VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belagavi-590018

Mini Project Report On

INDRA-CAST: REALTIME WEATHER FORCASTING WEBAPP

Submitted by

GEDDADA NETHRANAND — **4SN21CS035**

MADAN M RAIKAR — **4SN21CS049**

COMPUTER SCIENCE & ENGINEERING

(Visvesvaraya Technological University)

Under the Guidance of

DR. JITHENDRA P R N

Associate Professor

Department of Computer Science & Engineering

SRINIVAS INSTITUTE OF TECHNOLOGY

(NAAC ACCREDITED)

MANGALURU-574143, KARNATAKA

2024 – 2025

SRINIVAS INSTITUTE OF TECHNOLOGY
(NAAC ACCREDITED)
MANGALURU -574143 – KARNATAKA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project entitled ***"INDRA_CAST: REALTIME WEATHER FORECASTING WEBAPP",*** *is an authentic record of the work carried out by*

GEDDADA NETHRANAND	**MADAN M RAIKAR**
4SN21CS035	**4SN21CS049**

as prescribed by Visvesvaraya Technological University, Belagavi, for VI Semester B.E. in Computer Science & Engineering during the year 2024-2025.

Dr. Jithendra P R N Project Guide	Prof. Alwyn Edison Mendonca Mini-project Coordinator	Dr. Suresha D Head of the Department

Dr. Shrinivasa Mayya D
Principal

Name of the Examiners **Signature with Date**

1.

2.

ABSTRACT

This web application integrates comprehensive weather data and mapping functionalities to provide users with accurate and real-time weather information. The application leverages advanced weather APIs to offer current conditions and forecasts. Additionally, it incorporates map integration to visually display geospatial data. Users can access detailed weather information for specific locations, enhancing their ability to plan and make informed decisions. The app's user-friendly interface ensures a seamless experience, catering to both casual users and weather enthusiasts. IndraCast also allows users to compare weather details such as temperature, wind speed, and humidity between two cities, providing a side-by-side analysis to facilitate more informed decision-making based on weather conditions in different locations.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any work would be incomplete without thanking the persons who made it perfect with their constant guidance and encouragement.

We take this opportunity to express my sincere thanks and indebtedness to our Project guide and mentor, **Dr. Jithendra P N**, Department of Computer Science and Engineering, for his support and guidance. His vision and suggestions throughout the project period have been fundamental in the completion of the project.

We would like to extend my deep sense of acknowledgement to our Project Co-Ordinator **Prof. Alwyn Edison Mendonca**, Assistant Professor, Department of Computer Science and Engineering, for his encouragement and timely advice provided to us during the project work.

We extend my warm gratitude to **Dr. Suresha D**, Head of the Department, Department of Computer Science and Engineering, for his constant support and advice that helped us to complete this project successfully.

We extremely grateful to our beloved Principal, **Dr. Shrinivasa Mayya D** for his encouragement to come up with new ideas and advice to express them in a systematic manner.

We also like to thank all Teaching & Non-teaching staff of Srinivas Institute of Technology, Mangalore for their kind co-operation during the course of the work.

Finally, we are extremely thankful to our family and friends who helped us in our work and made the project a successful one.

Geddada Nethranand

Madan M Raikar

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Indra_Cast is an innovative web application designed to provide comprehensive weather data and mapping functionalities. By leveraging advanced weather APIs and integrating dynamic map features, this application aims to deliver real-time weather information and geospatial data to users. Whether for casual inquiries or detailed weather analysis, Indra_Cast offers a user-friendly interface that caters to both everyday users and weather enthusiasts. The application supports a range of features, including current weather conditions, forecasts, and the ability to compare weather details between different locations.

1.1 Problem Statement

In an increasingly data-driven world, access to accurate and timely weather information is essential for various personal and professional decisions. Existing weather applications often lack intuitive interfaces or fail to integrate weather data with spatial context effectively. Additionally, users frequently face challenges in comparing weather conditions across different locations, which can hinder decision-making. Indra_Cast addresses these issues by providing a comprehensive solution that combines real-time weather data with interactive mapping capabilities, allowing users to seamlessly access and compare weather information across multiple locations.

