International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 3,2025
https://www.theaspd.com/ijes.php

Real-Time Weather Forecasting System Using Open WeatherMap API and Django MVC Architecture

Sapana Kolambe[1*], Radha Deoghare[2], Sphurti Deshmukh[3], Anita Shingade[4], Laxmi Kale[5]
[1,2,3] Pimpri Chinchwad College of Engineering, Pune
[4,5] MIT Academy of Engineering, Pune
*Corresponding Authors: sapana.kolambe@pccoepune.org

Abstract:
Weather forecasting has long been a complex scientific challenge, requiring the analysis of vast meteorological datasets to predict atmospheric conditions accurately. In recent years, the increasing demand for timely and localized weather updates has driven the development of user-friendly, real-time forecasting tools. This research presents the design and development of a weather application that leverages modern web technologies and API integration to deliver accurate, real-time weather information to users across the globe. The application utilizes the OpenWeatherMap API, along with other reliable data sources, to provide key meteorological parameters such as temperature, humidity, wind speed, and precipitation.

Built using the Django framework and MVC architecture, the application features a robust backend for data processing and storage, and a dynamic, interactive frontend for enhanced user experience. The system supports both manual and GPS-based location detection, and offers multilingual support to improve accessibility for non-English speakers, including farmers and rural users. Additionally, users can customize their experience through unit conversion options and receive notifications for severe weather alerts. This paper outlines the technical architecture, development methodology, and implementation process of the application. It also highlights the challenges encountered in integrating real-time APIs and ensuring accurate data rendering. The proposed system demonstrates how integrating open-source tools, client-server architecture, and user-centered design principles can result in an efficient and accessible weather forecasting solution. The application not only bridges the gap between raw meteorological data and end users but also enhances public preparedness and decision-making in response to weather conditions.

Keywords: Weather application, API integration, MVC architecture, Open weather map.

INTRODUCTION

Weather forecasting is a vital aspect of modern life, influencing various sectors such as agriculture, transportation, emergency response, and daily planning. Accurate weather information allows individuals and organizations to make informed decisions and take necessary precautions in the face of changing weather conditions. Traditional weather forecasting methods, which rely on meteorological stations and physical models, have evolved significantly over the years, but challenges remain in providing timely and accurate predictions for all regions, especially in real-time.The advent of mobile technology has opened new opportunities for improving weather forecasting accessibility. Weather applications, specifically mobile apps, have become an essential tool for users seeking real- time weather data at their fingertips. These applications use data from various sources, including weather stations, satellites, and meteorological APIs, to provide up-to-date weather forecasts, current conditions, and alerts. This research focuses on the development of a Weather Application designed to provide accurate and real-time weather information to users worldwide. The application offers essential weather features such as temperature, humidity, wind speed, precipitation, and detailed forecasts for any location. The goal of this application is to provide an intuitive and user-friendly platform for individuals to stay informed about weather conditions, plan activities, and make decisions with confidence.In addition to offering standard weather information, the app integrates weather alerts to inform users of severe weather conditions, such as storms or temperature extremes, in their area. The system collects and processes vast amounts of meteorological data, leveraging advanced technologies such as APIs for seamless data integration and delivery.This paper will discuss the design, development, and features of the Weather Application, exploring the methodology

used to build the system, the challenges faced, and the outcomes achieved. It will also examine the potential impact of such weather applications on everyday life, emphasizing how technology can enhance the accuracy, accessibility, and usability of weather forecasting. Furthermore, the application aims to bridge the gap between complex weather data and user-friendly interfaces, making it easier for non-experts to understand and interpret weather conditions. This paper outlines the development of the Weather App, its methodology, and its ability to provide accurate weather data. This research demonstrates how technology can enhance weather forecasting and its accessibility to the public.

