAI KUMAR REDDY (Y21IT122)

## ABSTRACT

Weather forecasting is always needed by a person daily, for example, you would already be aware of a rainstorm hours before it would arrive. So, to display weather forecasts on my website, I would connect server data i.e., an **API** (An application program interface (**API**) is a set of routines, protocols, and tools for building software applications. An **API** specifies how software components should interact. Additionally, **APIs** are used when programming graphical user interface (GUI) components) through which I would access its data and connect it with my website through my **API** key. I would use open weather **API** for this project through which I will access all its real-time weather data and display it on my web page using Javascript, CSS, and HTML.

# TABLE OF CONTENTS

<b>Chapter No. &amp; Name</b>	<b>Page No.</b>
<b>1. Introduction</b>	<b>1</b>
1.1 Outline of the Project	2
1.2 Literature Review	2
1.3 Problem Statement	3
1.4 Objectives	3
<b>2. Aim &amp; Scope</b>	<b>4</b>
2.1 Requirements	4
2.1.1 Hardware Requirements	
2.1.2 Software Requirements	
2.2 Role of Editor	4
2.3 Languages Used	5
2.3.1 HTML	5
2.3.2 CSS	5
2.3.3 JavaScript	5
2.4 Open Weather API	6
2.4.1 Current Weather Data	6
2.4.2 Forecast	6
2.4.3 Searching	6
2.4.4 Maps	7
<b>3. UML Diagrams</b>	
3.1 Use Case Diagrams	8
3.1.1 Identification of Actors, Use Cases, Relations	8
3.1.2 Construction of Use Case Diagram, Flow of Events	12
3.2 Activity Diagram & its Construction	15
3.3 Sequence Diagram & its Construction	17
3.4 Collaboration Diagram & its Construction	19
3.5 Class Diagram & its Construction	20

<b>4.</b>	<b>Methods &amp; Material Used</b>	25
	4.1 Designing of Text Editor	25
	4.2 Connection of API	32
<b>5.</b>	<b>Result &amp; Discussion</b>	34
<b>6.</b>	<b>Conclusion and Future Work</b>	37
	6.1 Summary	
	6.2 Future Work	
	<b>References</b>	38

## LIST OF FIGURES

<b>Figure Name</b>	<b>Page No.</b>
Outline of the Project	25
Today's Forecasting Page	26
Next 4 Days Forecast	27
Styling of the Web page	27
CSS using id and classes	28
Apply styles to more than one element	28
CSS styling using classes	29
Including API Key in JS file	29
Hourly Forecast function	30
Daily Forecast function	31
Conversion of Temperature	32
Open Weather Map Website	32
Open Weather Map Signup page	33
API Key	33
Weather Application	34
Weather in Guntur	34
Weather in Delhi	35
Weather in Washington, US	35
Search for the wrong location	36

## LIST OF ABBREVIATIONS

### ABBREVIATION

HTML

CSS

JS

API

### EXPANSION

Hyper Text Mark Up Language

Cascading Style Sheet

JavaScript

Application Program Interface



# CHAPTER 1

## INTRODUCTION

**Hypertext Markup Language (HTML)** is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items. HTML elements are delimited by tags, and written using angle brackets. Tags such as `<img />` and `<input />` directly introduce content to the page. Other tags such as `<p>` surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. The inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

## API

In computer programming, an **application programming interface (API)** is a set of subroutine definitions, communication protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication among various components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.

An API may be for a web-based system, operating system, database system, computer hardware, or software library.

An API specification can take many forms but often includes specifications for routines, data structures, object classes, variables, or remote calls. POSIX, Windows API, and ASPI are examples of different forms of APIs. Documentation for the API usually is provided to facilitate usage and implementation.

## 1.1. OUTLINE OF THE PROJECT

Throughout human history, people have been keen to know about the weather, its parameters, and its impacts on their daily lives. By virtue the technological advancement, in this era, the information about the weather lies in our hands (through mobile phones or websites). We can now make ourselves aware of not only our location's temperature but also any part of the world. In this project, we will learn how to make a weather application using JavaScript.

As a web developer, grabbing data from APIs is something you are going to do often. Fetching weather data is a perfect way to get your feet wet. In this project, we are going to use the browser's built-in fetch with JavaScript to grab data from Open Weather Map's API.

## 1.2. LITERATURE REVIEW

**JavaScript**, often abbreviated as **JS**, is a high-level, interpreted scripting language that conforms to the ECMA Script specification. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it, and major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. It has APIs for working with text, arrays, dates, regular expressions, and the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities. It relies upon the host environment in which it is embedded to provide these features.

Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets.

### **1.3. PROBLEM STATEMENT**

Develop a user-friendly website that provides accurate and up-to-date temperature forecasts for cities around the world using the OpenWeather API. The website should allow users to search for a city and view its current temperature, along with a 5-day forecast. It should also display additional weather details such as weather conditions. The interface should be intuitive, responsive, and visually appealing across various devices. Ensure the website fetches real-time data from the OpenWeather API and presents it in a clear and understandable format for users.

### **1.4. OBJECTIVES**

First, we need to outline the project using HTML and CSS to create the front end of the web page.

## **CHAPTER 2**

### **AIM & SCOPE OF WEB DEVELOPMENT**

#### **2.1. REQUIREMENTS:**

##### **2.1.1. HARDWARE REQUIREMENTS:**

1. Any processor with minimum GPU memory

##### **2.1.2. SOFTWARE REQUIREMENTS:**

1. Visual Studio Code Editor
2. Open Weather API

#### **2.2. ROLE OF VISUAL STUDIO CODE:**

**Visual Studio Code**, also commonly referred to as **VS Code**, is a source-code editor developed by Microsoft for Windows, Linux, and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, and preferences, and install extensions that add functionality.

Visual Studio Code was first announced on April 29, 2015, by Microsoft at the 2015 Build conference. A preview build was released shortly thereafter.

On November 18, 2015, the source code of Visual Studio Code was released under the MIT License and made available on GitHub. Extension support was also announced. On April 14, 2016, Visual Studio Code graduated from the public preview stage and was released to the web. Microsoft has released most of Visual Studio Code's source code on GitHub under the permissive MIT License, while the binary releases by Microsoft are freeware, and include proprietary code.

Visual Studio Code is a source code editor that can be used with a variety of programming languages, including C, CSS, C++, HTML, Java, JavaScript, Node.js, and Python. It is based on the Electron framework, which is used to develop Node.js web applications that run on the Blink layout engine.

## **2.3. LANGUAGES USED**

- **HTML**
- **CSS**
- **JAVASCRIPT**

### **2.3.1. HTML**

Hypertext Markup Language (HTML) is a standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Web browsers receive HTML documents from the web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of the web page semantically and originally included cues for the appearance of the document.

### **2.3.2. CSS**

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, and enable multiple web pages to share formatting by specifying the relevant CSS separately. CSS file, and reduce complexity and repetition in the structural content.

### **2.3.3. JAVA SCRIPT**

**JavaScript** often abbreviated as **JS**, is a high-level, interpreted scripting language that conforms to the ECMA Script specification. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object orientation, and first-class functions. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications.

## **2.4. OPEN WEATHER API**

Open Weather Map is an online service that provides weather data, including current weather data, forecasts, and historical data to the developers of web services and mobile applications. For data sources, it utilizes meteorological broadcast services, raw data from airport weather stations, raw data from radar stations, and raw data from other official weather stations. All data is processed by Open Weather Map in a way that it attempts to provide accurate online weather forecast data and weather maps, such as those for clouds or precipitation. Beyond that, the service is focused on the social aspect by involving weather station owners in connecting to the service and thereby increasing weather data accuracy. The ideology is inspired by Open Street Map and Wikipedia which makes information free and available for everybody. It uses the Open Street Map for the display of weather maps.

Open Weather Map provides an API with JSON, XML, and HTML endpoints and a limited free usage tier. Making more than 60 calls per minute requires a paid subscription starting at USD 40 per month.

Access to historical data requires a subscription starting at US\$150 per month.

Users can request current weather information, extended forecasts, and graphical maps (showing cloud cover, wind speed, pressure, and precipitation).

### **2.4.1. CURRENT WEATHER DATA**

Current data is refreshed every ten minutes; it can be searched by city or by geographic coordinates on Earth.

### **2.4.2. FORECASTS**

Weather forecasts can be searched by city or by coordinates. Three-hourly forecasts are available for up to 5 days, while daily forecasts are available for up to 16 days.

### **2.4.3. SEARCHING**

The Open Weather Map geocoding system allows users to select cities by name, country, zip - code, or geographic coordinates. It is possible to search by part of the city name. To make searching result more accurate city name and country should be divided by a comma.

#### **2.4.4. MAPS**

Open Weather Map service provides lots of weather maps including Precipitation, Clouds, Pressure, Temperature, Wind, and many others. Maps can be connected to mobile applications and websites. Weather maps can be connected as layers to a wide range of maps including Direct tiles, WMS, Open Layers, Leaflet, Google maps, and Yandex maps.

## CHAPTER 3

### UML DIAGRAMS

#### 3.1 Use Case Diagrams

A **use case diagram** is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

While a use case itself might drill into a lot of detail about every possibility, a use-case diagram can help provide a higher-level view of the system. It has been said before that “Use case diagrams are the blueprints for your system”.

Due to their simplistic nature, use case diagrams can be a good communication tool for stakeholders. The drawings attempt to mimic the real world and provide a view for the stakeholder to understand how the system is going to be designed.

##### 3.1.1 Identification of Actors, Use Cases, Relations

###### Identification of Actors:

Actors represent system users. They are not part of the system. They represent anyone or anything that interacts with the system.

An actor is someone or something that:

- Interacts with or uses the system
- Provides input to and receives information from the system
- Is external to the system and has no control over the use cases

Actors are discovered by examining:

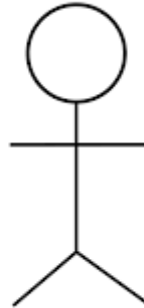
- Who directly uses the system
- Who is responsible for maintaining the system
- External hardware used by the system
- Other systems that need to interact with the system

The needs of the actor are used to develop use cases. This ensures that the system will be what the user expected.



### Graphical depiction:

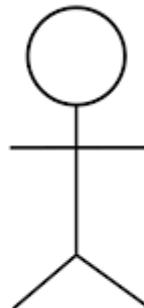
An actor is a stereotype of a class and is depicted as a “stickman” on a use-case diagram. For example,



**Actor**

Actors identified in the information system are:

- 1) User: User is allowed to use the website provided
  - to enter the location
  - to view the weather conditions along with hourly, day forecasts



**User**

### Identification of Use-Cases or Sub Use-Cases

Use case can be described as a specific way of using the system from a user’s perspective. A more detailed description might characterize a use case as:

- A pattern of behaviour the system exhibits
- A sequence of related transactions performed by an actor and the system

The UML notation for use case is:



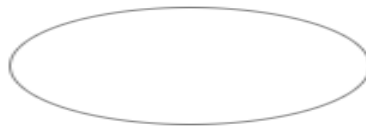
### **Purpose of Use Cases:**

- Well structured use cases denote essential system or subsystem behaviours only, and are neither overly general nor too specific.
- A use case represents a functional requirement of the system as a whole
- Use cases represent an external view of the system
- A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system with the system itself.

### **Use -cases identified for Weather Forecasting Website are:**

#### **1. Use-case name: Search Location**

This is the use case which is used by the actor to search a location and view the Weather given by Open Weather API for that particular location.



Search Location

#### **2. Use-case name: View Current Weather**

This is the use case which shows the current weather conditions to the user.



View Current Weather

#### **3. Use-case name: View Upcoming Forecasts**

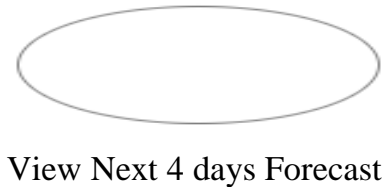
This is the use case that allow user to see the upcoming forecasts for next 5 different time slots. Note that the difference between two successive time slots is 3 hours.