1.2 Scope of the Project

The scope of Indra_Cast includes:

1. **Weather Data Integration:** Utilize advanced weather APIs to fetch real-time weather data, including temperature, humidity, wind speed, and weather descriptions.
2. **Map Integration:** Implement Google Maps to visually display geospatial data and provide users with interactive mapping features, including location search and visualization.
3. **User Interface:** Develop a user-friendly interface that enables users to:
    - View current weather conditions.

- Compare weather details between different locations through graphical representation.
- Access detailed weather forecasts and historical data.

4. **Feature Implementation:**
    - **Homepage:** Display current weather conditions and a search feature for users to enter their location.
    - **Compare Page:** Offer a side-by-side comparison of weather data between selected locations, visualized through charts and graphs.
    - **About Page:** Provide information about the application, its features, and the team behind it.
    - **Contact Page:** Include contact details, location information, and a map for user inquiries.
5. **Technical Requirements:**
    - **Backend:** Django framework for server-side logic and data management.
    - **Frontend:** HTML, CSS, JavaScript for user interface development and interaction.
    - **APIs:** Integration with weather APIs (e.g., OpenWeatherMap) and Google Maps API.
6. **Future Enhancements:** Potential future improvements may include additional features such as user accounts, personalized weather alerts, and expanded weather data options.

CHAPTER 2

REQUIREMENT SPECIFICATION

2.1 Functional Requirements

- **User Interface Navigation:**
  - Users should be able to navigate between pages: Home, Compare, About, and Contact.
- **Weather Information:**
  - The Home page should display weather information based on user input.
  - Show current weather conditions including temperature, humidity, wind speed, and weather description.
  - Provide visual representation (icons) for different weather conditions.
- **Map Integration:**
  - Integrate Google Maps to display a map of the specified location.
  - Handle scenarios where the map or location is not found.
- **Weather Comparison:**
  - The Compare page should allow users to compare weather conditions between two locations.
  - Display comparison results using charts.
- **About Page:**
  - Provide information about the application and the team behind it.
- **Contact Page:**
  - Display contact information and a map showing the location of your institution.
  - Provide contact details including phone numbers and email addresses.

2.2 Non-Functional Requirements

1. **Performance:**
   - The application should load quickly and respond promptly to user interactions.
2. **Usability:**
   - The interface should be user-friendly and intuitive.
   - Ensure that the application is responsive and works well on different devices.

3. **Security:**
    - Protect user data and ensure that sensitive information is handled securely.
    - Implement error handling to manage API failures and invalid user inputs gracefully.
4. **Scalability:**
    - The application should be designed to handle increased user loads and additional features in the future.
5. **Reliability:**
    - Ensure high availability of the application with minimal downtime.

2.3 Hardware Requirements

1. **Server:**
    - A web server (such as Apache or Nginx) to host the Django application.
    - Adequate server resources (CPU, RAM, and storage) depending on expected traffic.
2. **Client Devices:**
    - Users should be able to access the application from various devices, including desktops, tablets, and smartphones.

2.4 Software Requirements

1. **Operating System:**
    - Any OS that supports Django and its dependencies (e.g., Windows, macOS).
2. **Web Server:**
    - Apache or Nginx for deploying the Django application.
3. **Database:**
    - A database system supported by Django (e.g., PostgreSQL, MySQL, SQLite).
4. **Python:**
    - Python 3.x (Django's supported version).
5. **Django:**
    - Django framework for backend development.

2.5 Software Tools Used

1. **Django:**
   - Backend framework used for building the application.
2. **HTML/CSS:**
   - For structuring and styling the web pages.
3. **JavaScript:**
   - For dynamic client-side interactions and map integration.
4. **Chart.js:**
   - For visualizing weather comparison data.
5. **Google Maps API:**
   - For integrating maps and displaying location data.

2.5.1 Front End Tool/User Interfaces

1. **HTML/CSS:**
   - Used for creating the structure and styling of the web pages.
2. **JavaScript:**
   - For interactive elements and handling API responses.
3. **Boxicons:**
   - For iconography in the user interface.
4. **Bootstrap Icons:**
   - Used for various icons on the Contact page.

2.5.2 Back End Database Used

1. **Django:**
   - Manages server-side logic, API integration, and data handling.
2. **Views and URLs:**
   - Define how different pages are rendered and the routing of requests.
3. **Static Files:**
   - Includes CSS, JavaScript, and image files used for styling and functionality.

