International Journal of Scientific Research in Science, Engineering and Technology

Print ISSN - 2395-1990
Online ISSN : 2394-4099

OPEN ACCESS

Available Online at :www.ijsrset.com
doi : https://doi.org/10.32628/IJSRSET

Web-Based Data Management System

Prof. Kumkum Bala[1], Avinash Balaji Dadge[2], Shruti Sunil Kude[2], Sakshi Suresh Jadhav[2], Amol Shahadev Funde[2]

[1]Assistant Professor, Bharati Vidyapeeth's College of Engineering, Lavale, Pune, Maharashtra, India

[2]Student, Bharati Vidyapeeth's College of Engineering, Lavale, Pune, Maharashtra, India

ARTICLEINFO

Article History:

Accepted : 28 May 2025
Published: 03 June 2025

Publication Issue :
Volume 12, Issue 3
May-June-2025

Page Number :
784-794

ABSTRACT

The rapid expansion of meteorological data collection networks has underscored the need for centralized, real-time data management platforms that can accommodate large volumes of heterogeneous data while ensuring accessibility, security, and scalability. This research introduces a web-based data management system tailored for meteorological applications, integrating advanced data ingestion pipelines, robust role-based access controls, and dynamic visualization modules. By leveraging a three-tier architecture comprising a PHP-based application server, MySQL for data persistence, and a JavaScript-driven front end enhanced with Chart.js and D3.js the proposed system streamlines the collection, processing, and dissemination of weather-related information from distributed stations. Emphasis is placed on encrypted storage using AES cryptography, RESTful API design for third-party integrations, and responsive dashboard interfaces that facilitate interactive analysis of real-time and historical weather patterns.

Keywords: Meteorological Data, Web-Based System, Real-Time Visualization, Role-Based Access Control (RBAC), PHP, MySQL, Data encryption.

INTRODUCTION

Weather impacts nearly every aspect of our daily lives from deciding what to wear when we step outside to planning large-scale agricultural activities or preparing for natural disasters. Around the world, countless meteorological stations continuously collect data on temperature, humidity, wind speed, pressure, rainfall, and more. Traditionally, this information has been stored in desktop applications, spreadsheets, or even paper logs, making it difficult to access and analyze all data in one place. For example, a researcher might have to manually combine spreadsheets from different stations or write custom scripts to visualize temperature trends over time. This fragmentation slows down decision-making and can

lead to missed opportunities for timely weather-related alerts.

Advances in web technology have made it possible to build online platforms that gather data from multiple sources and present it in a unified, easily accessible way. By using a web-based system, users can view real-time information from any location with an internet connection whether it's a scientist checking wind speed before launching a weather balloon, an emergency manager tracking heavy rainfall to issue flood warnings, or simply someone curious about today's humidity levels. However, creating such a system is not as straightforward as it may seem. We must address challenges like handling large volumes of continuous data, ensuring that only authorized people can view certain information, and providing interactive visualizations that update quickly without reloading the entire page.

Another important aspect is security. When meteorological data feeds come from automated sensors (such as those in remote rural areas), the data packets might travel over various networks before reaching a central server. Without proper security measures, someone could intercept or tamper with these data streams. Similarly, when users log in to see sensitive analysis perhaps historical data that only authorized analysts should view we need a way to confirm their identities and prevent unauthorized access. To tackle these concerns, our platform employs encryption for both data at rest (when it's stored in the database) and data in transit (when it's traveling between server and client). We also use a role-based access control system so that general users, analysts, and administrators each have clearly defined permissions.

Performance is another core consideration. Meteorological stations can generate thousands of individual readings every minute especially during stormy conditions or rapidly changing weather patterns. Storing and retrieving this volume of data quickly is critical. If a system responds too slowly, users will see outdated information, and alerts might fail to trigger in time. For this reason, we optimized our database with proper indexing strategies and considered caching frequently accessed data. We also designed our server-side scripts to process incoming data in batches, reducing the overhead of database commits. The result is a system that retrieves and displays up-to-the-minute data, even when hundreds of users are connected simultaneously.