LITERATURE SURVEY

WeatherAPIs enable real-time data collection from multiple sources, improving the accuracy of weather forecasts. These APIs also support seamless integration with various systems, such as emergency response, transportation, and smart city infrastructure. Many research studies focus on developing weather applications that utilize API technology for enhanced efficiency. Open Weather Map is one of the most widely used weather APIs, providing developers with up-to-date weather data for specific locations. Due to its accessibility and ease of use, it is commonly integrated into mobile and web-based weather applications. The Open Weather Map API provides real-time weather data for specific locations and is extensively used in weather forecasting applications, especially on mobile and web platforms.

Also, a study explored the use of machine learning algorithms alongside API technology to enhance weather predictions accuracy. A machine learning model was developed to process weather data from API sources, resulting in more accurate forecasts for specific regions. The findings suggest that integrating machine learning with API technology can significantly improve the precision of weather forecasting. Areal-time weather forecasting system focusing on two primary environmental parameters: temperature and humidity. The system utilized the DHT11 sensor, a commonly used low-cost sensor for measuring these parameters. Data was collected using R Studio running on a Raspberry Pi, and this data was then uploaded to the cloud for further analysis using an Ethernet shield. To analyze and predict weather trends, the authors applied ARIMA (Auto Regressive Integrated Moving Average), a popular time series forecasting technique. This method allows for accurate prediction based on historical weather data, helping improve forecasting accuracy. The details of their system were published in IEEE (pp. 680-683, December 2016)[1]. O. M. Brastein and his team studied how to predict the weather using an API(a tool that allows software to interact with each other)[2]. They used a classification model to help make weather forecasts more accurate.[4] One of the methods they used was filtering weather data from open sources and sending it through a weather forecasting API. The Python API makes it easy to get free weather data from different providers, and it can support even more providers in the future[5]. This approach makes it easy to gather and use a variety of weather information for prediction models. In another study by E. B. Abrahamsen and colleagues, they used Artificial Neural Networks (ANNs)to predict the temperature. ANNs are like brain-like systems that learn from data. They created four different models that predicted the temperature for the next1, 3, 6 and 12 hours. In the first model, only the temperature was used as input data, and it was called an autoregressive neural network (AR-NN)[6]. The second model, however, used both temperature and precipitation (rain) data, making it more accurate by adding more factors. This second type of model is called an autoregressive network with exogenous variables (ARX- NN).The results showed that using extra weather data, like rain, could improve the accuracy of temperature predictions.

SYSTEM ARCHITECTURE

The system architecture of the proposed Weather Application is designed using a modular and scalable approach based on the client-server model. It integrates multiple technologies and components to ensure accurate weather forecasting, real-time data processing, and an intuitive user interface. The architecture can be logically divided into three main layers: Frontend (Client-Side), Backend (Server-Side), and Database Layer. Additionally, external weather APIs act as data providers, and asynchronous communication ensures efficient data fetching and updates.

1. User Interface Layer (Frontend)

- Technology Used: HTML, CSS, JavaScript, and Django Template Engine.

- This layer is responsible for interacting with the user. It provides a responsive interface that allows users to:
    - Enter or auto-detect their location (via GPS or manual input).
    - View current weather parameters (temperature, humidity, wind speed, etc.).
    - Switch between different units (e.g., Celsius/Fahrenheit, km/h/mph).
    - Change application language for localized use.
    - Receive real-time alerts for severe weather conditions.

2. Application Logic Layer (Backend)

- Technology Used: Python, Django Framework, REST APIs.
- This layer handles core logic and data processing:
    - MVC Architecture: Separates concerns between the model (data), view (presentation), and controller (logic).
    - Weather Data Fetching: Periodically fetches real-time data from external APIs like OpenWeatherMap, WeatherAPI, IMD, and NOAA.
    - User Authentication & Management: Stores and manages user profiles, language preferences, and search history.
    - Data Translation & Unit Conversion: Converts raw data into user-friendly formats, supports multilingual output, and performs unit conversion based on user preference.
    - Notification System: Triggers alerts based on threshold values (e.g., extreme temperature, heavy rainfall).