2.5.3 Development Environment

1. **IDE/Editor:**
   - An Integrated Development Environment (IDE) or code editor like VS Code or PyCharm.
2. **Version Control:**
   - Git for managing code versions and collaboration.
3. **Local Server:**
   - Django's built-in development server for testing and debugging.
4. **Virtual Environment:**
   - Use a virtual environment to manage project dependencies.

CHAPTER 3

SYSTEM DESIGN

Indra-Cast is a web application designed to provide real-time weather information with an intuitive interface. It features a dashboard for quick weather updates, a comparison page for analyzing weather conditions across multiple locations side-by-side, and a detailed page for in-depth meteorological data. Using HTML, CSS, and JavaScript, the web app integrates with APIs like OpenWeatherMap for weather data and Mapbox for interactive maps. This setup ensures users can view current weather conditions, forecasts, and historical weather data, compare weather across different locations, save comparison results as CSV files, and manage their favorite locations, all directly within their web browser.

3.1 Flow Chart

The flowchart for the Indra-Cast Weather Forecasting Webapp starts with user interaction where they input a location or allow location access. The web app then requests weather data from an API like OpenWeatherMap, which is processed and displayed to the user in real-time. If the user wants to compare weather conditions across multiple locations, the web app fetches and shows data for all selected locations side-by-side. The process concludes when the user finishes their interaction or closes the web app.

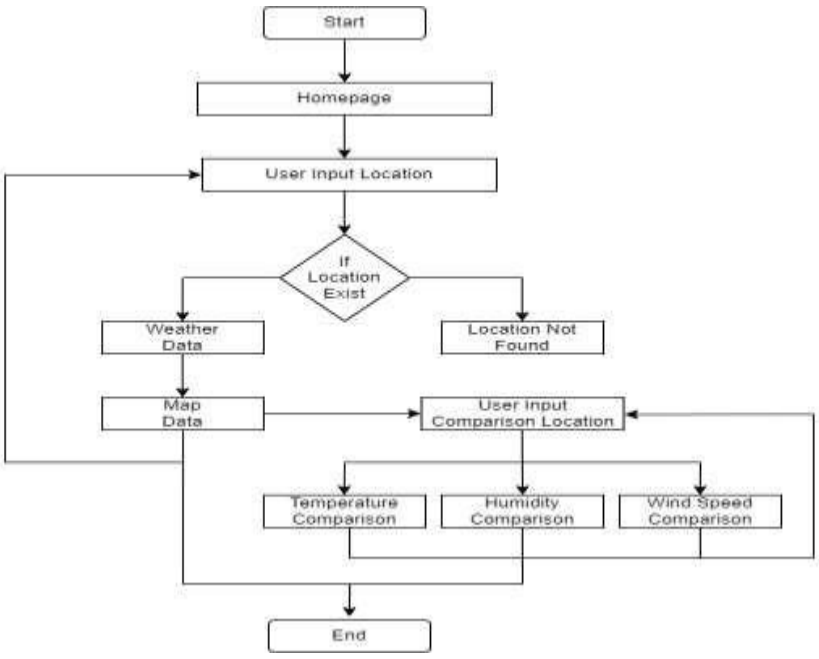

Fig 3.1: *Flowchart of Indra-Cast*

3.2 Use case Diagram

A use case diagram for the Indra-Cast Weather Forecasting Webapp would visually represent the interactions between users (actors) and the system, illustrating the system's functionalities from the user's perspective. In the diagram, actors such as "User" and "Admin" are depicted as stick figures, while the system's functions are shown as ovals, called use cases. These use cases include actions like "Input Location," "Fetch Weather Data," "Display Weather Data," "Compare Locations," "Save Comparison Data," and "Manage Favorite Locations." The diagram also includes relationships: associations, which connect actors to use cases to show interactions; include relationships, indicating that one use case is always part of another (e.g., "Fetch Weather Data" includes "Retrieve Weather Information"); and extend relationships, showing optional or conditional behavior (e.g., "Compare Locations" extends "Fetch Weather Data" if additional location details are requested). The system boundary, a rectangle surrounding the use cases, defines the scope of the system and separates it from external actors. This diagram helps clarify how users will interact with the system and what functionalities are available.