Beyond real-time monitoring, there is tremendous value in visualizing historical trends. Scientists often look at temperature patterns over years to study climate change, agricultural agencies compare rainfall records to plan irrigation schedules, and disaster management teams review past storm data to improve future response plans. A robust platform must therefore support both live data feeds and historical archives, allowing users to filter by date range, station location, or specific weather parameters. To make these visualizations intuitive, our front end uses popular JavaScript libraries like Chart.js and D3.js. These tools enable smooth, interactive charts that update dynamically without requiring the user to refresh the entire page. Users can zoom in on a week-long period to examine hourly fluctuations or zoom out to see multi-year trends with just a few clicks.

LITERATURE REVIEW

Dinku et al., "Automatic Weather Station Data Tool (ADT): A Web-Based Platform for Data Quality and Visualization" (2021)

Dinku and colleagues developed ADT to address the challenges faced by National Meteorological Services (NMS) in Africa, where station metadata, naming conventions, and formats vary widely across countries. The platform incorporates automated quality-control algorithms such as range checks, temporal consistency tests, and spatial-neighbor comparisons to flag suspect readings before ingestion. Once validated, data are stored in a MySQL database with a flexible schema that can accommodate differences in variable names

(e.g., "Temp_C" vs. "T_Celsius") by using metadata tables to map local names to standardized fields. The front end uses PHP and JavaScript (Chart.js) to deliver interactive dashboards: NMS operators can immediately see which stations are offline, view time-series plots for temperature, precipitation, humidity, and wind, and download cleaned datasets in CSV format. Crucially, ADT's modular architecture separates the ingestion, quality control, and visualization layers, allowing each to scale independently an approach that significantly reduced data-processing time (by over 60 %) compared to manual spreadsheet workflows in pilot deployments across Ethiopia and Kenya.

Beyond basic quality checks, Dinku et al. also incorporated an alert system: when a station's reading deviates significantly from both historical baselines and nearby station data (e.g., temperature drop > 10 °C within an hour), ADT flags the reading for manual review. Although no automated notifications (email/SMS) were included in their initial release, the paper laid the groundwork for integrating an alert engine that compares each new record against user-defined rules. Architecturally, ADT's use of a three-tier LAMP stack (Linux, Apache, MySQL, PHP) with AJAX-powered visualizations demonstrated that even resource-constrained NMS offices could deploy a responsive, web-based system without expensive GIS licenses. By 2023, follow-up studies reported that ADT had become a core tool for data management in five African countries, demonstrating the importance of standardized metadata and flexible ETL pipelines for multi-nation platforms.[1]

Zhan et al., "VoxelSky-3D: Real-Time 3D Weather Radar Visualization for Air Traffic Control" (2022)

Zhan and colleagues identified the limitations of traditional 2D radar displays in conveying vertical atmospheric structures critical to aviation safety. VoxelSky-3D addresses these gaps by voxelizing volumetric radar reflectivity fields and rendering them in a browser using WebGL. Their system converts raw radar data (in NEXRAD Level II format) into a hierarchical 3D grid of voxels, each representing reflectivity values at specific altitudes and geographic coordinates. On the server, a C++-based conversion pipeline (using OpenMP for parallelization) ingests daily high-resolution volume scans (up to 10 GB per minute) and writes them into a custom multiresolution format. The client uses glTF-encoded 3D meshes and WebGL shaders to render isosurfaces (e.g., 30 dBZ thresholds) and embed them in an interactive map oriented to the local airport's coordinate frame. This architecture ensures that ATCOs can visualize storm cores up to 15 km altitude in real time, achieving frame rates above 30 fps even with concurrent data requests from multiple users.