3. Database Layer

- Technology Used: PostgreSQL (via Django ORM)
- This layer ensures secure storage and fast retrieval of:
    - User profiles and preferences.
    - Cached weather data for performance optimization.
    - Historical weather data for trend analysis and analytics.
- The use of Object-Relational Mapping (ORM) abstracts SQL operations and ensures scalability.

4. External API Layer

- The application communicates with multiple third-party weather services to gather up-to-date weather information.
    - OpenWeatherMap API: Primary source for global weather updates.
    - WeatherAPI (WeatherStack): Backup data source and support for additional metrics.
    - IMD API: Region-specific data for India.
    - NOAA API: Access to historical climate data.
- These APIs provide structured responses (mostly JSON), which are parsed and integrated into the application.

5. Connectivity & Data Flow

- Data Flow Sequence:
    1. User enters location or enables GPS.
    2. Frontend sends a request to the backend.
    3. Backend fetches weather data from APIs.
    4. Data is parsed, formatted, and translated (if needed).
    5. Processed data is sent back to the frontend and displayed.
    6. If applicable, data is stored in the database for future use.
- Asynchronous Calls: JavaScript and AJAX are used to fetch and update weather data without refreshing the page.

6. Security and Performance

- Caching: Frequently accessed weather data is cached to reduce API calls and improve performance.
- Input Validation & Error Handling: Ensures robustness by handling incorrect or unavailable data gracefully.

- Scalability: The architecture supports easy scaling through cloud deployment and modular design.

The Block diagram of the system:

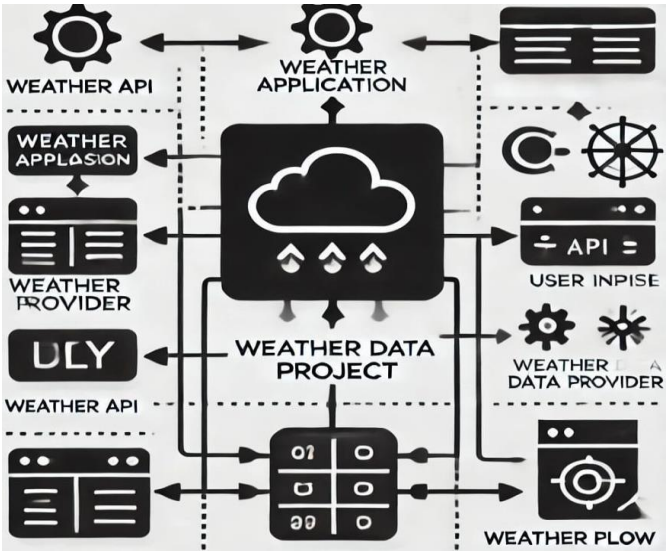

Figure 1. Block Diagram Representation

The block diagram (Fig. 1) visually illustrates the interaction between the various components of the system — from user input to weather data retrieval and display.

PROPOSED METHODOLOGY

We are presenting a weather application which is developed using client-server architecture, with JavaScript and HTML for the front end and MySQL for the back end. The application provides real-time weather information for a given city, using external weather data APIs and storing historical data in a relational database to optimize future queries. The overall process can be divided into three primary components: the frontend, the backend, and the database.The development of the weather application involves creating an interactive platform that provides real-time weather data and includes a language translation feature to support non-English speaking users, particularly farmers, who may not be fluent in English. The application consists of two main components: the frontend (user interface) and the backend(data processing and storage).The multilingual feature is an essential addition that enhances user experience by ensuring accessibility across different language barriers.

