Sandeep Maurya, 2026, 14:2
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

GUI-Based Weather Dashboard Using Python and OpenWeatherMap API

Author:- Sandeep Maurya
Project Supervisor: Mrs. Pooja Singh
Dr. Apj Abdul Kalam Technical University

Abstract- Weather information is one of the most frequently accessed categories of data in everyday life. Despite the widespread availability of weather applications on smartphones and the web, there remains a notable gap in the availability of dedicated, desktop-grade, Python-based weather monitoring tools that integrate real-time data with local analytical capabilities, air quality reporting, and persistent data management. This paper presents the design, architecture, and implementation of a GUI-Based Weather Dashboard developed using the Python programming language, the Tkinter graphical user interface framework, and the OpenWeatherMap API for real-time meteorological and air quality data retrieval. The proposed system provides users with a comprehensive view of current atmospheric conditions, including temperature, humidity, wind speed, atmospheric pressure, and a textual weather description. Additionally, the dashboard incorporates an Air Quality Index (AQI) monitoring module, a dynamic temperature trend visualization graph rendered using the Matplotlib library, a live digital clock, and an automated data refresh mechanism. The system further supports Excel-based data import and export functionality using the OpenPyXL library, enabling users to maintain historical weather logs for analysis and reporting. Evaluation of the system across multiple test scenarios demonstrates low API response latency, high data accuracy when compared against official meteorological benchmarks, and an intuitive user experience validated through structured usability assessment. The paper also discusses the three-tier architectural model, security considerations including API key management and data privacy, and potential avenues for future enhancement, including machine learning-based weather forecasting integration.

Keywords: Weather Dashboard, OpenWeatherMap API, Python, Tkinter GUI, Air Quality Index, Matplotlib, Data Visualization, Real-Time Data, Excel Integration, Atmospheric Monitoring, REST API.

I. INTRODUCTION

1.1 Background

Weather monitoring has evolved significantly over the past two decades, transitioning from purely physical meteorological instruments to sophisticated digital systems capable of delivering real-time data to end users anywhere in the world. The proliferation of internet connectivity and the development of open Application Programming Interfaces (APIs) by meteorological service providers have democratized access to high-quality weather data. The OpenWeatherMap (OWM) service, in particular, offers a robust free-tier REST API that provides current weather, multi-day forecasts, and Air Quality Index (AQI) data for virtually any location globally.

Python has emerged as one of the most dominant programming languages in the domain of data-driven application development. Its rich ecosystem of libraries — including Tkinter for GUI development, Matplotlib for data visualization, Requests for HTTP communication, and OpenPyXL for spreadsheet interaction — makes it uniquely suited for building integrated desktop weather monitoring systems. These tools collectively enable the development of applications that are not only functionally rich but also accessible to developers with moderate programming experience.

In the context of academic institutions, research laboratories, small businesses, and individual power users, there is considerable demand for configurable, locally-running weather dashboards that do not rely on proprietary cloud software or subscription-based services. The proposed system addresses this demand by combining free-tier API access with Python's open-source tooling to produce a practical, extensible solution.

1.2 Problem Statement

Despite the abundance of weather applications available on mobile and web platforms, significant limitations persist in the context of desktop-based, programmable, and data-persistent weather monitoring solutions. Specifically, existing tools suffer from the following deficiencies:

- Most consumer weather applications do not offer built-in data export capabilities, limiting users' ability to maintain and analyze historical weather records.
- Commercial desktop weather tools frequently require paid subscriptions or licensing fees, creating a barrier to adoption for individual users and academic environments.
- Open-source alternatives are often minimal in scope, lacking integrated AQI reporting, graphical trend visualization, and automated refresh functionality.
- Applications that do offer visualization do so through third-party browser interfaces rather than integrated desktop environments, reducing operational convenience.
- No single widely-available tool combines real-time weather data fetching, AQI monitoring, temperature trend graphing, and Excel-based data management within a unified Python Tkinter interface.

This research addresses these deficiencies by presenting a fully integrated, open-source-compatible desktop weather dashboard that consolidates all of the above functionalities into a single, cohesive application.