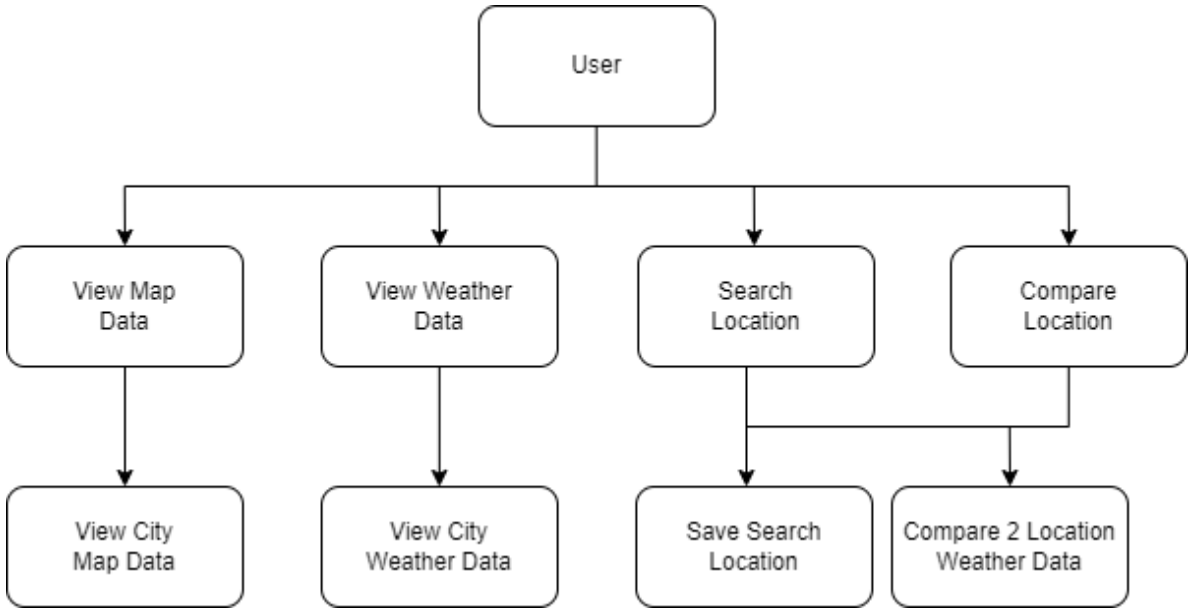

Fig 3.2: *Use case diagram of Indra-Cast*

CHAPTER 4

IMPLEMENTATION

Overview: IndraCast is a weather comparison web application that allows users to fetch and compare weather data for multiple cities. Built using Django, it handles user requests, fetches data from APIs, and displays the information through dynamic HTML templates.

4.1 Working of Indra-Cast

HomePage (Index.html)

```
{% load static %}
<head>
   <title> {% block title %} Indra_Cast {% endblock %} </title>
   <link rel="stylesheet" href="{% static 'style.css' %}">
</head>
<body>
   <nav class="navbar">
      <div class="navcontainer" style="width: 500px;">
      </div>
   </nav>
   <div class="container">
      <div class="search-box">
         <i class='bx bxs-map'></i>
         <form class="hello"  action=""  >
            <input class="inputs" type="text" placeholder="enter your location">
         </form>
         <button class="bx bx-search"></button>
      </div>
      <div class="weather-box">
      </div>
      <div class="weather-details">
      </div>
      <div class="not-found">
      </div>
   </div>
   <div class="map_container">
      <div id="map"></div>
      <script>
      </script>
      <div class="mapnot-found">
      </div>
   </div>
   <script>
```

```
    </script>
    <script src="{% static 'script.js' %}"></script>
</body>
</html>
```

Compare Page (Compare.html)

```
{% load static %}
<head>
    <title>Indra_Cast- Compare</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>

<body style="overflow-y: scroll;">
    <nav class="navbar">
        <div class="navcontainer" style="width: 500px;">
            <a href="#" class="navbar-brand">INDRA_CAST</a>
        </div>
    </nav>

    <section class="py-2 text-center container" style='align-items: center; width: 98%; height: 100%; margin: 5px auto; margin-top: 100px; '>
            <div class="compare">
                <div class="bar" style="height: 450px; width: 500px;">
                    <canvas id="myChart1"></canvas>
                </div>
            </div>

            <div class="compare">
                <div class="bar" style="height: 450px; width: 500px;">
                    <canvas id="myChart2"></canvas>
                </div>
            </div>
            <div class="compare">
                <div class="bar" style="height: 450px; width: 500px;">
                    <canvas id="myChart3"></canvas>
                </div>
            </div>
        </div>
    </section>

    <script src="{% static 'myscript.js' %}"></script>
</body>
</html>
```