View Upcoming Forecasts

#### 4. Use-case name: View Next 4 days Forecast

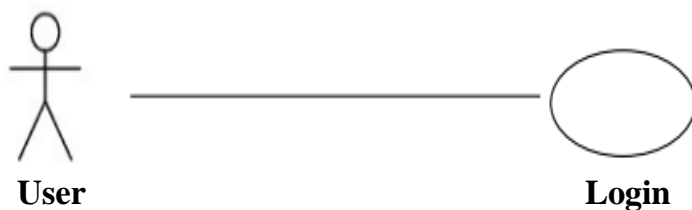
This is the use case that allow user to see the upcoming forecasts for next 4 days.



### Identification of Relations

#### Association Relationship:

An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. If two objects are usually considered independently, the relationship is an association.



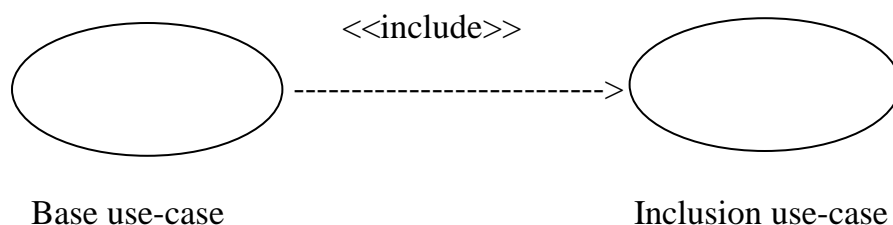
#### Dependency Relationship:

A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning.

We can provide here:

##### 1. Include relationship:

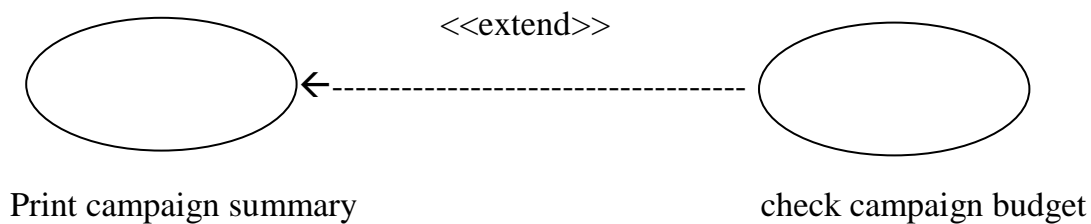
It is a stereotyped relationship that connects a base use case to an inclusion use case. An include relationship specifies how the behaviour in the inclusion use case is used by the base use case.



## 2. Extend relationship:

It is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case.

<<extend>> is used when you wish to show that a use case provides additional functionality that may be required in another use case.

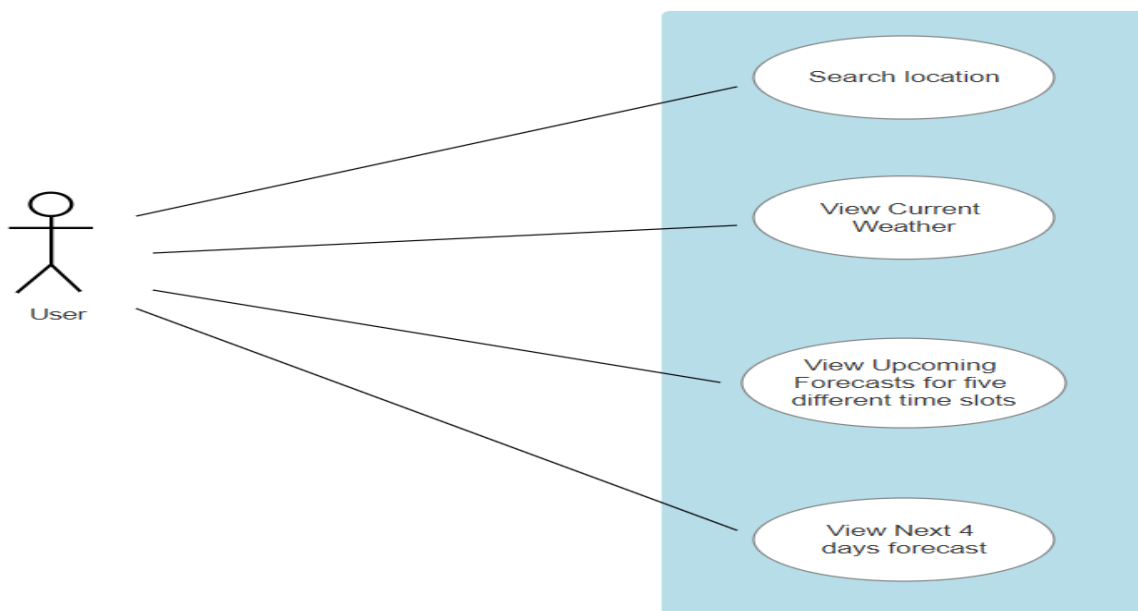


### 3.1.2 Construction of Use Case Diagram and Flow of Events

Use-case diagrams graphically represent system behavior. These diagrams present a high level view of how the system is used as viewed from an outsider's perspective.

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented.

#### Use case diagram for Weather Forecasting Website:



## Flow of Events

A flow of events is a sequence of operations performed by the system. They typically contain very detailed information. Flow of events document is typically created in the elaboration phase.

Each use case is documented with flow of events

- A description of events needed to accomplish required behaviour
- Written in terms of what the system should do, not how it should do
- Written in the domain language, not in terms of the implementation

A flow of events should include

- When and how the use case starts and ends
- What interaction the use case has with the actors
- What data is needed by the use case
- The description of any alternate or exceptional flows

The flow of events for a use case is contained in a document called the use case specification. Each project should use a standard template for the creation of the use case specification. Includes the following

1. Use case name - Brief Description
2. Flow of events –
  1. Basic flow
  2. Alternate flow
  3. Special requirements
  4. Pre-conditions
  5. Post-conditions
  6. Extension points

## Flow of Events for Search Location:

### 1. Use Case: Search location

Brief Description: This use case is started by User. It provides the capability for the user to enter the location.

### 2. Actor: User

### 3. Flow events:

#### 3.1 Basic flow:

- This use case begins when the website is opened and enters location in the search text box.
- If the searched location not exist the alternate flow 3.2 is executed.

### **3.2 Alternative flow:**

- If user enters the location that does not exist in the world, it does not give the clear output that is expected by the user.

**4. Pre-condition:** User have to enter the location that exists in the world.

**5. Post-condition:** If the Website did not give the clear output, he gave the wrong location and have to change the location that exists in the world.

### **Flow of Events for View Current Weather:**

**1. Use case:** View Current Weather.

Brief Description: This use case allows the user to see the current weather conditions for a searched location.

**2. Actor:** User

**3. Flow Events:**

#### **3.1 Basic flow:**

- This use case begins when the user searched for a location to see the weather conditions.
- If the searched location not exist the alternate flow 3.2 is executed.

#### **3.2. Alternate flow:**

- If the user enters a location that does not exist in the world, it does not give the clear output that is expected by the user.

**4. Pre-condition:** The user has to enter the location that exists in the world.

**5. Post-condition:** If the Website did not give a clear output, he gave the wrong location and had to change the location that exists in the world.

### **Flow of Events for View Current Weather:**

**1. Use case:** View Upcoming Forecast for five different time slots.

Brief Description: This use case allows the user to see the range to temperature along with the weather conditions for the next five different time slots for a searched location.

**2. Actor:** User

**3. Flow Events:**

#### **3.1 Basic flow:**

- This use case begins when the user searches for a location to see the weather conditions.
- If the searched location does not exist the alternate flow 3.2 is executed.

### **3.2. Alternate flow:**

- If user enters the location that does not exist in the world, it does not give the clear output that is expected by the user.

**4. Pre-condition:** User have to enter the location that exists in the world.

**5. Post-condition:** User can be able to see the weather conditions for the searched location only.

### **Flow of Events for View Next 4 days forecast:**

**1. Use case:** View Next 4 days forecast.

Brief Description: This use case allows the user to see the range to temperature along with the weather conditions for next four days forecast for a searched location.

**2. Actor:** User

**3. Flow Events:**

#### **3.1 Basic flow:**

- This use case begins when the user searches for a location to see the weather conditions.
- If the searched location does not exist the alternate flow 3.2 is executed.

#### **3.2. Alternate flow:**

- If the user enters a location that does not exist in the world, it does not give the clear output that is expected by the user.

**4. Pre-condition:** The user has to enter the location that exists in the world.

**5. Post-condition:** The user can be able to see the weather conditions for the searched location only.

### **3.2 Activity Diagram**

An Activity diagram is a variation of a special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. The purpose of the Activity diagram is to provide a view of flows and what is going on inside a use case or among several classes. You can also use activity diagrams to model code-specific information such as a class operation. Activity diagrams are very similar to a flowchart because you can model a workflow from activity to activity. An activity diagram is a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities.

- Activity Diagrams also may be created at this stage in the life cycle. These diagrams represent the dynamics of the system. They are flow charts that are used to show the workflow of a system; that is, they show the flow of control from activity to activity in the system, what activities can be done in parallel, and any alternate paths through the flow.

- At this point in the life cycle, activity diagrams may be created to represent the flow across use cases or they may be created to represent the flow within a particular use case.
- Later in the life cycle, activity diagrams may be created to show the workflow for an operation.

The following tools are used on the activity diagram toolbox to model activity diagrams:

**Activities:** An activity represents the performance of some behavior in the workflow.



**Transitions:** Transitions are used to show the passing of the flow of control from activity to activity. They are typically triggered by the completion of the behavior in the originating activity.



**Decision Points:** When modeling the workflow of a system it is often necessary to show where the flow of control branches based on a decision point. The transitions from a decision point contain a guard condition, which is used to determine which path from the decision point is taken. Decisions along with their guard conditions allow you to show alternate paths through a work flow.



Decision Point

**Start state:** A start state explicitly shows the beginning of a workflow on an activity diagram or the beginning of the execution of a state machine on a state chart diagram.



Start state

**End state:** An End state represents a final or terminal state on an activity diagram or state chart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a state chart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.



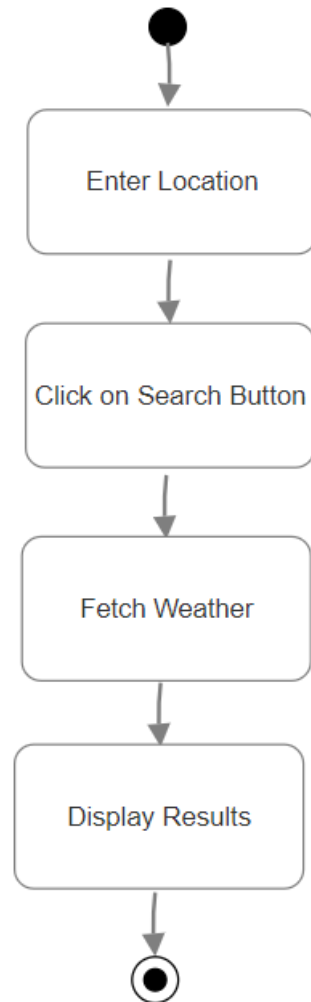
End state

**Swim Lanes:** Swim lanes may be used to partition an activity diagram. This typically is done to show what person or organisation is responsible for the activities contained in the swim lane.

- Horizontal synchronization
- Vertical synchronization



## Activity Diagram for Weather Forecasting



### 3.3 Sequence Diagram

A sequence diagram is a graphical view of a scenario that shows object interaction in a time based sequence--what happens first, what happens next...

Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

A sequence diagram has two dimensions: the vertical dimension represents time; the horizontal dimension represents different objects. The vertical line is called the object's lifeline. The lifeline represents the object's existence during the interaction.