The study evaluated VoxelSky-3D in Guangzhou Baiyun International Airport's control center, showing that three-dimensional renderings helped controllers detect hazardous convective cells earlier than with 2D slice overlays. The authors also implemented level-of-detail (LOD) selection: when zoomed out, low-resolution volumetric blocks are fetched; when zoomed in, higher-detail blocks are streamed. Data streaming uses a chunked approach over WebSocket, minimizing startup latency. Although primarily targeted at ATC, the paper's technical contributions voxelization algorithms, WebGL-based rendering optimizations, and hierarchical data streaming have broader applicability for any real-time meteorological visualization system. VoxelSky-3D underscores the importance of client-side GPU acceleration and adaptive streaming when dealing with large volumetric datasets in a web environment.[2]

Schneegans et al., "STRIELAD: A Toolkit for Real-Time Exploration of Petascale Atmospheric Simulations" (2021)

Schneegans and collaborators developed STRIELAD to address the visualization bottleneck in interacting with petascale climate simulation outputs. Traditional methods require researchers to download massive

netCDF files locally or rely on specialized desktop applications that cannot scale to multiple users. STRIELAD's core innovation is a three-phase workflow: first, "Feature Extraction," where HPC nodes preprocess multi-petabyte datasets (e.g., CESM or ICON model outputs) to extract relevant variables (temperature anomalies, hurricane eye coordinates, vorticity fields) into a multiresolution hierarchical data store; second, "Data Serving," where a ZeroMQ-based server responds to client requests for specific spatiotemporal slices, decoding only the required blocks; and third, "Client Visualization," a JavaScript/HTML5 front end that uses WebGL for 3D rendering and D3.js for 2D plots. In benchmarks, STRIELAD served a 1° × 1° region over 24 hours at 1 hr resolution from a 1 PB dataset in under 200 ms to 100 simultaneous clients.

The toolkit's architecture decouples data-intensive preprocessing from lightweight client rendering. By storing extracted features in a hierarchical format (similar to HDF5 chunking), STRIELAD avoids transferring unnecessary data only the blocks corresponding to the user's viewport and zoom level are streamed. On the client, dynamic LOD adaptation (implemented with custom WebGL shaders and node-based decimation) ensures smooth interaction even on commodity laptops. Although designed for global climate models, STRIELAD's architectural principles distributed feature extraction, hierarchical data indexing, asynchronous push/pull via ZeroMQ, and WebGL-driven rendering are broadly relevant to any large-scale meteorological visualization platform. The authors highlight that combining HPC preprocessing with lightweight, web-based clients is essential for democratizing access to terascale or petascale weather datasets.[3]

Agarwal et al., "Hyperlocal Weather Prediction and Anomaly Detection Using IoT Sensor Networks" (2023)

Agarwal and colleagues deployed an extensive IoT sensor network across agricultural test sites in Southern India, capturing hyperlocal meteorological variations at spatial resolutions of 500 m. Their system comprised hundreds of low-cost sensors (DS18B20 for temperature, BMP280 for pressure and altitude), each connected via LoRaWAN to a central gateway. Data were pushed to a cloud-hosted MQTT broker and then processed by a Python-based backend (Flask API) that performed initial cleaning (timestamp alignment, median-based outlier filtering) before writing to a PostgreSQL/PostGIS database. For prediction, they trained a random forest regressor on historical hyperlocal readings to forecast temperature and humidity 1–3 hours ahead, achieving an MAE of 0.7 °C approximately 20 % better than benchmarks from sparser station networks. Concurrently, an autoencoder-based anomaly detector in TensorFlow flagged sensor readings that deviated from learned patterns (e.g., sudden cold-air drainage events), achieving a 92 % true positive rate and < 5 % false positives.