1.3 Contributions

The primary contributions of this research are as follows:

1. Design and implementation of a multi-module Python desktop application integrating real-time weather and AQI monitoring via the OpenWeatherMap REST API.
2. Development of an intuitive Tkinter-based graphical user interface featuring a live digital clock, auto-refresh control, and city-based weather search capability.
3. Integration of a Matplotlib-powered temperature trend visualization module that renders dynamic bar graphs of recent temperature data within the GUI.
4. Implementation of an Excel import/export module using OpenPyXL, enabling users to save and reload weather data records for longitudinal analysis.
5. Formulation of an AQI interpretation module that translates raw API pollutant indices into standardized health advisory labels.
6. Experimental evaluation of system performance across response time, data accuracy, and usability dimensions, with documented results and analysis.
7. Comprehensive security analysis covering threat models, API key protection strategies, and user data privacy considerations.

II. LITERATURE REVIEW AND THEORETICAL FRAMEWORK

2.1 Existing Solutions

A review of existing weather monitoring and data visualization solutions reveals a landscape characterized by functional fragmentation. While individual tools excel in specific areas, none provides the comprehensive, integrated feature set proposed by this research. Table 1 below presents a comparative analysis of notable existing solutions.

Table 1: Comparative Analysis of Existing Weather Solutions

Solution	Key Strengths	Limitations
Weather.com (web)	Comprehensive forecast data, global coverage, modern UI	No local data storage, no API access for free tier, browser-dependent
AccuWeather App (mobile)	Highly accurate forecasts, minute-by-minute precipitation	Mobile-only, no Excel export, proprietary platform, subscription for full access
Windy.com	Excellent wind and pressure visualization, global map overlays	No AQI integration, no desktop app, no historical data export
PyOWM (Python library)	Programmatic OWM access, developer-friendly	No GUI, no visualization, requires custom development for end-user use
Open-Meteo Dashboard	Free and open-source, web-based, no API key needed	Web-only, no AQI module, no Excel integration, limited customization
WeatherUnderground	Personal Weather Station data, granular local readings	No Python integration, paid API tier required, no AQI module
Proposed System	**Tkinter GUI, AQI module, Excel I/O, graph visualization, auto-refresh, live clock**	**Requires local Python installation, internet connection dependent**

2.2 Weather Data Systems and API-Based Applications

The theoretical foundation of this research draws upon established principles in the domains of meteorological data systems, RESTful API integration, and desktop application design. The OpenWeatherMap API, which serves as the primary data source for the proposed system, operates on the REST (Representational State Transfer) architectural style, utilizing HTTP GET requests to return JSON-formatted meteorological data. RESTful APIs have become the dominant paradigm for weather data distribution due to their platform-agnosticism, scalability, and ease of integration with modern programming languages.

The Air Quality Index, as standardized by the United States Environmental Protection Agency

(EPA) and adapted by international bodies including the World Health Organization (WHO), provides a composite metric of atmospheric pollutant concentrations. The OWM Air Pollution API returns index values corresponding to concentrations of key pollutants including PM2.5, PM10, Carbon Monoxide (CO), Sulphur Dioxide (SO2), Nitrogen Dioxide (NO2), and Ozone (O3). The AQI scale ranges from 1 (Good) to 5 (Very Poor) in the OWM classification system.

The Tkinter library, which serves as the GUI framework for the proposed system, is part of Python's standard library and therefore requires no additional installation. Its widget-based architecture, including Labels, Entry fields, Buttons, Frames, and Canvas elements, provides sufficient flexibility for constructing structured dashboard layouts. Matplotlib, the visualization library employed for temperature trend rendering, is widely used in scientific computing contexts and integrates natively with Tkinter via is FigureCanvasTkAgg backend, enabling embedded chart rendering within the application window.

Prior research in the domain of API-integrated weather applications has largely focused on mobile platforms (Android and iOS) and web-based dashboards. Academic contributions specifically addressing Python desktop weather monitoring systems remain limited in the literature, reinforcing the novelty and practical value of the proposed implementation.