#### Steps:

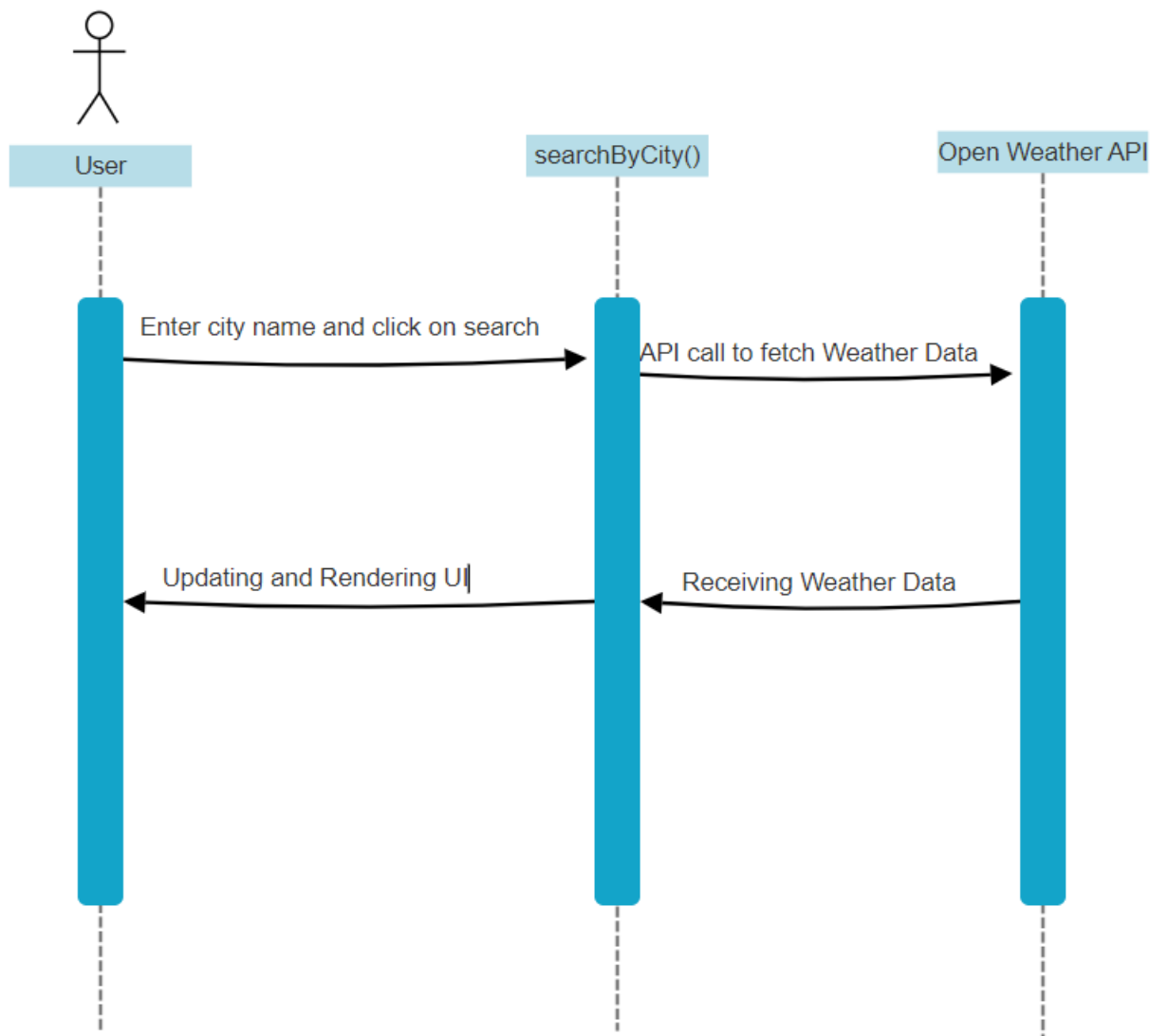
1. An object is shown as a box at the top of a dashed vertical line. Object names can be specific (e.g., Algebra 101, Section 1) or they can be general (e.g., a course offering). Often, an anonymous object (class name may be used to represent any object in the class).
2. Each message is represented by an Arrow between the lifelines of two objects. The order in which these messages occur is shown from top to bottom on the page. Each message is labeled with the message name.

There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other. A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

### Elements of the Sequence Diagram:

- Objects
- Links
- Messages
- Focus of control
- Object life line

### Sequence Diagram for Weather Forecasting



### 3.4 Collaboration Diagram

Collaboration diagrams are the second kind of interaction diagram in the UML diagrams. They are used to represent the collaboration that realizes a use case. The most significant difference between the two types of interaction diagram is that a collaboration diagram explicitly shows the links between the objects that participate in a collaboration, as in sequence diagrams, there is no explicit time dimension.

#### Message labels in collaboration diagrams:

Messages on a collaboration diagram are represented by a set of symbols that are the same as those used in a sequence diagram, but with some additional elements to show sequencing and recurrence as these cannot be inferred from the structure of the diagram. Each message label includes the message signature and a sequence number that reflects call nesting, iteration, branching, concurrency, and synchronization within the interaction.

The formal message label syntax is as follows:

```
[predecessor] [guard-condition] sequence-expression [return-value ':='] message-name '('  
[argument-list] ')'
```

A **predecessor** is a list of sequence numbers of the messages that must occur before the current message can be enabled. This permits the detailed specification of branching pathways. The message with the immediately preceding sequence number is assumed to be the predecessor by default, so if an interaction has no alternative pathways the predecessor list may be omitted without any ambiguity. The syntax for a predecessor is as follows:

```
sequence-number { ',' sequence-number } 'I'
```

The 'I' at the end of this expression indicates the end of the list and is only included when an explicit predecessor is shown.

**Guard conditions** are written in Object Constraint Language (OCL), and are only shown where the enabling of a message is subject to the defined condition. A guard condition may be used to represent the synchronization of different threads of control.

A **sequence expression** is a list of integers separated by dots ('.') optionally followed by a name (a single letter), optionally followed by a recurrence term, and terminated by a colon. A sequence expression has the following syntax:

```
integer { '.' Integer } [name][recurrence] ':'
```

In this expression, the integer represents the sequential order of the message. This may be nested within a loop or a branch construct, so that, for example, message 5.1 occurs after message 5.2 and both are contained within the activation of message 5.

The name of a sequence expression is used to differentiate two concurrent messages since these are given the same sequence number. For example, messages 3.2.1a and 3.2.1b are concurrent with the activation of message 3.2.

Recurrence reflects either iterative or conditional execution and its syntax is as follows:

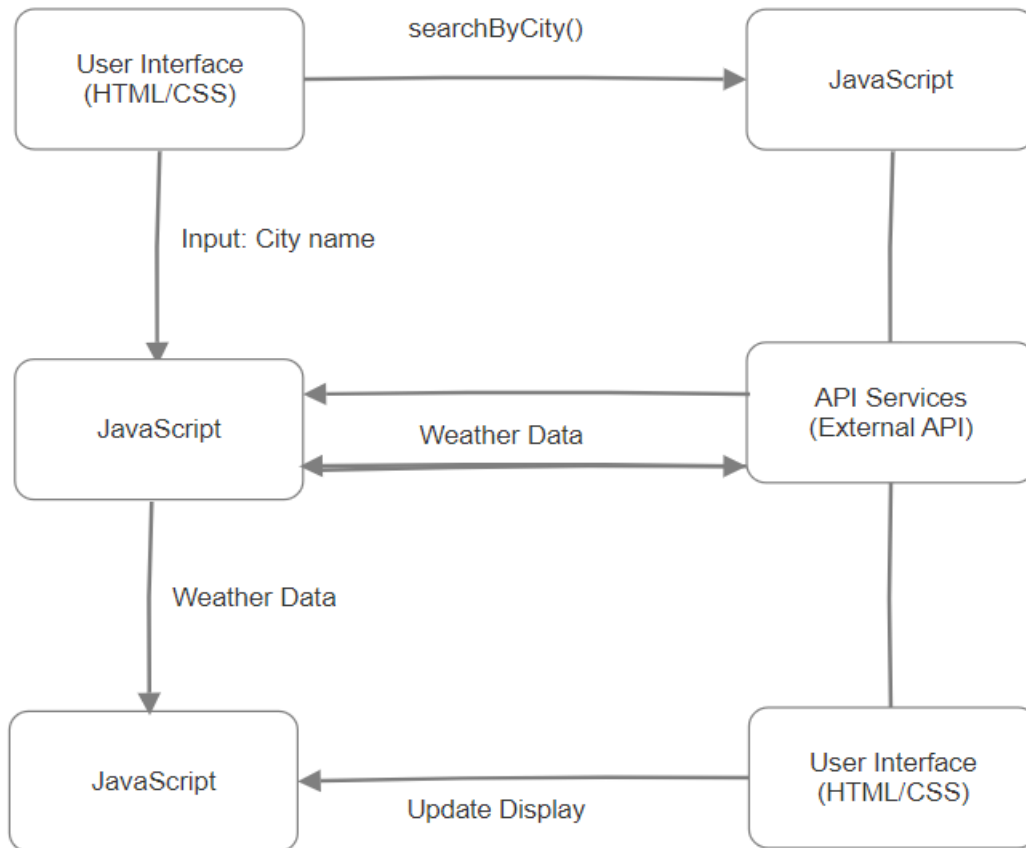
Branching: '['condition-clause'],

Iteration: '\* '['iteration-clause']'

### Elements:

- Objects, Messages, Path, Sequence Numbers, Links

### Collaboration Diagram for Weather Forecasting



### 3.5 Class Diagram

Class diagrams contain icons representing classes, packages, interfaces, and their relationships. You can create one or more class diagrams to depict the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model.

**Class:** A Class is a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects, and common semantics.

Thus, a class is a template to create objects. Each object is an instance of some class and objects cannot be instances of more than one class.

Classes should be named using the vocabulary of the domain.

For example, the CourseOffering class may be defined with the following characteristics:

Attributes - location, time offered

Operations - retrieve location, retrieve time of day, add a student to the offering.

Each object would have a value for the attributes and access to the operations specified by the CourseOffering class.

In the UML, classes are represented as compartmentalized rectangles.

The top compartment contains the name of the class.

The middle compartment contains the structure of the class (attributes).

The bottom compartment contains the behavior of the class (operations) as shown below.



### Object:

- An Object is a representation of an entity, either real-world or conceptual.
- An Object is a concept, abstraction, or thing with well-defined boundaries and meaning for an application.
- Each Object in a system has three characteristics: state, behavior, and identity.

**State:** The State of an object is one of the possible conditions in which it may exist. The state of an object typically changes over time, and is defined by a set of properties (called attributes), with the values of the properties, plus the relationships the object may have with other objects.

For example, a course offered object in the registration system may be in one of two states: open and closed. It is available in the open state if the value is less than 10 otherwise closed.

### Behavior:

- Behavior determines how an object responds to requests from other objects.
- Behavior is implemented by the set of operations for the object.

For example, In the registration system, a course offering could have the behaviors add a student and delete a student.

**Identity:** Identity means that each object is unique even if its state is identical to that of another object.

**Attributes:** Attributes are part of the essential description of a class. They belong to the class, unlike objects, which instantiate the class. Attributes are the common structure of what a member of the class can 'know'. Each object will have its own, possibly unique, value for each attribute.

Guidelines for identifying attributes of classes are as follows:

- Attributes usually correspond to nouns followed by prepositional phrases
- Keep the class simple; state only enough attributes to define the object state.
- Attributes are less likely to be fully described in the problem statement.
- Omit derived attributes.
- Do not carry discovery attributes to excess.

### **Stereotypes and Classes:**

As like stereotypes for relationships in use case diagrams, classes can also have stereotypes. Here a stereotype provides the capability to create a new kind of modeling element. Here, we can create new kinds of classes. Some common stereotypes for a class are entity Class, boundary Class, control class, and exception.