Fetching Backend (script.js)

```
const fetchData = async () => {
    const APIKey = '05f73b7b5b8440f0f61fec0f584d0cd9';
    const city = document.querySelector('.search-box input').value;
```

```
    localStorage.setItem("city", city)
    if (city == '')
        return;
    const res = await
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&units=metric&appid=${
APIKey}`);
    const json = await res.json();
    console.log("json")
    if (json.cod == '404') {
    }
    console.log(json);
    if (cityHide.textContent === city) {
        return;
    } else {
        weatherDetails.classList.add('active');
        error404.classList.remove('active');
        setTimeout(() => {
        }, 2500);
        switch (json.weather[0].main) {
            case 'Clear':
                image.src = clear;
                break;
            case 'Rain':
                image.src = rain;
                break;
            case 'Clouds':
                image.src = cloud;
                break;
            default:
                image.src = cloud;
        }
        document.querySelectorAll('.active-clone').forEach(el => el.remove());
        elCloneInfoWeather.id = 'clone-info-weather';
        elCloneInfoWeather.classList.add('active-clone');
        elCloneInfoHumidity.id = 'clone-info-humidity';
        elCloneInfoHumidity.classList.add('active-clone');
        elCloneInfoWind.id = 'clone-info-wind';
        elCloneInfoWind.classList.add('active-clone');
        setTimeout(() => {
    }
    return json;
}
const initMap = async (json) => {
    if (json.cod == '404') {
        if (mapContainer.firstChild) {
          // Remove existing map data
        }
        maperror404.classList.add('active');
        return;
    }
```

```
    else {
        maperror404.classList.remove('active');
        setTimeout(() => {
            mapContainer.classList.remove('active');
        }, 2500);
        const { Map } = await google.maps.importLibrary("maps");
        const map = new Map(mapContainer, {
        });
        new google.maps.Marker({
        });
    }
}
const submit = document.querySelector(".hello");
submit.addEventListener("submit", async (e) => {
});
earch.addEventListener('click', async () => {
    const json = await fetchData();
    initMap(json);
});
```

Comparision Backend (myscript.js)

```
let chart1, chart2, chart3;
const fetchData2 = async (city) => {
    const APIKey = '05f73b7b5b8440f0f61fec0f584d0cd9';
    if (city === '') return;
    const res2 = await
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&units=metric&appid=${
APIKey}`);
    const json2 = await res2.json();
}
const fetchData3 = async (city) => {
    const APIKey = '05f73b7b5b8440f0f61fec0f584d0cd9';
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&units=metric&appid=${
APIKey}`);
}
const fetchData4 = async (json2, json3) => {
    const site1 = json2?.name;
    const site2 = json3?.name;
    const temp1 = json2?.main?.temp;
    const temp2 = json3?.main?.temp;
    console.log(site1);
    console.log(hud1);
    if (chart1) chart1.destroy();
    var ctx1 = document.getElementById('myChart1').getContext('2d');
    chart1 = new Chart(ctx1, {
        type: 'bar',
        data: {
        },
```

```
    if (chart2) chart2.destroy();
    var ctx2 = document.getElementById('myChart2').getContext('2d');
    chart2 = new Chart(ctx2, {
        type: 'doughnut',
        data: {
            labels:  [site1,  site2],
    if (chart3) chart3.destroy();
    var ctx3 = document.getElementById('myChart3').getContext('2d');
    chart3 = new Chart(ctx3, {
        type: 'bar',
const submit = document.querySelector(".hello");
submit.addEventListener("submit", async (e) => {
    if (!city2 || !search) {
        console.error("Both cities must be provided");
        return;
    }
    if (!json2 || !json3) {
        console.error("Failed to fetch weather data for one or both cities");
        return;
    }
    fetchData4(json2, json3);
    const graphs = document.getElementById("graphs");
    graphs.style.display = "flex";
    setTimeout(() => {
        graphs.classList.add("show");
    }, 0);
});
```

Working of the Project

1. **URL Routing**:
    - **Project URLs**: Routes requests to the appropriate app.
    - **App URLs**: Routes specific weather-related requests to views.
2. **Views**:
    - Handle HTTP requests.
    - Fetch weather data using APIs.
    - Compare weather data for multiple cities.
    - Render templates with the fetched data.
3. **Templates**:
    - Render the user interface.
    - Display weather information and comparisons.