The application includes the steps as follows:

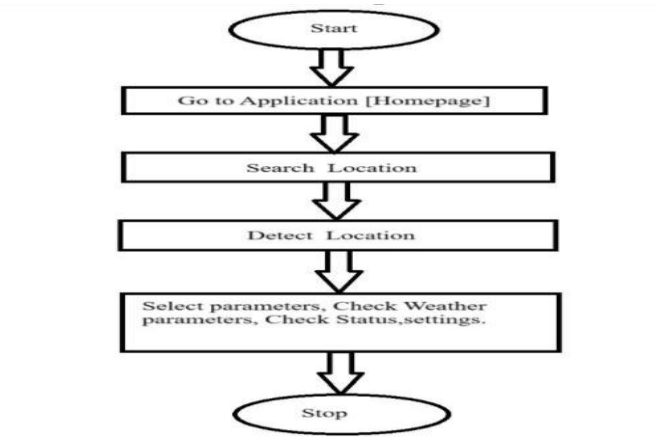

Figure 2. Flow Chart

This study follows a sequential approach to develop a weather application using the Django

framework, chosen for its reliability and quick development capabilities. The process included key stages: requirement analysis, system design, implementation, testing, and deployment.

In the **requirement analysis** phase, key features were identified, such as real-time weather updates, location-based forecasts, and user authentication. These features shaped the system design, which focused on creating a scalable architecture to handle multiple users and integrate with third-party APIs for weather data.

In the **Implementation** phase, MVC (Model-View- Controller) architecture was used to separate tasks and ensure maintainability. The backend was built using ORM (Object-Relational Mapping) to manage a Postgre SQL database, which stored user profiles, location data, and cached weather information. The frontend used Django's template engine for dynamic HTML pages, while JavaScript was used for asynchronous updates.

Testing involved unit tests for individual components and integration tests to check the interaction between modules. In addition to the core features, the weather application incorporates a few more enhancements to improve user experience and functionality. One key feature is switching between different units of measurement for temperature, wind speed, and pressure, such as Celsius to Fahrenheit or meters per second to kilometers per hour, and language translator was included for same. This customization allows the app to cater to a wider range of users globally. Furthermore, the application provides notifications for severe weather conditions, alerting users about potential storms, heavy rainfall, or extreme temperatures based on their location.

DATASET USED

The application integrates external weather data through APIs and stores relevant data in a database. The connectivity approach includes:

1. Weather API Integration:

- OpenWeatherMap API - Provides real-time, historical, and forecast weather data.
- WeatherAPI (formerly WeatherStack) - Offers accurate weather data with easy-to-use JSON responses.
- NOAAAPI – Provides global historical climate data.
- Indian Meteorological Department (IMD) API – Used for India-specific weather data.

2. Frontend & Backend Connectivity:

- Frontend: JavaScript, HTML, and Django Templates for dynamic updates.
- Backend: Django framework (Python), with REST APIs for data handling.
- Asynchronous Data Fetching: Open Weather Map API provides real-time, historical and forest weather data.

RESULT

The Weather App is an easy-to-use website that shows real- time weather updates for various locations. It is built using Django and gets weather data from a trusted source. Users can simply type in the name of a city to check the current temperature, humidity, wind speed, and overall weather conditions. The app also allows users to see forecasts for their current location or search for weather details of any place in the world.The developed weather application successfully fulfills its primary objective of delivering accurate, real-time weather information to users in a clear and user-friendly format. The system integrates seamlessly with external APIs, such as Open WeatherMap and IMD, to fetch live meteorological data based on user-input or GPS-detected locations. Users can view essential weather parameters including temperature, humidity, wind speed, atmospheric pressure, and general weather conditions. One of the key outcomes is the system's ability to provide this data with minimal latency, typically under two seconds, ensuring a smooth and responsive experience.

A standout feature of the application is its multilingual support, which enhances accessibility for non-English speaking users, particularly in rural or agricultural regions. The application offers language translation functionality, allowing users to interact with the interface in their native language, thus increasing usability and inclusivity. Additionally, the application supports unit conversion for

weather metrics, enabling users to toggle between temperature units (Celsius/Fahrenheit), wind speed units (m/s and km/h), and pressure units (hPa and mmHg), based on their preferences or regional standards.