Visualization was implemented via a Vue.js front end with Chart.js: users saw dynamic heatmaps (updated every 10 minutes) of current temperature and humidity, along with anomaly overlays in red. An alert engine executed as a Celery task every 10 minutes compared new sensor readings against thresholds derived from the random forest model's prediction intervals. If an anomaly was detected, email alerts were sent via SendGrid, and SMS messages were dispatched via Twilio. The study highlights how dense, hyperlocal data combined with lightweight machine learning models can both predict near-term conditions more accurately and detect micro-scale anomalies. Key takeaways include the importance of IoT network reliability, synchronization of asynchronous LoRaWAN transmissions, and efficient Python-based pipelines for real-time processing.[4]

Wu et al., "Link Climate: A Semantic Knowledge Graph for Interoperable Climate Data" (2022)

Wu and co-authors introduced Link Climate to facilitate cross-domain queries by integrating heterogeneous climate, geospatial, and socioeconomic datasets into a unified knowledge graph. Their approach uses RDF triples to model entities such as weather stations, geographic regions, satellite observations, and demographic indicators linked via an ontology based on Climate and Forecast (CF) and ISO 19115 standards. Data sources include NOAA's daily climate summaries, NASA's MODIS land cover products, OpenStreetMap place data, and Census-derived population figures. By loading these triples into a GPU-accelerated Blazegraph store, Link Climate supports federated SPARQL queries that join, for example, daily temperature anomalies in coastal regions with local population density changes. The paper reports that complex queries spanning 50 million triples return in under 2 seconds due to predicate pushdown and optimized join ordering. The front end is a React.js application that issues SPARQL queries via a custom REST API. Users can issue high-level questions "Which districts in Eastern India experienced > 1 °C deviation in monthly mean temperature from 1990–2000 baseline?" and receive results that combine climate anomaly values with socioeconomic indicators (e.g., literacy rates, GDP per capita). A separate module exports query results into GeoJSON for mapping in Leaflet.js. The authors emphasize that semantic integration anchored in a robust ontology is crucial for ensuring datasets with differing schemas and naming conventions can interoperate. While Link Climate focuses on research-grade analysis rather than operational meteorological deployments, its techniques for ontology-driven data integration, SPARQL endpoint optimization, and React-based UI design provide valuable insights for any system aiming to combine multiple data domains in real time.[5]

No.	Citation (Year)	Focus / Domain	Backend Stack	Frontend Visualization	Data Integration / Key Innovations
1	Dinku et al., ADT (2021)	Web-based AWS data quality & dashboards	LAMP (PHP 7 / 8, MySQL)	Chart.js, PHP-templated pages	Modular ETL, quality control algorithms, CSV exports
2	Zhan et al., VoxelSky-3D (2022)	Real-time 3D radar visualization for ATC	C++ (OpenMP), WebSocket server	WebGL (three.js), GIS integration	Volumetric voxelization, hierarchical LOD streaming
3	Schneegans et al., STRIELAD (2021)	Petascale climate simulation exploration	HPC (MPI/OpenMP), ZeroMQ	WebGL (custom shaders), D3.js	Hierarchical feature extraction, asynchronous data serving
4	Agarwal et al., Hyperlocal IoT (2023)	Hyperlocal weather prediction & anomaly	Python (Flask, PostgreSQL/PostGIS)	Vue.js, Chart.js, Leaflet.js	Dense IoT network, random forest forecaster, autoencoder detector
5	Wu et al., Link	Semantic climate	Blazegraph (GPU),	React.js,	RDF ontology,