CHAPTER 5

SCREEN SHOTS

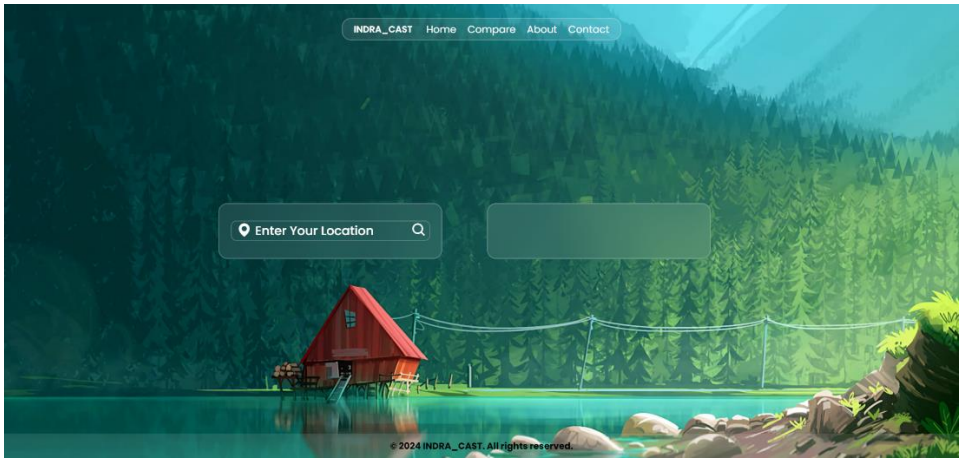

Fig 5.1: *Home Page*

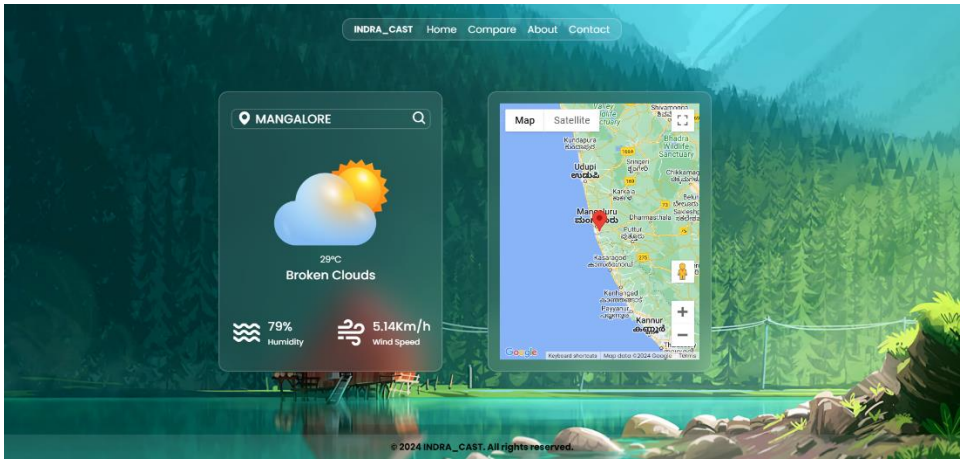

Fig 5.2: *Weather Forecast*

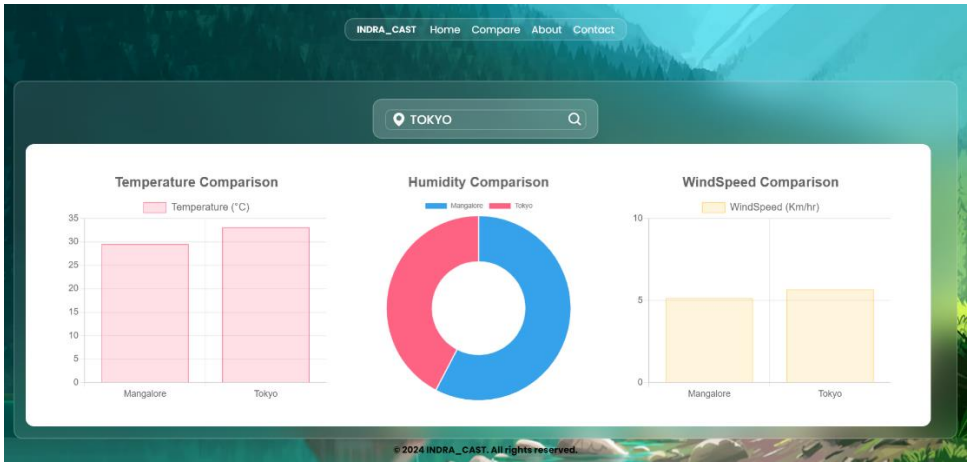

Fig 5.3: *Compare Page*

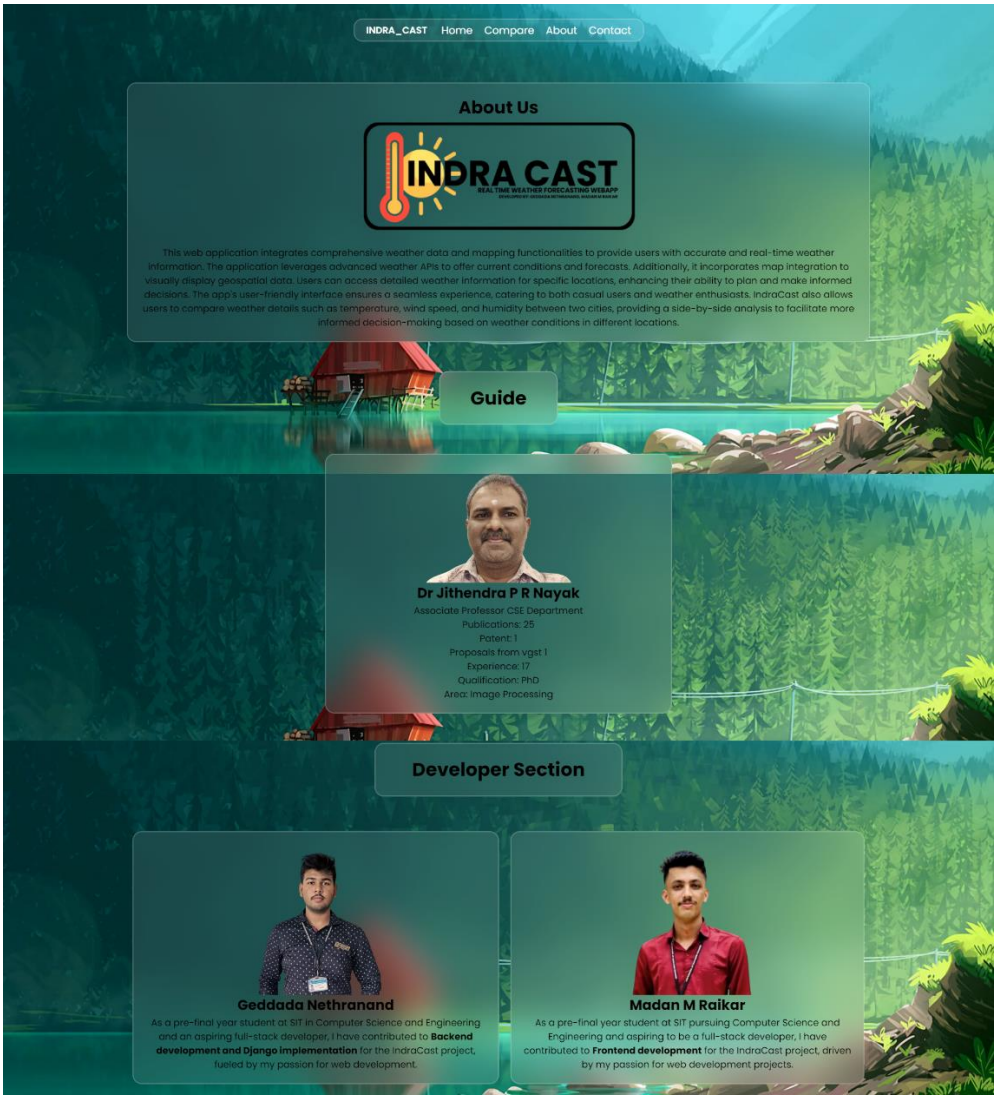

Fig 5.4: *About Page*

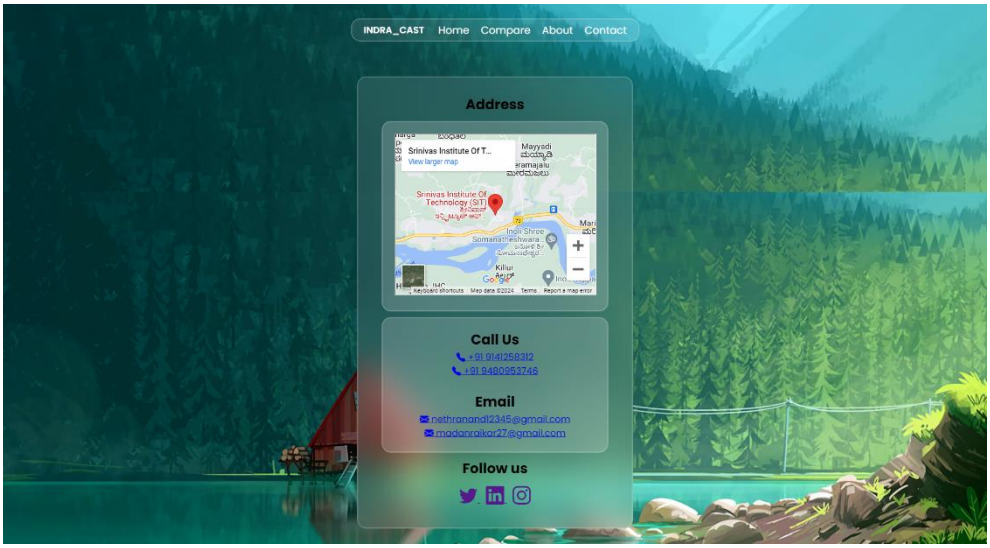

Fig 5.5: *Contact Page*

CONCLUSION AND SCOPE FOR FUTURE WORK

Indra-Cast is a robust web application that provides users with real-time weather information, empowering them to make informed decisions based on accurate and up-to-date data. By integrating interactive maps and pulling detailed weather data such as temperature, humidity, precipitation, and wind speed from the OpenWeatherMap API, users can plan their activities with confidence. Also has ability to compare weather conditions across locations. Built with Django, HTML, CSS, and JavaScript, Indra-Cast