### **Entity Classes**

- An **entity class** models information and associated behavior that is generally long-lived.
- This type of class may reflect a real-world entity or it may be needed to perform tasks internal to the system.
- They are typically independent of their surroundings; that is, they are not sensitive to how the surroundings communicate with the system.

### **Boundary Classes:**

Boundary classes handle the communication between the system surroundings and the inside of the system. They can provide the interface to a user or another system (i.e., the interface to an actor). They constitute the surroundings-dependent part of the system. Boundary classes are used to model the system interfaces.

Boundary classes are also added to facilitate communication with other systems. During the design phase, these classes are refined to take into consideration the chosen communication protocols.

### **Control Classes**

- Control classes model sequencing behavior specific to one or more use cases.
- Control classes coordinate the events needed to realize the behavior specified in the use case.
- Control classes typically are application-dependent classes.

In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use case.

In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use case.

### **Need for Relationships among Classes:**

All systems are made up of many classes and objects. System behavior is achieved through the collaboration of the objects in the system.

Two types of relationships in the CLASS diagram are:

1. Association Relationship
2. Aggregation Relationship

### **1. Association Relationship:**

An association is a bidirectional semantic connection between classes. It is not a data flow as defined in structured analysis and design data may flow in either direction across the association. An association between classes means that there is a link between objects in the associated classes.

### **2. Aggregation Relationship:**

An aggregation relationship is a specialized form of association in which a whole is related to its part(s). Aggregation is known as a "part-of" or containment relationship. The UML notation for an aggregation relationship is an association with a diamond next to the class denoting the aggregate(whole).

### **3. Super-sub structure (Generalization Hierarchy):**

These allow objects to be built from other objects. The super-sub class hierarchy is a relationship between classes, where one class is the parent class of another class.

### **NAMING RELATIONSHIP:**

An association may be named. Usually, the name is an active verb or verb phrase that communicates the meaning of the relationship. Since the verb phrase typically implies a reading direction, it is desirable to name the association so it reads correctly from left to right or top to bottom. The words may have to be changed to read the association in the other direction (e.g., Buses are allotted to Routes). It is important to note that the name of the association is optional.

### **ROLE NAMES:**

The end of an association where it connects to a class is called an association role Role names can be used instead of association names.

A role name is a noun that denotes how one class associates with another. The role name is placed on the association near the class that it modifies, and may be placed on one or both ends of an association line.

- It is not necessary to have both a role name and an association name.
- Associations are named or role names are used only when the names are needed for clarity.

### **MULTIPLICITY INDICATORS:**

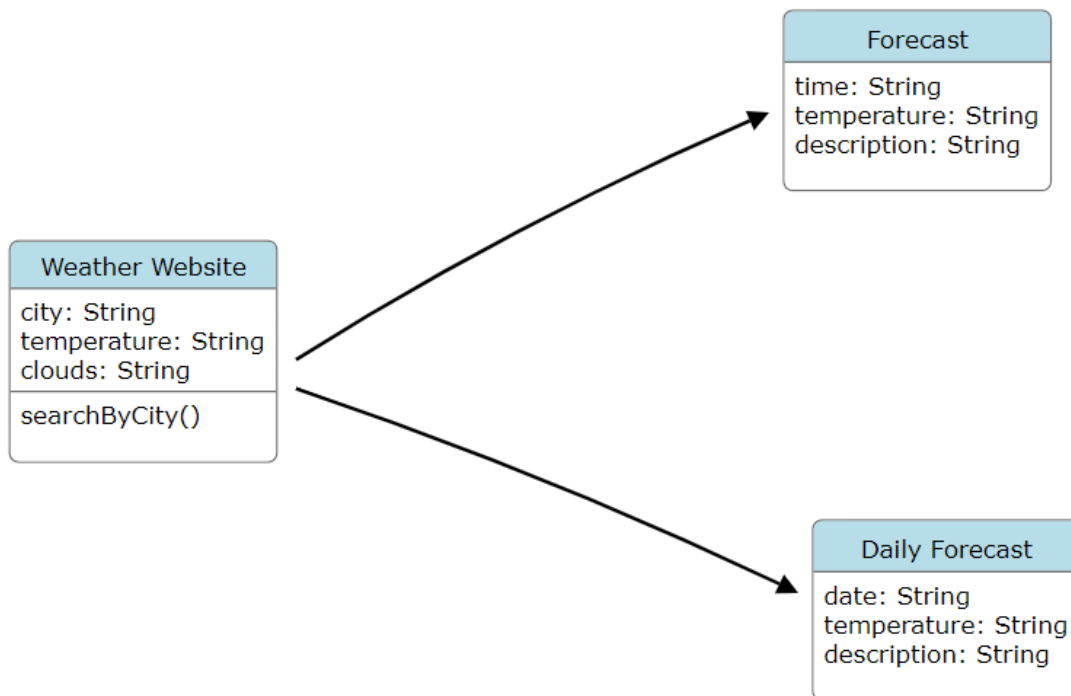
Although multiplicity is specified for classes, it defines the number of objects that participate in a relationship. Multiplicity defines the number of objects that are linked to one another. There are two multiplicity indicators for each association or aggregation one at each end of the line. Some common multiplicity indicators are

1	Exactly one
0...*	Zero or one
1...*	One or more

0...1  
5...8  
4...7,9

Zero or more  
Specific range (5,6,7, or 8)  
Combination (4,5,6,7, or 9)

## CLASS DIAGRAM FOR WEATHER FORECASTING





## CHAPTER 4

### METHODS & MATERIAL USED

#### MATERIAL USED:

- Visual Studio Code Editor
- API Key
- Google Chrome/Microsoft Edge/Firefox

#### 4.1 DESIGNING IN TEXT EDITOR:

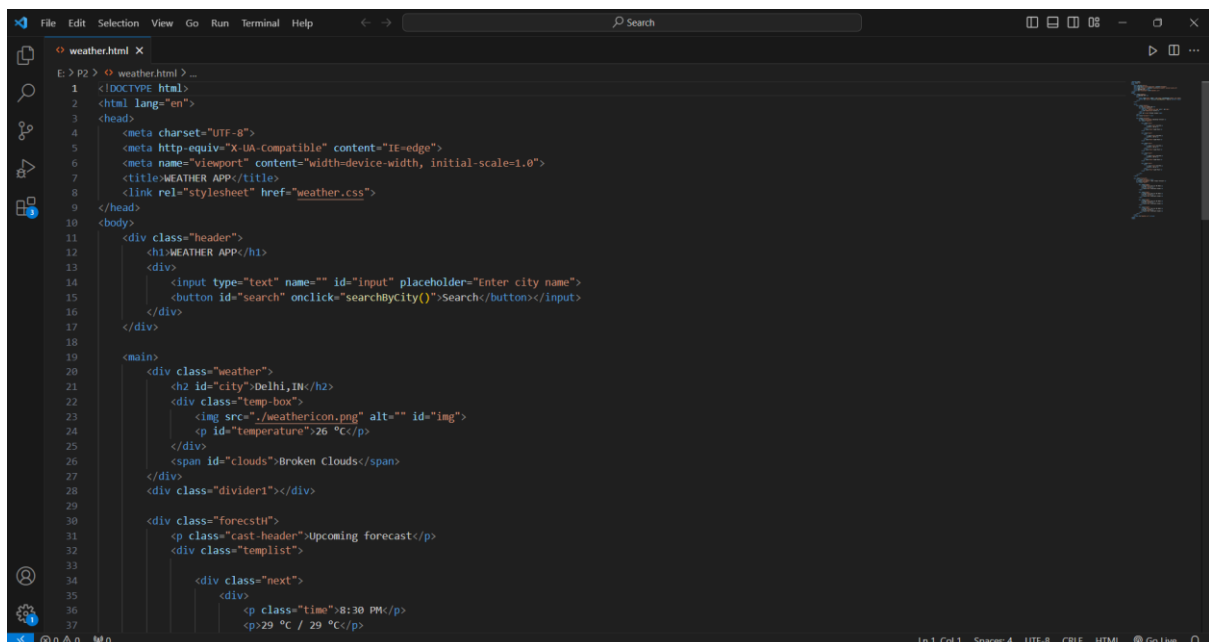
The project will stick to the basic functionalities expected of a simple text editor which includes the ability to write something on the notepad, save it, and open and modify it whenever required. For this tutorial, we will make use of VS Code editor to HTML and CSS code to create the front-end part of the web page.

#### SAMPLE APPLICATION CODE

Various modules in the system are

1. Weather.html
2. Weather.css
3. Weather.js

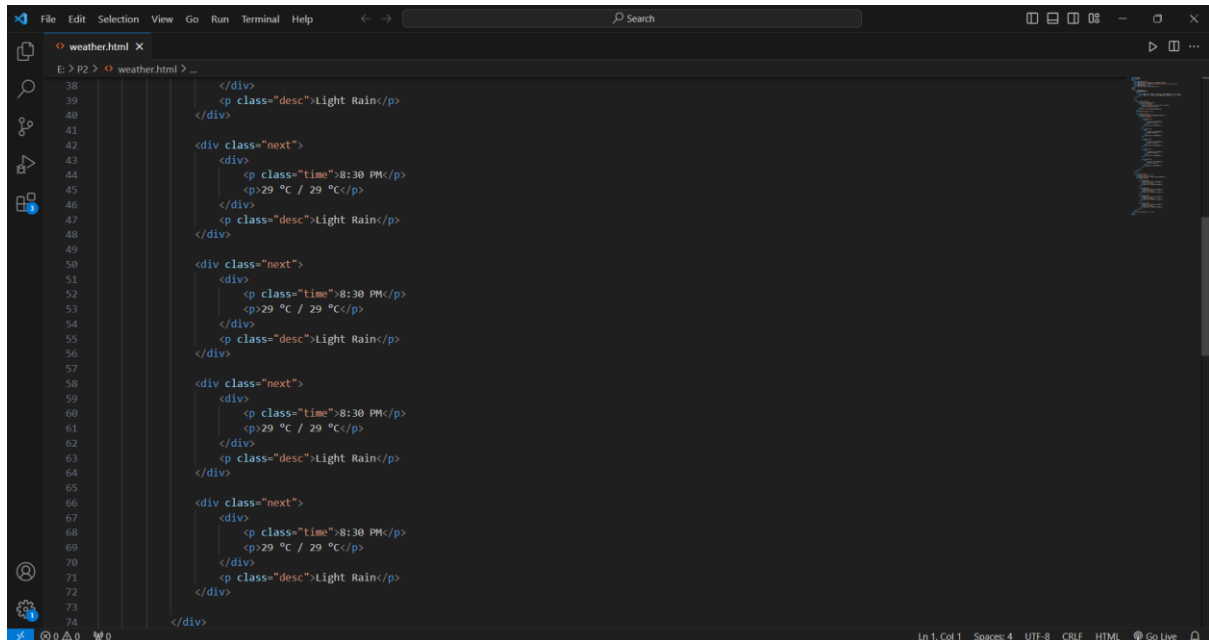
#### Code for Weather.html:



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>WEATHER APP</title>
8   <link rel="stylesheet" href="weather.css">
9 </head>
10 <body>
11   <div class="header">
12     <h1>WEATHER APP</h1>
13     <div>
14       <input type="text" name="" id="input" placeholder="Enter city name">
15       <button id="search" onclick="searchByCity()">Search</button></div>
16     </div>
17   </div>
18   <main>
19     <div class="weather">
20       <h2 id="city">Delhi, IN</h2>
21       <div class="temp-box">
22         
23         <p id="temperature">26 °C</p>
24       </div>
25       <p id="clouds">Broken Clouds</p>
26     </div>
27     <div class="divider1"></div>
28     <div class="forecastH">
29       <p class="cast-header">Upcoming forecast</p>
30       <div class="templist">
31         <div class="next">
32           <div>
33             <p class="time">8:30 PM</p>
34             <p>29 °C / 29 °C</p>
35           </div>
36         </div>
37       </div>
38     </div>
39   </main>
40 </body>
41 </html>
```

Fig. 4.1 OUTLINE OF THE PROJECT

At this point we have already looked at some HTML code, how to place heading, how to add text box, how to link CSS file to the HTML page (Note that both HTML and CSS files are in the same directory), how to add button, and how different tags (like div, span, etc.) works. Now it's time to look at the next part of the HTML page which includes adding weather forecasts to the static webpage.