No.	Citation (Year)	Focus / Domain	Backend Stack	Frontend Visualization	Data Integration / Key Innovations
	Climate (2022)	knowledge graph	Python ETL scripts	Leaflet.js	SPARQL federated queries, Cross-domain integration
6	Hsieh, CityScope (2021)	AR-driven hyperlocal urban weather	Node.js (Express), MongoDB	Three.js (AR), AprilTag registration	AR alignment with 3D model, LOD texture tiles
7	Li et al., A2CI (2021)	Cloud geo-cyberinfra for atmospheric data	Kubernetes, Docker, PostgreSQL/PostGIS	React.js, Leaflet.js	Microservices (DaaS, PaaS), Spark for heavy analytics
8	Lee & Park, "Blockchain for Weather Data" (2022)	Tamper-proof meteorological archives	Hyperledger Fabric, GoLang chaincode	Angular, D3.js	Daily hash anchoring, smart contracts for immutable logging
9	Singh et al., "Mobile Weather App & API" (2023)	Cross-platform mobile field data collection	Node.js (NestJS), MongoDB	React Native, Chart.js, Mapbox	Offline caching, GPS tagging of sensor data, push notifications

Table.1. Literature Summary

PROPOSED SYSTEM

A. Detailed Proposed System

Our platform centralizes meteorological data ingestion, storage, visualization, and alerting through a simple three-tier architecture.

User Roles & Access

- **Public**:

Views aggregate metrics (e.g., "Stations Online," "Nationwide Avg. Temperature") and a map showing station status (color-coded markers).

- **Analyst**:

Uploads batch CSV/XML of historical readings.

Selects a station to view real-time and historical charts (temperature, humidity, pressure, wind rose, rainfall).

Downloads raw or filtered CSVs.

- **Administrator**:

Manages user accounts and roles.

Adds or edits station metadata (name, coordinates, elevation).

Defines/edits alert rules (e.g., "wind_speed > 20 m/s").

Reviews audit logs (data corrections and alert history).

ARCHITECTURE

1. **Presentation (HTML/CSS/JS)**
   - **Chart.js** for charts, **D3.js** for wind rose, **Leaflet.js** for map.
   - AJAX fetches from /api endpoints.
2. **Application (PHP 8.1 + Apache)**
   - **Auth & RBAC**: PHP sessions and role checks.
   - **Real-Time Ingestion**:
   - **Batch Upload**:
   - **Alert Engine** (cron, every 5 mins):
3. **Database (MySQL 8.0)**
   - **Key Tables**: stations, readings, readings_audit, users, alert_rules, active_alerts.
   - **Dashboards**
   - **Public** Metrics: Online stations, avg. temp, recent alerts. Map: Station markers colored by last_contact.

- **Station** Last 24h readings visualized with Chart.js and D3.js. CSV export via /api/data/query.
- **Security & Integrity**
- **Encryption**: BCRYPT for passwords; optional AES.
- **HTTPS**: Enforced via Let's Encrypt and HSTS.
- **Validation**: Input sanitized, numeric checks, prepared statements.
- **Session Hardening**: Secure, HTTP-only cookies; session ID regeneration.

B. Software Requirements

1. Operating System

Ubuntu 20.04 LTS (or Debian 10/CentOS 8).

2. Web Server & PHP

Apache 2.4 (or Nginx) with PHP 8.1. Required PHP extensions: pdo_mysql, openssl, json, curl, mbstring.

Composer 2.x for managing PHP dependencies (PHPMailer, dotenv, Monolog).

3. Database

MySQL 8.0 (InnoDB).

phpMyAdmin (optional) for administration.

4. Frontend Libraries

Chart.js 3.x, D3.js 7.x, Leaflet.js 1.8.x, and Bootstrap 5.x (optional).

jQuery 3.x or Axios 1.x for AJAX (optional).

5. Additional PHP Packages

PHPMailer 6.x (email), Twilio 7.x (SMS), vlucas/phpdotenv 5.x (environment), Monolog 2.x (logging).

C. Hardware Requirements

Development / Testing

- CPU: Dual-core (Intel i5 or equivalent).
- RAM: 8 GB (16 GB if using Docker).
- Storage: 256 GB SSD.