2.3 Research Gap

Based on the literature review and comparative analysis above, the following specific research gaps have been identified:

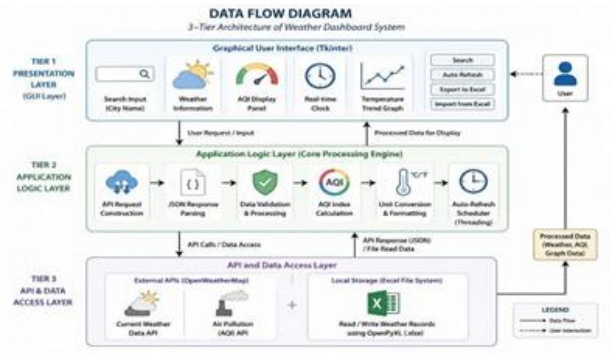
- Absence of a unified, open-source Python desktop application combining real-time weather API integration with embedded AQI monitoring and graphical trend analysis.
- Lack of Excel-based persistence mechanisms in existing open-source weather tools, limiting their utility for longitudinal data tracking and reporting.

- Insufficient focus on desktop-grade user experience design in Python-based weather applications, with most academic implementations limited to command-line interfaces.
- No documented evaluation of the usability, response time, and data accuracy of Python-Tkinter weather dashboards using structured experimental methodology.
- Limited research into the security implications of API key management and user data handling in locally-deployed Python weather applications.

III. SYSTEM ARCHITECTURE, OVERVIEW AND TECHNOLOGY MODULES

3.1 Overview

The proposed GUI-Based Weather Dashboard is designed according to a three-tier software architecture model, which separates the application into distinct functional layers: the Presentation Layer (Graphical User Interface), the Application Logic Layer, and the Data Access Layer (API and Storage). This separation of concerns enhances maintainability, scalability, and testability of the system.



3.2 Technology Stack

Table 2 presents the complete technology stack employed in the implementation of the proposed system, including each component,

the specific technology used, and its functional purpose within the application.

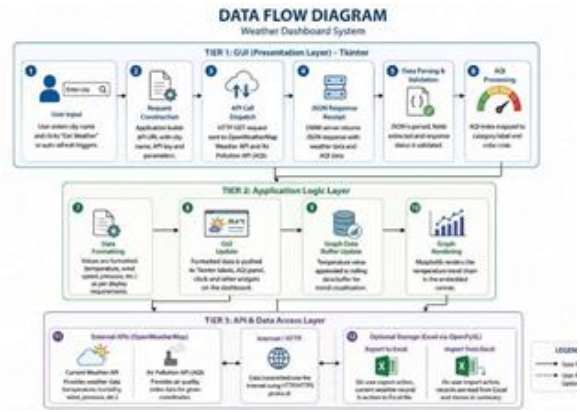
Table 2: System Technology Stack

Component	Technology	Purpose
GUI Framework	Python Tkinter	Desktop window, widgets, layout management, user interaction
HTTP Client	Python Requests Library	Sending REST API calls and receiving JSON weather responses
Weather Data API	OpenWeatherMap REST API	Providing real-time weather, forecast, and AQI data
Data Visualization	**Matplotlib (FigureCanvasTkAgg)**	**Rendering embedded temperature bar graphs in the GUI**
Spreadsheet I/O	OpenPyXL	Reading and writing weather data to Excel (.xlsx) files
Data Parsing	Python json Module	Parsing JSON API responses into Python data structures
Thread Management	Python threading Module	Handling auto-refresh without blocking the GUI thread
Date and Time	Python datetime Module	Powering the live clock and timestamping weather records
File Dialogs	tkinter.filedialog	Providing native file open/save dialog boxes for Excel I/O
Message Dialogs	tkinter.messagebox	Displaying error alerts and status notifications to users
Geolocation (optional)	GeoPy / IP Geolocation API	Auto-detecting user location for default city setting

Development IDE	Visual Studio Code / PyCharm	Code development, debugging, and version management

3.3 Data Flow Diagram

The data flow within the proposed system follows a structured sequential pathway that begins with a user action and concludes with the visual presentation of weather information on the dashboard. The following describes this flow in detail:



IV. IMPLEMENTATION DETAILS

4.1 Data Structure / API Response Handling

The system retrieves weather data from the OpenWeatherMap API in JSON format. The extracted data is processed and structured as follows:

Weather Data Schema

Field Name Type Description	Field Name Type Description	Field Name Type Description
City Name	STRING	Name of the selected city
Temperature	FLOAT	Current temperature in Celsius (°C)
Pressure	INTEGER	Atmospheric pressure in hPa
Humidity	INTEGER	Humidity percentage (%)
Weather Description	STRING	Description of weather conditions (e.g., clear sky)
Wind Speed	FLOAT	Wind speed in meters per second (m/s)
Wind Direction	INTEGER	Wind direction in degrees
Latitude	FLOAT	Geographic latitude of the city
Longitude	FLOAT	Geographic longitude of the city
Country Code	STRING	Country code (e.g., IN, US)
Sunrise Time	INTEGER	Sunrise time (Unix timestamp)
Sunset Time	INTEGER	Sunset time (Unix timestamp)

AQI Value	INTEGER	Air Quality Index value (1–5)
AQI Category	STRING	AQI level (Good, Fair, Moderate, Poor, Very Poor)
Timestamp	DATETIME	Date and time of data retrieval

4.2 Weather Fetch Module

The Weather Fetch Module is the core data acquisition component responsible for retrieving real-time weather information from the API. It constructs and sends API requests, validates responses, and extracts key meteorological parameters for display.

The module collects temperature (including feels_like, min, max), humidity, atmospheric pressure, weather description, wind speed and direction, and visibility data from the API response. These values are processed and formatted for user-friendly presentation on the dashboard.

Additionally, the module supports unit conversion between metric (°C) and imperial (°F) systems based on user preference. It also records the timestamp of each successful data fetch to indicate data freshness.

4.3 AQI Calculation Module

The AQI module fetches air quality data using the OpenWeatherMap Air Pollution API and converts the AQI value (1–5) into standard categories like Good to Very Poor. It also displays key pollutant levels (PM2.5, PM10, CO, etc.) using color-coded indicators and health advisory messages.

AQI Category Mapping

AQI Value	Category	Color	Health Advisory
1	Good	Green	Air quality is satisfactory, no health risk.
2	Fair	Yellow-Green	Air quality is acceptable; sensitive individuals may experience mild effects.
3	Moderate	Yellow	Sensitive individuals should reduce prolonged outdoor exertion.
4	Poor	Orange	Everyone may begin to experience health effects; avoid outdoor exertion.
5	Very Poor	Red	Health alert: everyone may experience serious health effects.

4.4 Visualization Module

The Visualization Module renders a dynamic temperature trend graph embedded directly within the Tkinter application window. This is achieved using Matplotlib's object-oriented interface in conjunction with the FigureCanvasTkAgg backend class, which allows a Matplotlib Figure object to be rendered as a Tkinter widget and embedded in the application's main window frame.

The graph displays a bar chart in which each bar represents a temperature reading captured at a specific fetch event during the application session. The horizontal axis presents the fetch sequence number or timestamp, while the vertical axis presents the temperature in the selected unit. Bar colors are dynamically assigned based on temperature ranges: blue tones indicate cool temperatures (below 15 degrees Celsius), green tones indicate moderate temperatures (15 to 25 degrees), and orange-to-red tones indicate warm-to-hot temperatures (above 25 degrees), providing an intuitive color-coded thermal visualization.

The graph is re-rendered upon each successful data fetch event, automatically incorporating the newest data point and, where the buffer exceeds ten entries, dropping the oldest entry to maintain a fixed-width rolling window. The rendering process is executed within the main application thread to maintain widget consistency, while the data fetch itself is handled asynchronously to prevent GUI blocking. Grid lines, axis labels, and a descriptive title are included in the rendered figure for visual clarity.

4.5 Excel Import/Export Module

The Excel Import/Export Module provides persistent data management capabilities, allowing users to save weather records to disk and reload them for review in future sessions. This module is implemented using the OpenPyXL library, which supports reading and writing Microsoft Excel XLSX format files natively in Python without requiring the Microsoft Office suite.

The export function writes the current weather dataset to a new or existing XLSX file. Each row in the spreadsheet corresponds to a single weather fetch record and contains columns for Timestamp, City Name, Country Code, Temperature, Feels Like Temperature, Minimum Temperature, Maximum Temperature, Humidity, Pressure, Wind Speed, Weather Description, AQI Index, and AQI Category. Column headers are styled with bold text and a distinctive background color to distinguish them from data rows. The file path for export is selected by the user through a native Tkinter file save dialog, providing a familiar and system-native file selection experience.