```

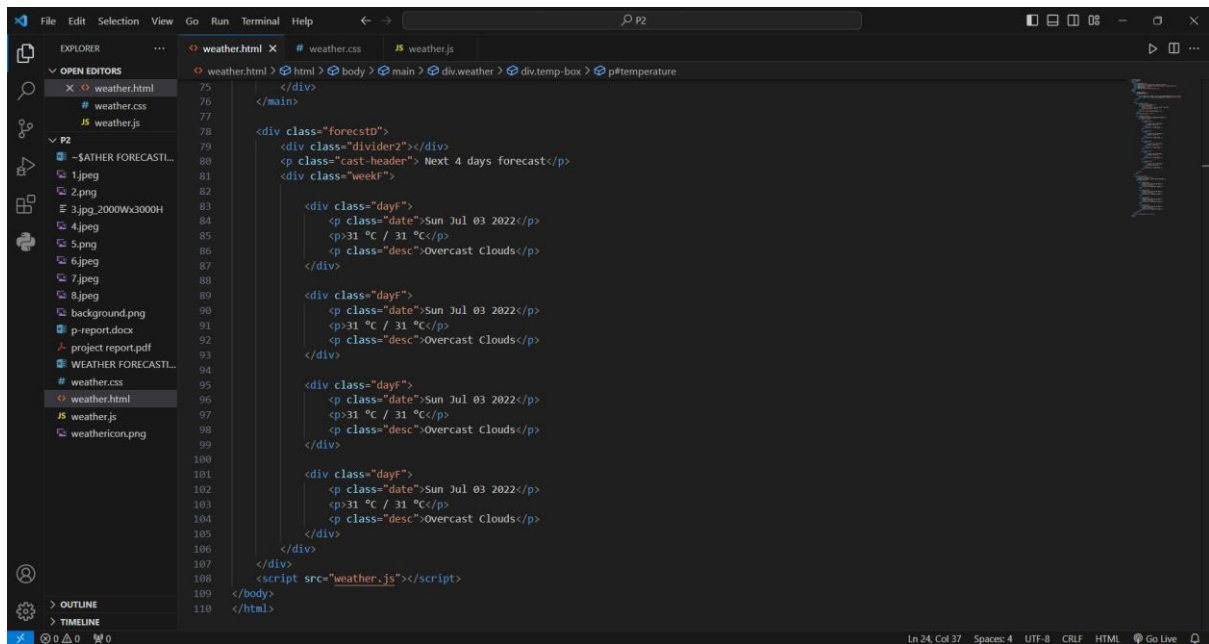
38     </div>
39     <p class="desc">Light Rain</p>
40 </div>
41
42 <div class="next">
43   <div>
44     <p class="time">8:30 PM</p>
45     <p>29 °C / 29 °C</p>
46   </div>
47   <p class="desc">Light Rain</p>
48 </div>
49
50 <div class="next">
51   <div>
52     <p class="time">8:30 PM</p>
53     <p>29 °C / 29 °C</p>
54   </div>
55   <p class="desc">Light Rain</p>
56 </div>
57
58 <div class="next">
59   <div>
60     <p class="time">8:30 PM</p>
61     <p>29 °C / 29 °C</p>
62   </div>
63   <p class="desc">Light Rain</p>
64 </div>
65
66 <div class="next">
67   <div>
68     <p class="time">8:30 PM</p>
69     <p>29 °C / 29 °C</p>
70   </div>
71   <p class="desc">Light Rain</p>
72 </div>
73
74 </div>

```

**FIG. 4.2 TODAY'S FORECASTING PAGE**

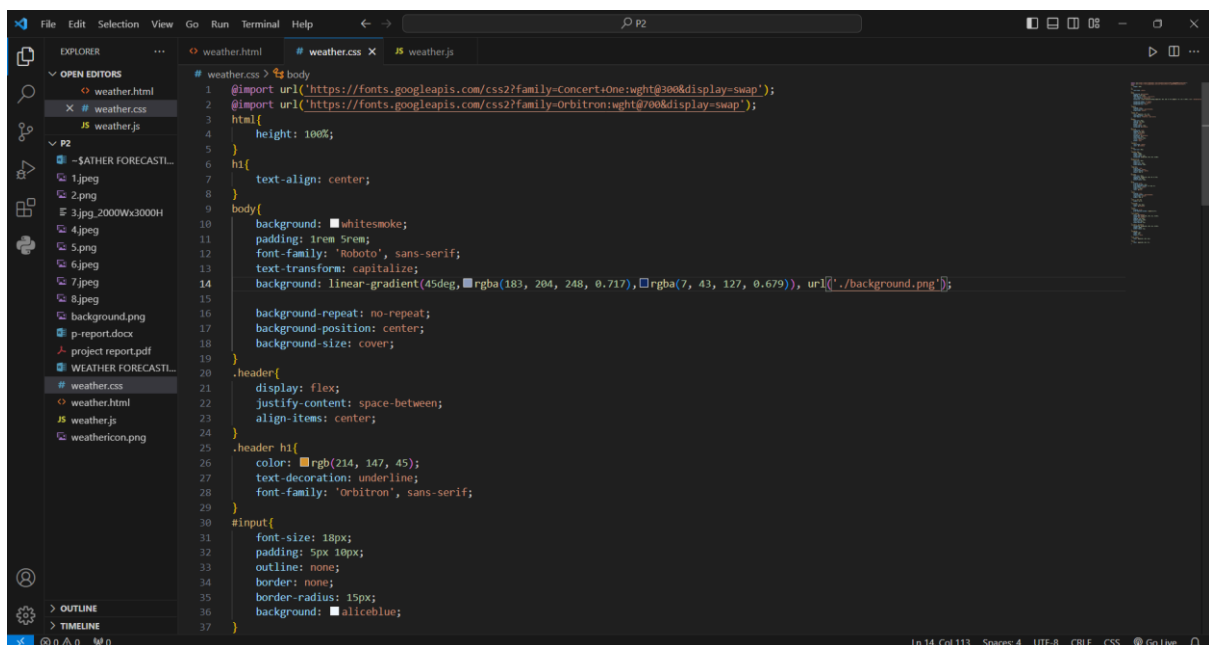
In the above figure we observed that our **“WEATHER FORECASTING WEBSITE”** includes the range of today's weather in 5 different time slots. The time slots depend on the time of searching. For example, if we searched at 7:00 PM, it shows the weather for 8:30 PM, 11:30 PM, 2:30 AM, 5:30 AM, and 8:30 AM. If we searched at 7:00 AM, then it shows weather for 8:30 AM, 11:30 AM, 2:30 PM, 5:30 PM, 8:30 PM. It prints the temperature in °C along with the weather conditions (like Light Rain, Overcast Clouds, Broken Clouds, Scattered Clouds, Clear Sky, Few Clouds, etc.).

In the next figure, we observe that our website includes the range of the next four days forecast from the day of the search.



**FIG. 4.3 NEXT 4 DAYS FORECAST**

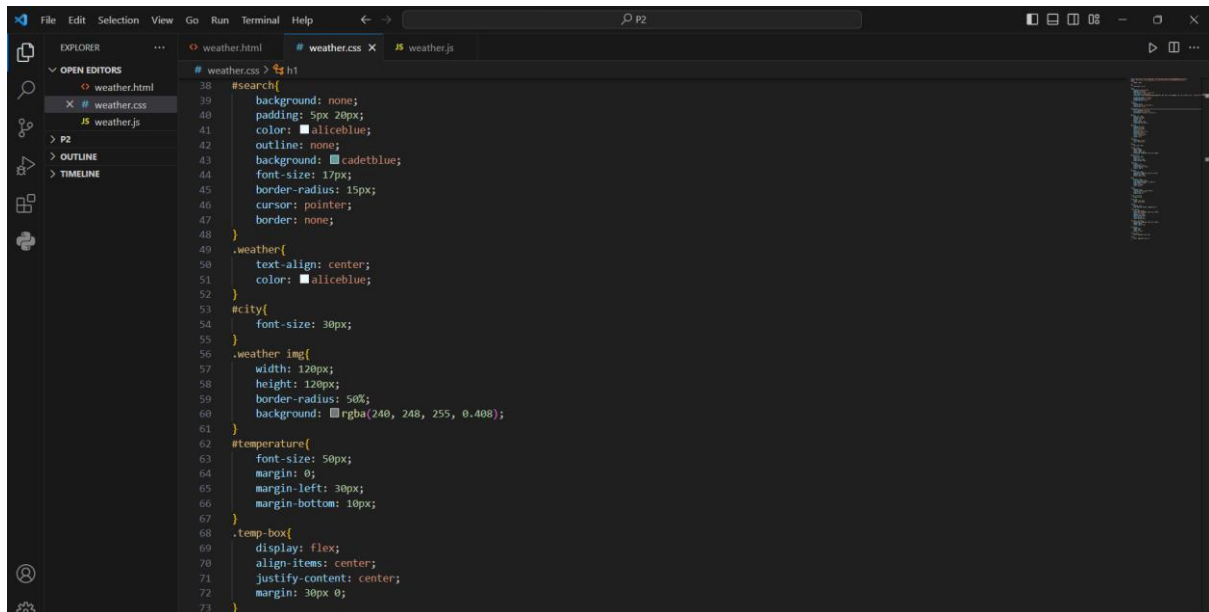
Now this is the time to look at CSS styling, the below figure describes how to style different elements like headings (h1, h2, h3, h4, h5, h6), body, etc. CSS allows us to apply styles through “id’s”, and “classes”. To apply styling through “id” we can use the “#” symbol and for applying styles through “class” we can use the “.” symbol.



**FIG. 4.4 STYLING OF THE WEB PAGE**

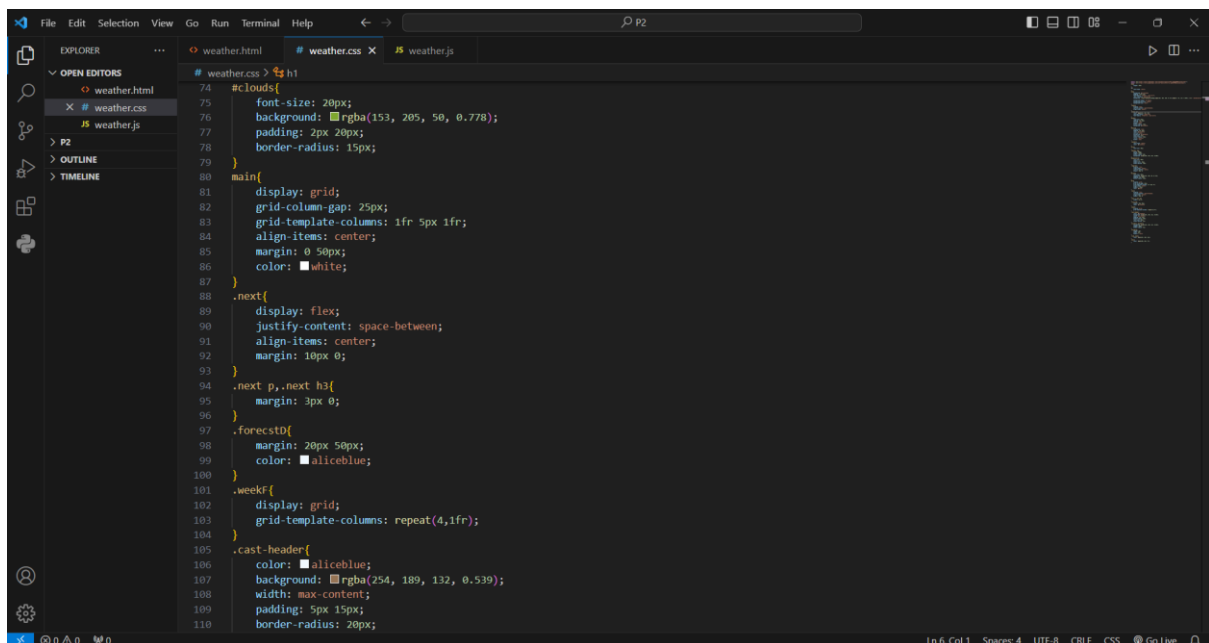
In Fig. 3.5, we applied styles using id’s(#) and classes(.). For this, we have to include the id or class attribute in the element we want to style. Note that no two elements have the same id, and CSS allows any number of elements to use the same class name.