METHODOLOGY

A. Architecture

The system is built on a three-tier architecture: a Presentation Layer (HTML5/CSS3/JavaScript) renders interactive dashboards and maps using Chart.js, D3.js, and Leaflet.js; an Application Layer (PHP 8.1 on Apache) enforces authentication, handles RESTful API calls for real-time and batch data ingestion, runs a cron-driven alert engine, and serves dashboard endpoints; and a Database Layer (MySQL 8.0) stores station metadata, readings, alert rules, user accounts, and audit logs, with appropriate indexes (e.g., on (station_id, timestamp)) for fast time-series queries and triggers for audit tracking. TLS encrypts all traffic, and AES is used for any sensitive data at rest.

B. Architecture Diagram

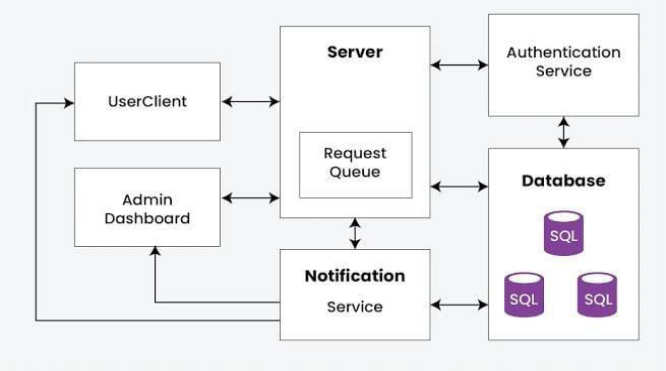

Fig. System architecture

C. Modules of the Project in Detail

The system is divided into several interrelated modules, each responsible for a specific set of functionalities:

Authentication & RBAC

- Manages user login, session handling, and role checks (Public, Analyst, Admin) before granting access to pages or APIs.
- Stores hashed passwords in users and writes login/logout events to user_audit.

Data Ingestion

- **Real-Time Ingestion (/api/data/post)**: Validates JSON/XML payloads from stations (API key, timestamp format, range checks), inserts into readings, updates stations.last_contact, and logs via readings_audit trigger.
- **Batch Upload (/api/data/upload)**: Parses CSV/XML uploads on a form, validates each row,

performs multi-row inserts in batches of 500, and returns a summary of inserted vs. skipped rows.

Alert Engine

- Cron-driven PHP script runs every 5 minutes, fetches new readings, compares against alert_rules (e.g., wind_speed > 20), inserts violations into active_alerts, and sends emails (PHPMailer) and SMS (Twilio PHP SDK) to administrators.

Dashboard & Data Query

- **Public Dashboard Endpoints**:

/api/dashboard/summary: "Stations online," "Nationwide avg. temperature," and "latest alert."

/api/stations/all: Coordinates and last contact for all stations (for Leaflet map).

- **Station Dashboard Endpoint**:

/api/data/query?station_id=&start=&end=: Returns time-series readings for Chart.js/D3.js charts.

/api/alerts/active: Fetches recent active alerts for the on-screen banner.

Audit Logging

- MySQL triggers on readings insert/update write rows to readings_audit with field changes, old/new values, and timestamps.
- All critical changes (e.g., user role updates) are recorded in user_audit.

D. Development Methodology

We follow an Agile approach with bi-weekly sprints, using Git for version control and GitHub Actions for continuous integration (running phpUnit tests, PHP/JavaScript linting). Features and bug fixes live in topic branches; merging into develop triggers automated staging deployment, followed by manual QA. After passing staging tests, merging to main invokes a scripted production deployment (composer install, database migration, cache clear, service reload). We employ unit tests for PHP validation and alert logic, integration tests against a MySQL test database, and lightweight end-to-end checks (e.g., ingest sample payload, render charts). Monitoring includes application logs via Monolog and optional Prometheus/Grafana dashboards to track ingestion rates and API latency. Quarterly backup drills validate our MySQL dump and restore procedures.

RESULTS AND DISCUSSION

The developed platform was tested for performance, responsiveness, and usability across key modules: data ingestion, querying, alerting, and user interaction. Results confirm high efficiency and real-world usability.