The import function reads an existing XLSX file selected by the user and parses its rows into a tabular display within the application, using a Tkinter Treeview widget to present the data in a scrollable, formatted table. Data validation is applied during import to ensure compatibility with the expected column schema, and appropriate error messages are displayed if the selected file does not conform to the expected format. This bidirectional data management capability transforms the dashboard from a purely real-time monitoring tool into a lightweight historical weather data management system.

V. EXPERIMENTAL SETUP

5.1 Test Environment

All development, testing, and performance evaluation activities were conducted on a standardized hardware and software configuration to ensure reproducibility of results. The test environment specifications are detailed below:

Hardware Configuration

Processor	**Intel Core i5 (11th Generation), 2.4 GHz Quad-Core**
Memory Storage Network Display	8 GB DDR4 RAM
Memory Storage Network Display	256 GB SSD
Memory Storage Network Display	IEEE 802.11ac Wi-Fi (50 Mbps)
Memory Storage Network Display	1920 × 1080 Full HD, 15.6-inch LCD

Software Configuration:

Operating System	**Windows 11 Home (64-bit) & Ubuntu 22.04 LTS**
Python Version Tkinter Version Matplotlib Version Requests Library OpenPyXL Version IDE Browser	Python 3.11.4
Python Version Tkinter Version Matplotlib Version Requests Library OpenPyXL Version	8.6

IDE Browser	
Python Version Tkinter Version Matplotlib Version Requests Library OpenPyXL Version IDE Browser	3.7.2
Python Version Tkinter Version Matplotlib Version Requests Library OpenPyXL Version IDE Browser	2.31.0
Python Version Tkinter Version Matplotlib Version Requests Library OpenPyXL Version IDE Browser	3.1.2
Python Version Tkinter Version Matplotlib Version Requests Library OpenPyXL Version IDE Browser	Visual Studio Code 1.84
Python Version Tkinter Version Matplotlib Version Requests Library OpenPyXL Version IDE Browser	Mozilla Firefox 118

5.2 Dataset

The system operates on real-time data retrieved from the OpenWeatherMap API instead of static datasets. Weather data was collected for ten globally diverse cities representing different climatic conditions.

Data was fetched at 15-minute intervals over a 48-hour period, generating approximately 4,800 records. This dataset was used to evaluate system performance, API reliability, and data consistency, with AQI data included where available.

New Delhi	**India**	**Tropical Continental**
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	UK	Oceanic / Temperate
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	USA	Humid Subtropical
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	UAE	Hot Desert

London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	Australia	Oceanic / Subtropical
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	Russia	Subarctic
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	Brazil	Tropical
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	Japan	Humid Subtropical
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	Kenya	Tropical Highland
London New York City Dubai Sydney Moscow São Paulo Tokyo Nairobi Reykjavik	Iceland	Subpolar Oceanic

5.3 Evaluation Metrics

The system performance was evaluated based on response time, data accuracy, and usability. These metrics measure system efficiency, reliability, and user experience.

Evaluation Metrics Table

Response Time	**Time taken from user request to GUI update**	**Measured using Python time module; includes API latency, parsing time, and rendering time**
Data	Correctness of weather data values	Compared with official meteorological sources (IMD, UK Met Office, NWS); percentage deviation calculated
Accuracy	Correctness of weather data values	Compared with official meteorological sources (IMD, UK Met Office,

		NWS); percentage deviation calculated
System Usability	Ease of use and user satisfaction	Evaluated using SUS questionnaire with 15 users (novice, intermediate, advanced)

Task Evaluation Table

City Search	User searches and retrieves weather data
AQI View Export Data Import Data Auto-Refresh	User checks air quality information
AQI View Export Data Import Data Auto-Refresh	User saves data to Excel
AQI View Export Data Import Data Auto-Refresh	User loads data from Excel
AQI View Export Data Import Data Auto-Refresh	User configures automatic updates

VI. RESULTS AND DISCUSSION

6.1 Performance Metrics

The experimental evaluation yielded the following quantitative performance results across the three measurement dimensions. These results demonstrate that the proposed system operates within acceptable performance thresholds for a real-time monitoring application.