But while applying styles using classes, if we want to apply the style to the particular element having the class name the same as another element's class name, use the class name followed by the element name in a format of “.class\_name element\_name”.

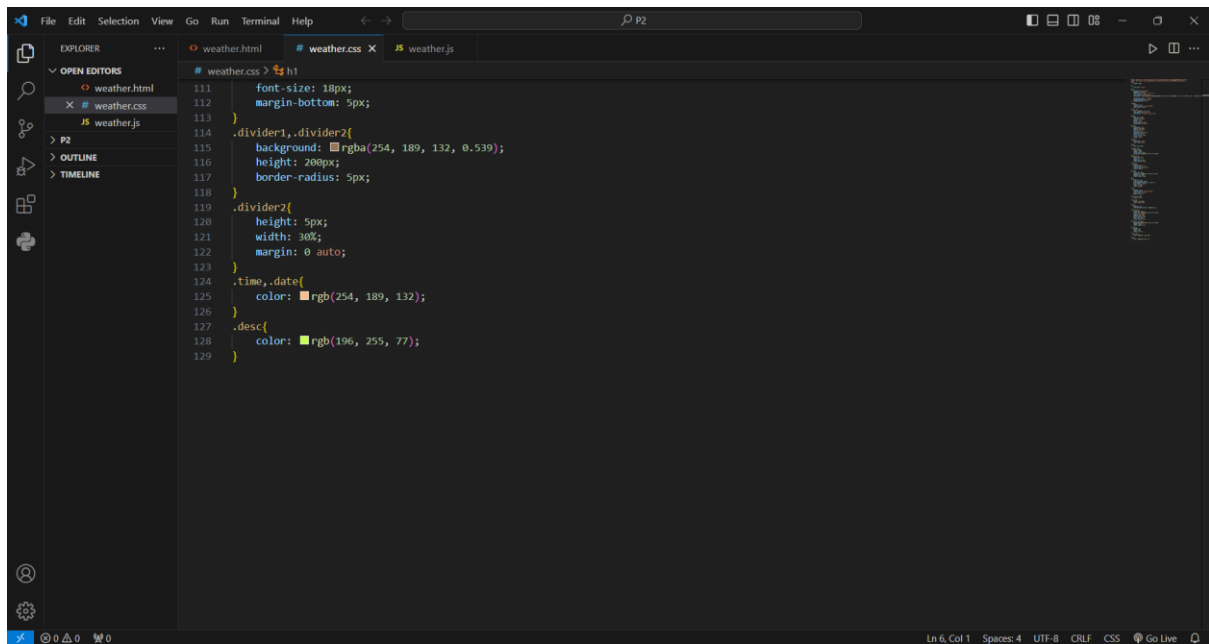


**FIG. 4.5 CSS USING ID'S AND CLASSES**

The below figures show how to apply styles to more than one element simultaneously. To do this use the elements separated by commas to apply the same styles.

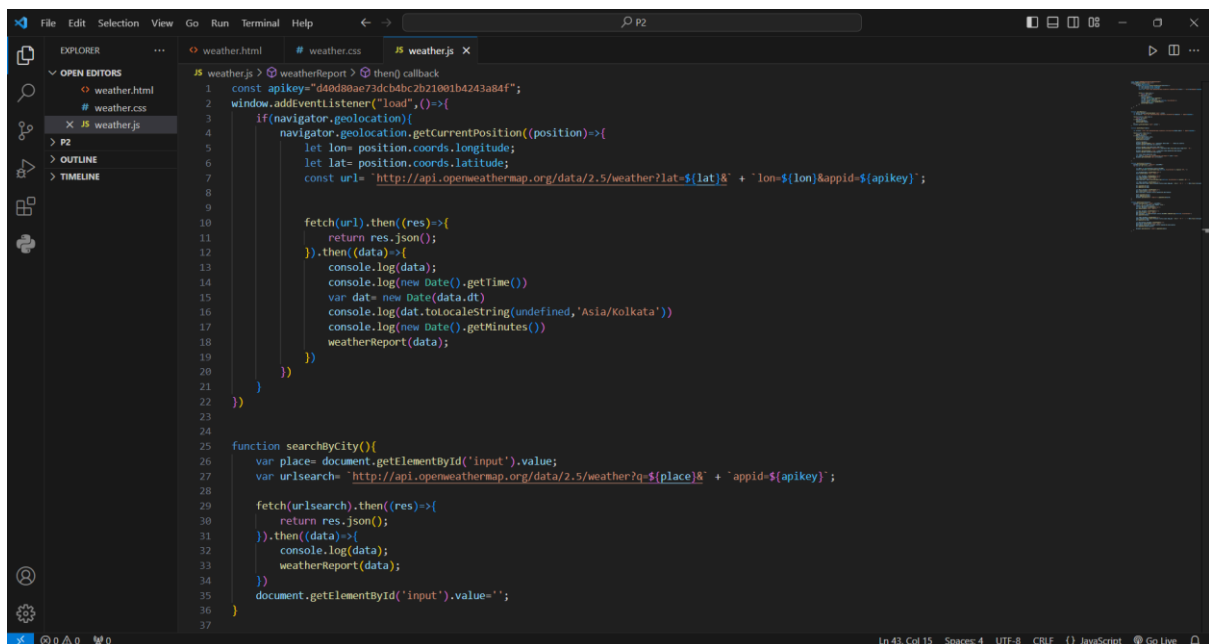


**FIG. 4.6 APPLY STYLES TO MORE THAN ONE ELEMENT**



**FIG. 4.7 CSS STYLING USING CLASSES**

The above figure describes applying of styles using classes. At the same time, we also applied same style simultaneously to the two elements.



**FIG. 4.8 INCLUDING API KEY IN JS FILE**

### API KEY AND EVENT LISTENER:

- In the first line of above figure, API key is stored in the “apikey” variable to access the weather. Using that key we can access temperature, humidity, weather conditions, etc.
- In the next line, the event listener triggers when the webpage finishes loading.

## GEOLOCATION-BASED WEATHER RETRIEVAL:

- `'navigator.geolocation.getCurrentPosition((position)=>{...});'`: Checks if the browser supports geolocation. If it does, it retrieves the user's current position.
- The latitude and longitude of the user's location are obtained from `'position.coords.latitude'` and `'position.coords.longitude'`.
- Constructs the URL using the obtained latitude, longitude, and the API key to fetch weather data from the OpenWeatherMap API.  
(`'http://api.openweathermap.org/data/2.5/weather'`).

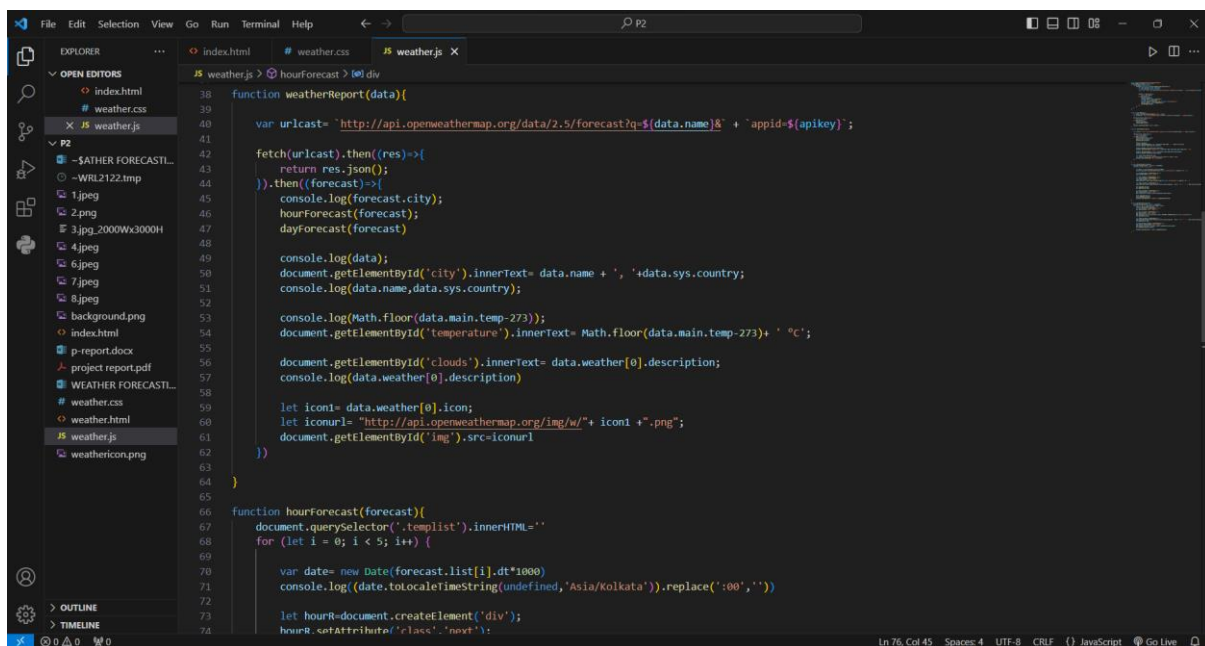
## FETCHING CURRENT WEATHER DATA

- Uses `'fetch'` to make an HTTP GET request to the constructed URL.
- Converts the response to JSON format using `'res.json()'` in the promise chain.
- Once the JSON data is retrieved, it logs the data, current time, and weather report details.
- Calls the `'weatherReport()'` function and passes the received data.

## ADDITIONAL FUNCTIONALITY:

- `'searchByCity()'`: Handles the user input for searching weather by city name using the OpenWeatherMap API.

The code uses the `'fetch()'` function to make asynchronous HTTP requests to the OpenWeatherMap API endpoints, retrieves JSON-formatted data, and then processes and displays this information on the webpage.