offers a responsive and visually appealing interface, ensuring seamless access across various browsers. The web app empowers users to stay informed and prepared for any weather conditions conveniently.

Future improvements for Indra-Cast include adding advanced tools and machine learning for more precise forecasts and insights. Expanding data sources to include more global weather services and climate data will attract a wider audience. Personalization features will let users customize their dashboards, set weather alerts, and get tailored advice. Creating a mobile version will provide real-time weather information on the go. Adding social features, like sharing options and community forums, will help users discuss weather patterns and preparedness strategies. Continuous optimization will make data fetching faster and interactions smoother. Stronger security measures will protect user data and ensure safe API interactions.

BIBLIOGRAPHY

[1] Django Documentation. Django Software Foundation. Available at: https://docs.djangoproject.com/en/stable/

[2] OpenWeatherMap API Documentation. OpenWeatherMap. Available at: https://openweathermap.org/api

[3] Python Requests Library Documentation. Available at: https://docs.python-requests.org/en/master/

[4] Bootstrap Documentation. Available at: https://getbootstrap.com/docs/5.1/getting-started/introduction/

[5] W3Schools HTML Tutorial. Available at: https://www.w3schools.com/html/

[6] JavaScript Documentation. MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[7] Django REST Framework Documentation. Available at: https://www.django-rest-framework.org/

[8] GitHub. Repository Hosting Service. Available at: https://github.com/

[9] Stack Overflow. Community-based Questions and Answers. Available at: https://stackoverflow.com/