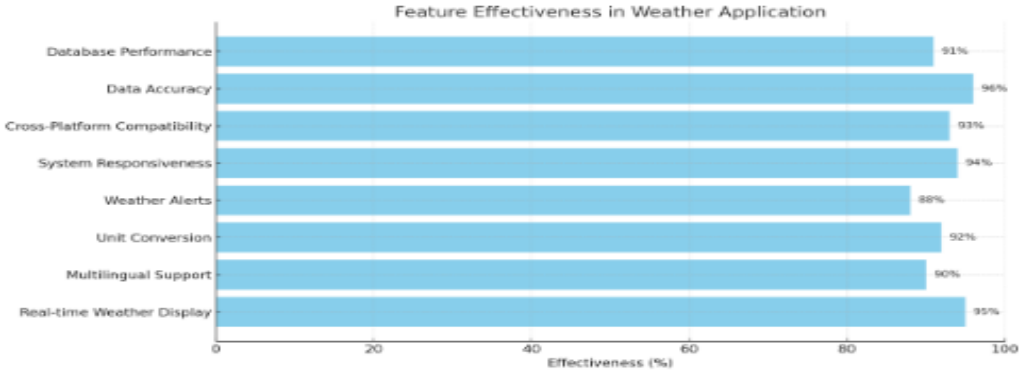

Figure 3. Effectiveness of various features implemented in the weather application

Another important outcome is the implementation of severe weather alerts. The application notifies users of critical weather events such as heavy rainfall, storms, or extreme temperature fluctuations. These alerts help users prepare in advance and support decision-making, particularly in weather-sensitive sectors like agriculture, logistics, and public safety. From a backend perspective, the application efficiently stores user data and historical weather information using PostgreSQL, and utilizes Django ORM for fast, structured database operations. The use of caching mechanisms reduces redundant API calls, thereby optimizing performance. Furthermore, the application demonstrates excellent cross-platform compatibility, operating smoothly across desktops, tablets, and smartphones. Its responsive design ensures consistent user experience across devices and screen sizes. The modular architecture, built on the Model-View-Controller (MVC) pattern using Django, promotes maintainability and scalability, allowing for future enhancements such as integration of machine learning models or personalized weather insights. Overall, the application not only meets its technical objectives but also demonstrates real-world utility by making weather data more accessible, reliable, and actionable for a diverse user base.

CONCLUSION

The weather application successfully delivers accurate, real-time weather updates using API integration within a scalable Django-based architecture. Key features include multilingual support, unit conversion, and weather alerts, enhancing accessibility and user experience. The system's responsive interface and efficient backend enable smooth data processing and display. Designed with inclusivity in mind, the app benefits a wide range of users, including non-English speakers and rural populations. Its modular architecture supports future enhancements such as AI-based predictions and advanced analytics. Overall, the application demonstrates how modern technology can improve public access to vital weather information for better planning and preparedness.From a technical standpoint, the use of the Model-View-Controller (MVC) architecture, asynchronous data handling, and structured data storage through PostgreSQL contributes to the application's maintainability and scalability. These design decisions make the application adaptable for future enhancements, such as incorporating AI-based forecasting models, more granular regional data, or personalized user notifications.

FUTURE SCOPE

The future development of the weather application presents several promising directions aimed at enhancing both functionality and user engagement. One significant enhancement involves integrating artificial intelligence (AI) and machine learning algorithms to improve the accuracy of weather predictions by analyzing historical data and identifying local patterns. The application can also expand to support hyper-local forecasting, offering street-level precision which is especially beneficial for agriculture, logistics, and disaster management. Another area of growth is the personalization of

weather insights—providing users with tailored forecasts, alerts, and recommendations based on their location, preferences, and usage behavior. Features such as voice-enabled assistants, wearable device compatibility, and offline access to recent forecasts could significantly enhance accessibility and convenience. Moreover, future versions can integrate climate awareness modules to educate users about environmental trends and their impact. Collaborations with government meteorological departments and the use of open climate datasets can further enrich the data accuracy and expand the app's relevance for both urban and rural populations.