```
38 function weatherReport(data){
39
40   var urlcast= "http://api.openweathermap.org/data/2.5/forecast?q=${data.name}&"+ 'appid=${apikey}';
41
42   fetch(urlcast).then((res)=>{
43     return res.json();
44   }).then((forecast)=>{
45     console.log(forecast.city);
46     hourForecast(forecast);
47     dayForecast(forecast);
48
49     console.log(data);
50     document.getElementById('city').innerText= data.name + ', '+data.sys.country;
51     console.log(data.name,data.sys.country);
52
53     console.log(Math.floor(data.main.temp-273));
54     document.getElementById('temperature').innerText= Math.floor(data.main.temp-273)+ ' °C';
55
56     document.getElementById('clouds').innerText= data.weather[0].description;
57     console.log(data.weather[0].description)
58
59     let icon1= data.weather[0].icon;
60     let iconurl= "http://api.openweathermap.org/img/w/"+ icon1 +".png";
61     document.getElementById('img').src=iconurl
62   })
63 }
64
65
66 function hourForecast(forecast){
67   document.querySelector('.templist').innerHTML=''
68   for (let i = 0; i < 5; i++) {
69
70
71     var date= new Date(forecast.list[i].dt*1000)
72     console.log((date.toLocaleTimeString(undefined,'Asia/Kolkata')).replace(':00',''))
73
74     let hourR=document.createElement('div');
75     hourR.setAttribute('class','next');
```

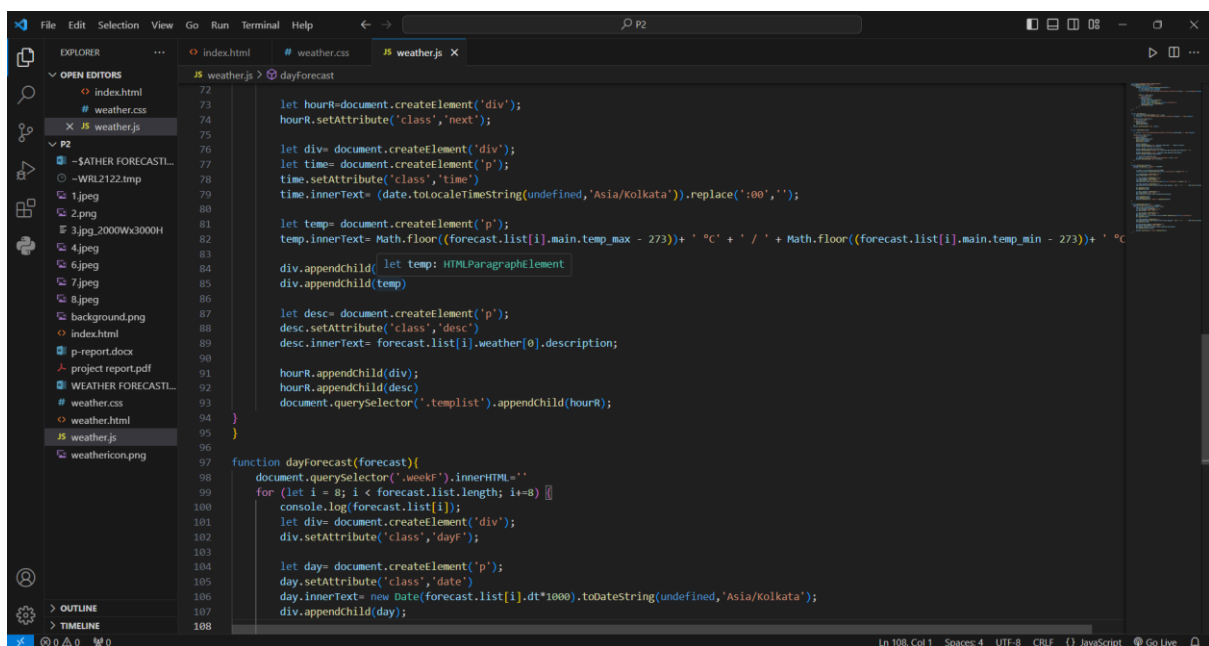
FIG. 4.9 HOURLY FORECAST FUNCTION

## FUNCTIONS FOR DISPLAYING WEATHER INFORMATION

- **'weatherReport(data)'**: Takes the weather data and displays the current weather details such as location, description, and an icon representing the weather.
- **'hourForecast(forecast)'**: Displays the hourly forecast for the next 5 hours, including time, temperature, range, and weather description.
- **'dayForecast(forecast)'**: Displays the daily forecast for the upcoming days, including date, temperature range, and weather description.

## HOURLY FORECAST

- The **'hourForecast(forecast)'** function is responsible for displaying the hourly weather forecast for the upcoming hours. It takes the forecast data retrieved from the OpenWeatherMap API as an argument.
- The function iterates through a portion of the forecast data (typically the next few hours) to display the weather forecast for each hour within that time range.



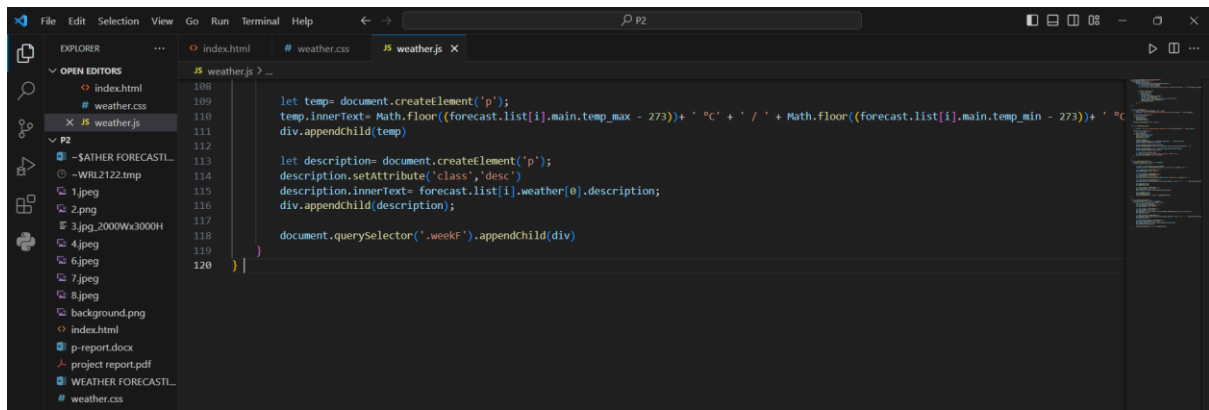
```
72 let hourR=document.createElement('div');
73 hourR.setAttribute('class','next');
74
75 let div= document.createElement('div');
76 let time= document.createElement('p');
77 time.setAttribute('class','time');
78 time.innerText= (date.toLocaleTimeString(undefined,'Asia/Kolkata')).replace(':00','');
79
80 let temp= document.createElement('p');
81 temp.innerText= Math.floor((forecast.list[i].main.temp_max - 273))+ ' °C' + ' / ' + Math.floor((forecast.list[i].main.temp_min - 273))+ ' °C';
82
83 div.appendChild( let temp: HTMLParagraphElement
84 div.appendChild(temp)
85
86 let desc= document.createElement('p');
87 desc.setAttribute('class','desc');
88 desc.innerText= forecast.list[i].weather[0].description;
89
90 hourR.appendChild(div);
91 hourR.appendChild(desc);
92 document.querySelector('.templst').appendChild(hourR);
93
94 }
95
96
97 function dayForecast(forecast){
98 document.querySelector('.week').innerHTML=""
99 for (let i = 0; i < forecast.list.length; i+=8) {
100 console.log(forecast.list[i]);
101 let div= document.createElement('div');
102 div.setAttribute('class','dayf');
103
104 let day= document.createElement('p');
105 day.setAttribute('class','date');
106 day.innerText= new Date(forecast.list[i].dt*1000).toDateString(undefined,'Asia/Kolkata');
107 div.appendChild(day);
108 }
```

FIG. 4.10 DAILY FORECAST FUNCTION

## DAILY FORECAST

- The **'dayForecast(forecast)'** function is responsible for displaying the daily weather forecast for the upcoming days. It takes the forecast data retrieved from the OpenWeatherMap API as an argument.
- The function iterates through a subset of the forecast data, especially targeting daily weather forecasts, and displays information for each day within that range.

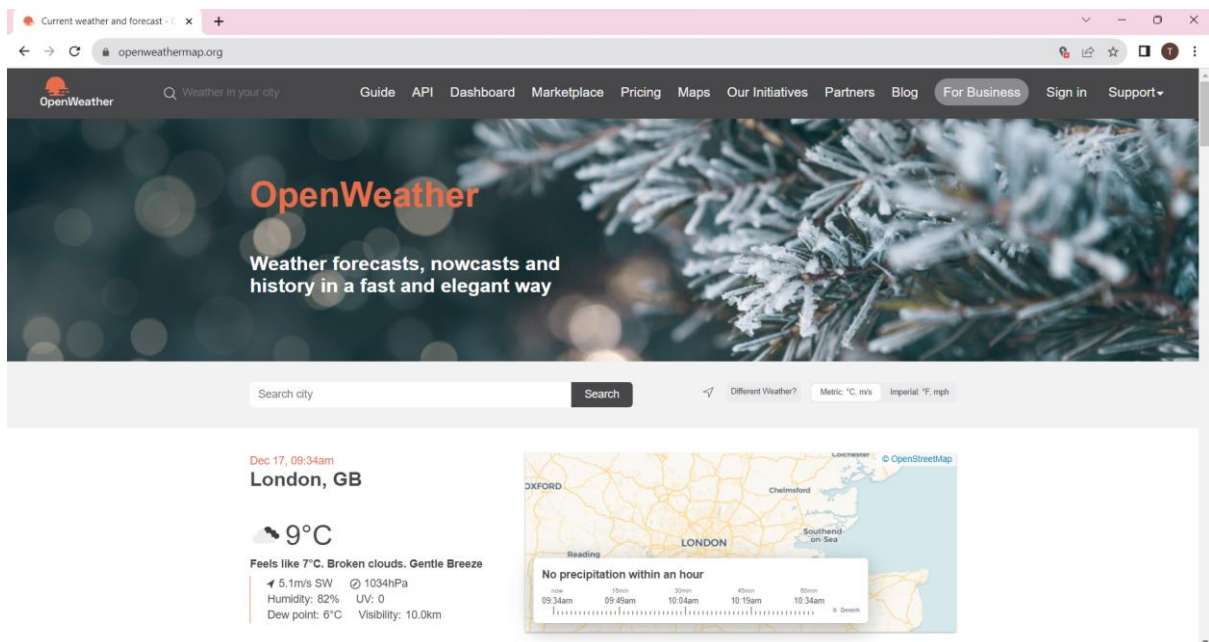




**FIG 4.11 CONVERSION OF TEMPERATURE**

## 4.2 CONNECTION OF API:

Since we use open weather API, we need to sign up on open weather map website i.e., <https://openweathermap.org/>



**FIG. 4.12 OPEN WEATHER MAP WEBSITE**

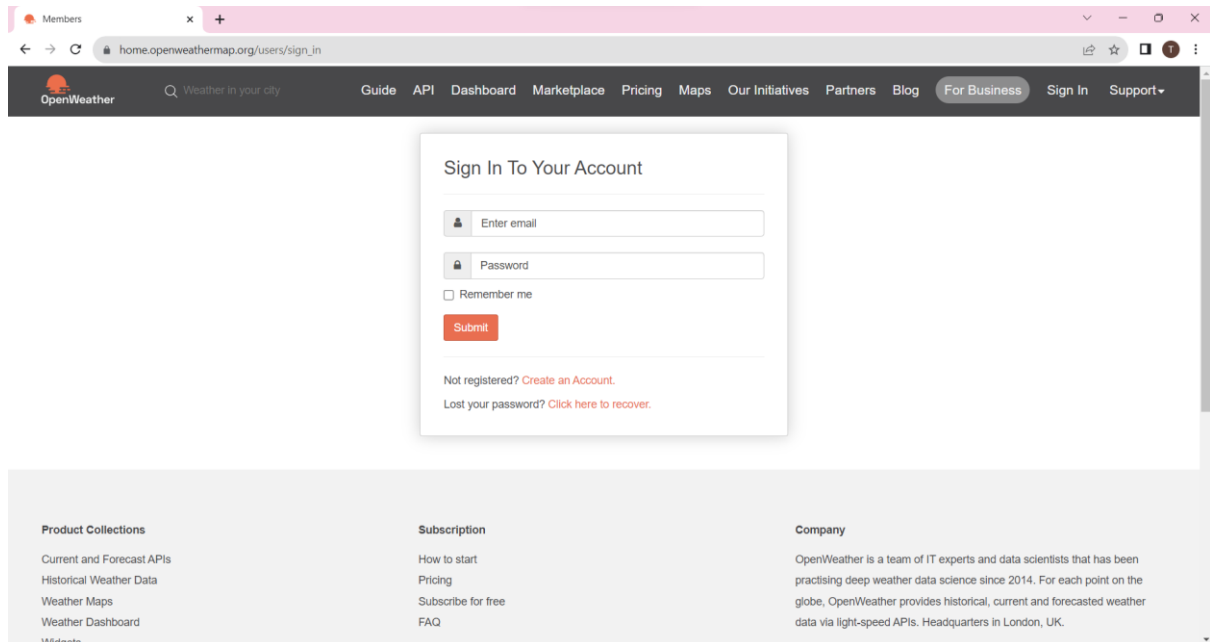
API recommend making calls to the API no more than one time every 10 minutes for one location (city / coordinates / zip-code). This is due to the fact that weather data in our system is updated no more than one time every 10 minutes.

The server name is `api.openweathermap.org`. Please, never use the IP address of the server.

Better call API by city name or city ID instead of city coordinates or zip code to get a precise result.