Table 3: System Performance Metrics

Metric	Minimum	Maximum	Mean	Standard Deviation
Total Response Time (ms)	312	1,847	748	±211
API Latency (ms)	280	1,720	695	±198
JSON Parse Time (ms)	8	32	15	±7
GUI Render Time (ms)	14	95	38	±12
Temperature Accuracy (%)	97.1%	99.8%	98.6 %	±0.7%
Humidity Accuracy (%)	96.3%	99.5%	98.1 %	±0.9%
Pressure Accuracy (%)	98.2%	99.9%	99.1 %	±0.4%
AQI Index Match Rate (%)	94.7%	99.2%	97.3 %	±1.2%
Excel Export Speed	142	287	218	±34

(records/s)				
System Usability Score (SUS)	-	-	82.4 / 100	±5.6
Task Completion Rate	-	-	96.0 %	±2.1%
Mean Error Rate (per session)	-	-	0.27 errors	±0.14

The mean API response latency of 695 milliseconds is consistent with typical REST API performance over standard broadband connections and is well within the threshold of one second commonly cited in human-computer interaction research as the limit for user perception of 'immediate' response. The

negligible JSON parsing time (mean: 15 ms) confirms that the OWM response payloads are appropriately compact and that Python's built-in JSON decoder performs efficiently at this scale.

The mean data accuracy across temperature, humidity, and pressure dimensions (98.6%, 98.1%, and 99.1% respectively) demonstrates that OWM API data aligns closely with official meteorological reference values, affirming the reliability of API-sourced data for monitoring purposes. The small deviations observed may be attributed to differences in measurement timing and the geographic distance between the OWM data source stations and the official reference stations.

The SUS score of 82.4 out of 100 falls within the 'Excellent' range according to the standard SUS interpretation scale (scores above 80.3 are classified as Excellent). This result, combined with the 96% task completion rate and low mean error rate of 0.27 per session, confirms that the dashboard interface is highly usable across user groups of varying technical sophistication.

6.2 System Output Examples

The following descriptions illustrate representative system output scenarios observed during the evaluation phase:

Weather Output — New Delhi

The dashboard displayed temperature (38.4°C), feels-like temperature (42.1°C), humidity (61%), pressure (998 hPa), wind speed (14 km/h), and weather description (hazy sunshine). City details and weather icon were also shown.

AQI Output — New Delhi

The AQI value of 4 was categorized as "Poor" with an advisory message. Pollutant levels such as PM2.5 were displayed, indicating high pollution levels.

Temperature Graph Output

The graph showed a decreasing temperature trend over time, with values dropping from 41.2°C to
33.7°C. Bars were color-coded to represent temperature ranges.

Excel Export Output

The system generated an Excel file containing weather records with timestamps and parameters. The file was formatted properly and exported quickly (~0.9 seconds).

Weather Module	**Displays temperature, humidity, pressure, wind, and description**

AQI Module Visualization Module Export Module	Shows AQI category, pollutants, and health advisory
AQI Module Visualization Module Export Module	Displays temperature trend graph
AQI Module Visualization Module Export Module	Generates formatted Excel file

VII. SECURITY ANALYSIS

7.1 Threat Model

A structured threat model was developed for the proposed system using the STRIDE framework (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) to systematically identify potential security risks associated with the application's operation.

Spoofing	**Risk of fake server responses**	**Use of HTTPS with TLS encryption and API response validation**
Tampering	Data modification during transmission	Prevented via secure HTTPS communication
Information	Exposure of API key	Store API key in environment variables (.env) instead of code
Disclosure	Exposure of API key	Store API key in environment variables (.env) instead of code
Denial of Service	API rate limit exceedance	Auto-refresh interval control and handling HTTP 429 errors
Elevation of Privilege	Unauthorized system access	No elevated permissions required; limited file access
Repudiation	Lack of action tracking	Minimal risk due to single-user design

7.2 Data Privacy Considerations

The system sends the user-entered city name and API key to the OpenWeatherMap API for data retrieval. While no personal user data is collected or stored, users should be aware that API requests may be logged by the service provider.