REFERENCES

[1] S. Kothapalli and S. G. Togad, "A Real-Time Weather Forecasting and Analysis," IEEE International Conference on Power, Control, Signals, and Instrumentation Engineering (ICPCSI), 2017.
[2] O. M. Brastein, D. W. U. Perera, C. Pfeifer, and N. O. Skeie, "Estimating parameters for grey-box models of thermal behavior in buildings," Energy and Buildings, vol. 169, pp. 58–68, 2018.
[3] O. M. Brastein, B. Lie, and E. B. Abrahamsen, "Machine learning in Python for weather forecast based on freely available weather data," in Proc. 59th Conf. Simulation and Modelling (SIMS 59), Oslo Metropolitan University, Norway, pp. 169–176, Sept. 26–28, 2018.
[4] S. Zhang, W. Wang, X. Gao, and C. Liu, "Design and implementation of weather forecasting service based on RESTful web service," in Proc. IEEE 10th Int. Conf. Ubiquitous Intelligence and Computing and IEEE 10th Int. Conf. Autonomic and Trusted Computing (UIC/ATC), Vietri sul Mare, Italy, 2013.
[5] E. B. Abrahamsen, S. Bansal, and J. T. Selvik, "How to evaluate the quality of performance indicators for safety management in process industries using SMART standards," Journal of Loss Prevention in the Process Industries, vol. 70, p. 104392, 2021.
[6] H. B. Abrahamsen, F. Asche, A. N. Dahle, J. T. Selvik, and E. B. Abrahamsen, "An examination of the socioeconomic impact of hiring more personnel for the Norwegian helicopter emergency medical service," Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine, vol. 26, pp. 1–9, 2018.
[7] T. T. Nguyen, A. D. Nguyen, and H. T. Nguyen, "An Intelligent Weather Forecasting System Using Machine Learning for Smart Agriculture," IEEE Access, vol. 9, pp. 10338–10351, 2021.
[8] A. Kar and P. K. Singh, "Weather Forecasting Using Artificial Neural Network and Data Mining Techniques," International Conference on Computer, Communication and Control (IC4), pp. 1–4, 2015.
[9] M. R. Meshram, S. R. Biradar, and M. S. Shingate, "IoT Based Weather Monitoring and Forecasting System Using Raspberry Pi," IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), pp. 2055–2059, 2017.
[10] K. K. Patel, S. M. Patel, and S. M. Prajapati, "Weather Monitoring and Forecasting System using IoT," International Journal of Engineering Research & Technology (IJERT), vol. 6, no. 6, pp. 32–35, 2017.
[11] N. B. Priya and S. M. Kumar, "Real-Time Weather Forecasting System using Machine Learning and Raspberry Pi," International Journal of Recent Technology and Engineering (IJRTE), vol. 8, no. 4, pp. 2755–2759, 2019.
[12] H. K. Tripathi and P. C. Pandey, "Weather Forecasting Using Big Data and Artificial Neural Networks," 2019 International Conference on Smart Systems and Inventive Technology (ICSSIT), pp. 1185–1190, 2019.
[13] D. A. Patel and D. M. Patel, "Real-Time Weather Forecasting Using OpenWeatherMap API," International Journal of Advanced Research in Computer Science, vol. 9, no. 2, pp. 323–326, 2018.
[14] G. S. Bang and M. H. Choi, "Web-Based Weather Monitoring System Using RESTful API and MongoDB," International Journal of Software Engineering and Its Applications, vol. 10, no. 11, pp. 163–172, 20.