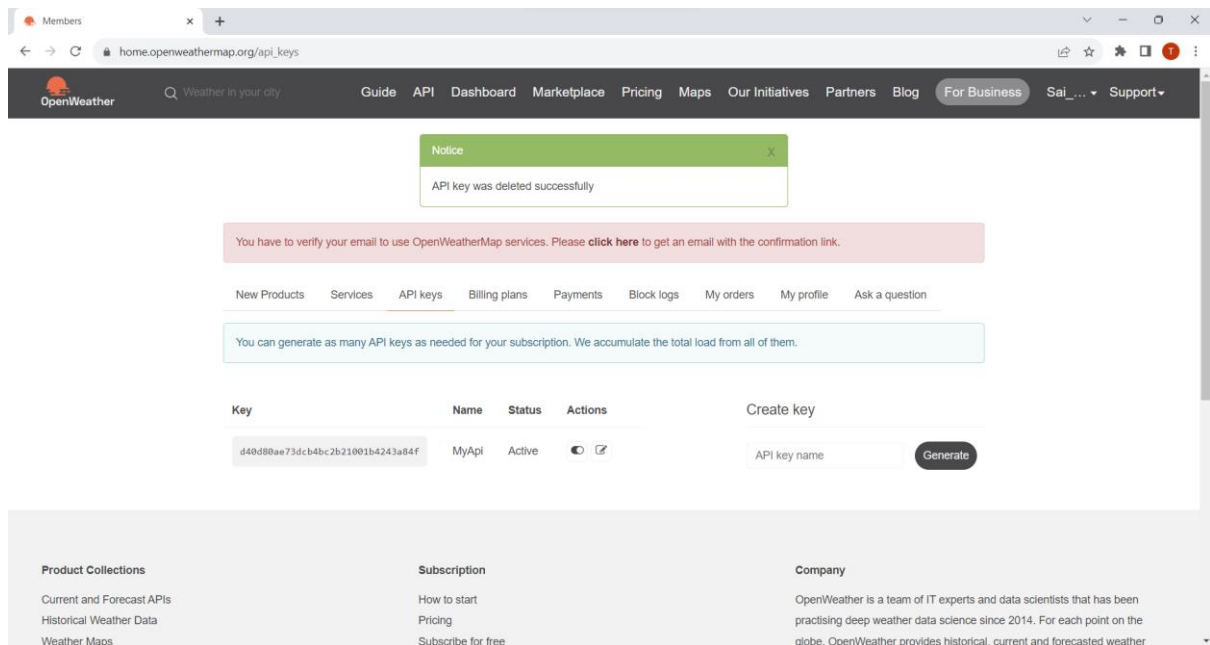
After opening the website enter your sign up details to log in.



The screenshot shows the OpenWeatherMap sign-in page. The browser address bar displays 'home.openweathermap.org/users/sign\_in'. The navigation bar includes links for Guide, API, Dashboard, Marketplace, Pricing, Maps, Our Initiatives, Partners, Blog, For Business, Sign In, and Support. The main content area features a 'Sign In To Your Account' form with fields for 'Enter email' and 'Password', a 'Remember me' checkbox, and a 'Submit' button. Below the form, there are links for 'Not registered? Create an Account.' and 'Lost your password? Click here to recover.' The footer contains three sections: 'Product Collections' (Current and Forecast APIs, Historical Weather Data, Weather Maps, Weather Dashboard, Widgets), 'Subscription' (How to start, Pricing, Subscribe for free, FAQ), and 'Company' (OpenWeatherMap description and location).

**FIG. 4.13 OPEN WEATHER MAP SIGN UP PAGE**

After you log in, go to API keys on the tabular column and then you will find your respective API key.



The screenshot shows the OpenWeatherMap API keys page. The browser address bar displays 'home.openweathermap.org/api\_keys'. The navigation bar includes links for Guide, API, Dashboard, Marketplace, Pricing, Maps, Our Initiatives, Partners, Blog, For Business, Sai\_..., and Support. A green notice box at the top states 'API key was deleted successfully'. Below this, a red box instructs the user to verify their email. The main content area features a 'Create key' section with a 'Generate' button. A table below the 'Create key' section lists API keys with columns for Key, Name, Status, and Actions. The table contains one entry with the key 'd40d80ae73dcb4bc2b21001b4243a84f', name 'MyApi', and status 'Active'. The footer contains three sections: 'Product Collections' (Current and Forecast APIs, Historical Weather Data, Weather Maps), 'Subscription' (How to start, Pricing, Subscribe for free), and 'Company' (OpenWeatherMap description and location).

**FIG. 4.14 API KEY**

## CHAPTER 5

### RESULT & DISCUSSION

Before running the code enter your API key in the java script file in the const apikey area, this will use your unique key to access all the weather data you want to use in your website and after running the code you get the front end part of the website.



**FIG 5.1 WEATHER APPLICATION**

Now enter any location you want to see the weather of, For example, I want to see the weather of Guntur so, I will enter Guntur on the search tab which will in turn access the weather data from the open weather API and show us the connected data.



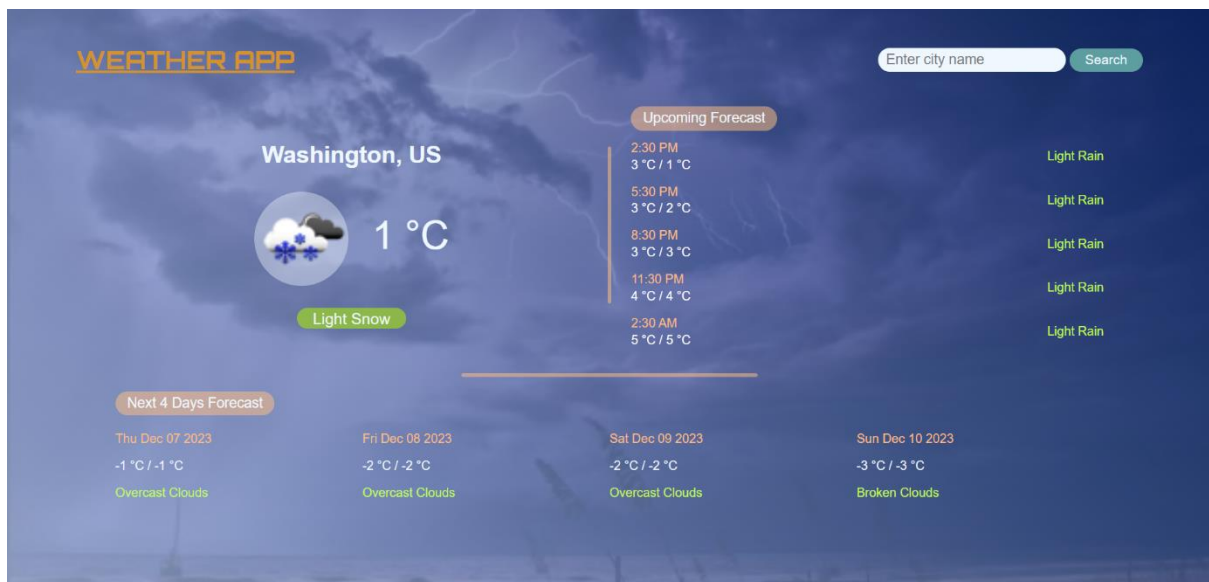
**FIG 5.2 WEATHER IN GUNTUR**

Similarly, let's see the weather of Delhi, India which shows that our website works for the cities that are available in India.



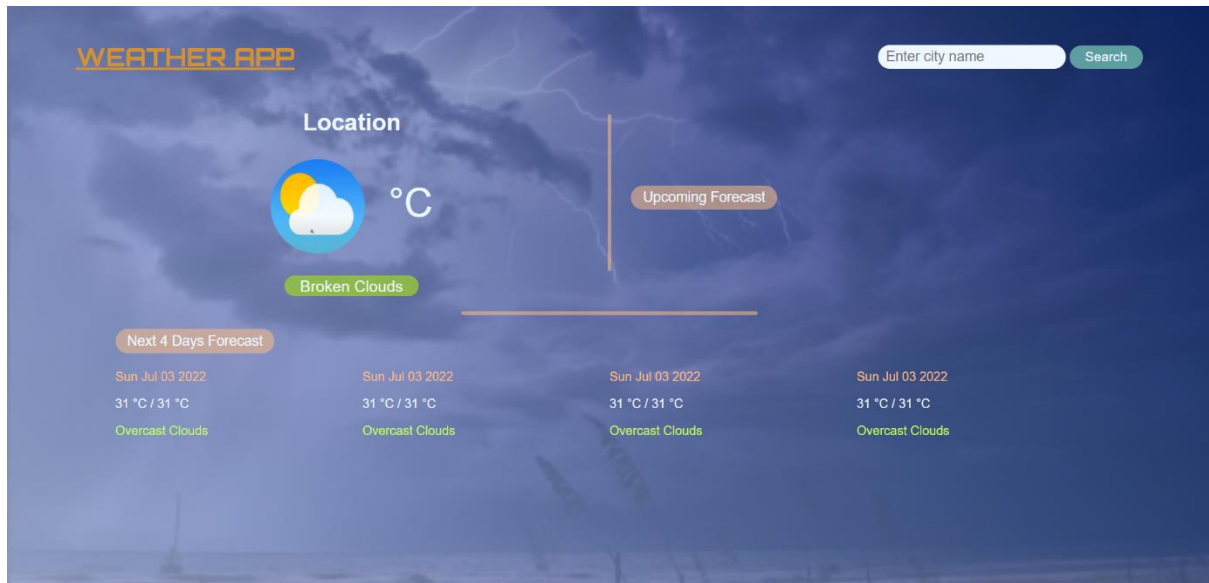
**FIG 5.4 WEATHER IN DELHI**

Similarly, let's see the weather of Washington, US which shows that our website works for cities that are available in the world.



**FIG 5.3 WEATHER IN WASHINGTON, US**

Lets see what happen when we enter the wrong location.



**FIG 5.4 SEARCH FOR A WRONG LOCATION**

In the above figure, we observe that it does not display any weather if we search for a wrong location.

## **CHAPTER 6**

### **CONCLUSION & FUTURE WORK**

#### **SUMMARY:**

The intension of the project is to just make a model of a weather forecast using web development and code it in a simple way so that one can know the weather of a location by accessing a API key. This application would be a part of the development platform of the technology.

#### **FUTURE WORK:**

In future projects, I will improve the design of my program by giving stylish themes and fonts.

Also, would compare the API data with a weather predictor which predicts weather with the data of the weather in that location in the past 5 years.

#### **CONCLUSION:**

Run the program, the google chrome window opens with your designed web page. Then enter the location of the place for which you want to see its weather. Now your web page shows the respective temperature and humidity of that particular area.

Hence, we have successfully created a website which displays the weather of any location entered at that particular time.

## REFERENCES

- Pickering, Heydon; Apps For All: Coding Accessible Web Applications(2014), Smashing Magazine GmbH.
- Pickering, Heydon; Inclusive Components(2018): The Book, Inclusive Components
- Pickering, Heydon; Inclusive Design Patterns(2016), Smashing Magazine.
- Thatcher, Jim et al; Constructing Accessible Web Sites(2002), Glasshaus.
- Thatcher, Jim et al; Web Accessibility: Web Standards and Regulatory Compliance(2006), friends of ED.