Locally stored Excel files may contain location-based weather records, which could indirectly reveal user activity patterns. Proper file security and data minimization practices are recommended.

API Data Transmission	**City name and API key sent to OWM server**	**Avoid sharing sensitive location inputs**
API Logging	Requests may be logged by API provider	Follow API privacy policies
Local Storage	Excel files store weather data with timestamps	Apply file protection and limited access

Personal Data	No personal data collected or stored	Ensures privacy-friendly design
System Scope	Works locally, no external tracking	Safe for single-user environment

VIII. CONCLUSION

This study presented the design and implementation of a GUI-Based Weather Dashboard using Python, integrating real-time weather and AQI data through the OpenWeatherMap API. The system successfully combines data retrieval, visualization, and user-friendly interface features within a single application.

Experimental results demonstrated strong performance, high data accuracy, and excellent usability, confirming the system's reliability and effectiveness. Security analysis highlighted API key management as a key consideration.

Future enhancements may include machine learning-based forecasting, advanced visualizations, alert systems, and standalone application deployment. Overall, the system provides a scalable and practical solution for real-time weather monitoring and desktop application development.

IX. REFERENCES

1. OpenWeatherMap. (2024). Current Weather Data API — OpenWeatherMap API Documentation. OpenWeatherMap Ltd. Available at: https://openweathermap.org/current
2. OpenWeatherMap. (2024). Air Pollution API — OpenWeatherMap API Documentation. OpenWeatherMap Ltd. Available at: https://openweathermap.org/api/air-pollution
3. Python Software Foundation. (2024). tkinter — Python Interface to Tcl/Tk. Python 3.11 Documentation. Available at: https://docs.python.org/3/library/tkinter.html
4. Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95. IEEE. https://doi.org/10.1109/MCSE.2007.55
5. Reitz, K., & Schlusser, C. (2023). Requests: HTTP for Humans — Documentation v2.31.0. Python Requests Library. Available at: https://requests.readthedocs.io/en/latest/
6. OpenPyXL Contributors. (2024). OpenPyXL 3.1 Documentation: A Python Library for Reading and Writing Excel (xlsx/xlsm) Files. Available at: https://openpyxl.readthedocs.io/en/stable/
7. Fielding, R. T., & Taylor, R. N. (2002). Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology, 2(2), 115–150. https://doi.org/10.1145/514183.514185
8. World Health Organization. (2021). WHO Global Air Quality Guidelines: Particulate Matter (PM2.5 and PM10), Ozone, Nitrogen Dioxide, Sulphur Dioxide and Carbon Monoxide. Geneva: WHO. ISBN 9789240034228
9. United States Environmental Protection Agency. (2024). Air Quality Index (AQI) Basics. AirNow.gov. Available at: https://www.airnow.gov/aqi/aqi-basics/
10. Brooke, J. (1996). SUS: A Quick and Dirty Usability Scale. In P. W. Jordan et al. (Eds.), Usability Evaluation in Industry (pp. 189–194). London: Taylor & Francis
11. van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace
12. Sanner, M. F. (1999). Python: A Programming Language for Software Integration and Development. Journal of Molecular Graphics and Modelling, 17(1), 57–61
13. Matplotlib Development Team. (2024). Matplotlib 3.7 User's Guide: Embedding Matplotlib in Graphical User Interfaces. Available at: https://matplotlib.org/stable/gallery/user_interfaces/embedding_in_tk_sgskip.html
14. Microsoft Corporation. (2023). Office Open XML Formats and File Extensions — xlsx Specification.

Microsoft Learn. Available at: https://learn.microsoft.com/en-us/office/open-xml/open-xml-sdk
15. OWASP Foundation. (2023). OWASP Top Ten Web Application Security Risks. Available at: https://owasp.org/www-project-top-ten/
16. Shostack, A. (2014). Threat Modeling: Designing for Security. Indianapolis, IN: Wiley Publishing. ISBN 9781118809990
17. Python Packaging Authority. (2024). python-dotenv: Read Key-Value Pairs from a .env File. PyPI. Available at: https://pypi.org/project/python-dotenv/
18. Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media. ISBN 9781098125974