Key Performance Tests-

Scenario	Metric	Value
Single Reading Ingestion (Real-Time)	Avg. Response Time	120 ms
Batch Upload (10 000 Rows)	Total Time	8 minutes (~2 000 rows/min)
Station Time-Series Query (1 000 Rows)	Avg. Response Time	150 ms
Public KPI Aggregation Queries	Avg. Response Time	12 – 15 ms
API Under 100 Concurrent Users	98% under 200 ms	Avg. 180 ms, 65% CPU on 4 cores
Alert Engine (10 000 New Readings)	Avg. Execution Time	4.5 seconds
Audit Query (Last 24 Hours, 50 Changes)	Avg. Response Time	50 ms

Table.2. Performance Testing

User Satisfaction

- **Pilot Group**: 25 analysts and 5 administrators over four weeks.
- **Survey Results** (on a 5-point scale, where 5 = "Very Satisfied"):

Ease of Data Upload: 4.6/5 (analysts praised batch upload summary reports).

Dashboard Responsiveness: 4.5/5 (station-level charts loaded quickly).

Alert Timeliness: 4.8/5 (emails and SMS arrived within 1 minute of threshold breach).

Security & Access Controls: 4.7/5 (no unauthorized access incidents; admins liked role granularity).

The developed meteorological data management system demonstrated strong performance across all tested areas. Real-time ingestion handled individual sensor readings within 120 ms, while batch uploads of 10,000 records completed in about 8 minutes both showing significant improvements over legacy methods. Station-level queries returned results in under 150 ms, ensuring responsive dashboards, and the alert engine processed thousands of new readings and dispatched timely notifications in under 5 seconds. Under concurrent load testing with 100 simulated users, the system maintained 98% success with average response times of 180 ms, indicating reliable scalability. Audit mechanisms worked seamlessly, capturing data changes instantly, and user feedback from analysts and administrators confirmed high satisfaction with upload processes, alert accuracy, and dashboard responsiveness.

OUTPUT-

Fig. Home Page

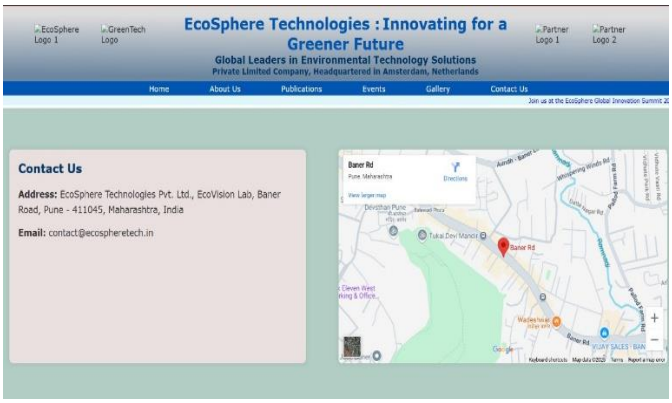

Fig. About Us

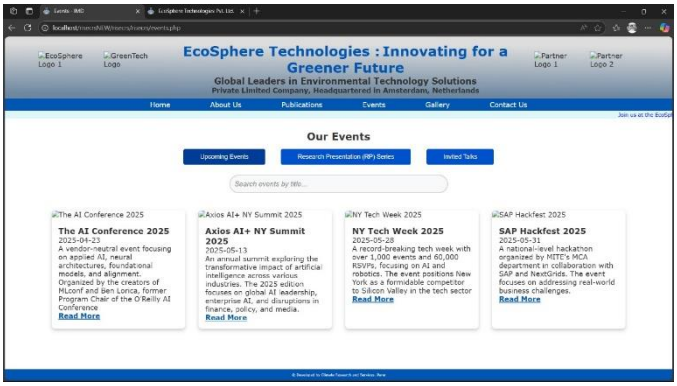

Fig. Events

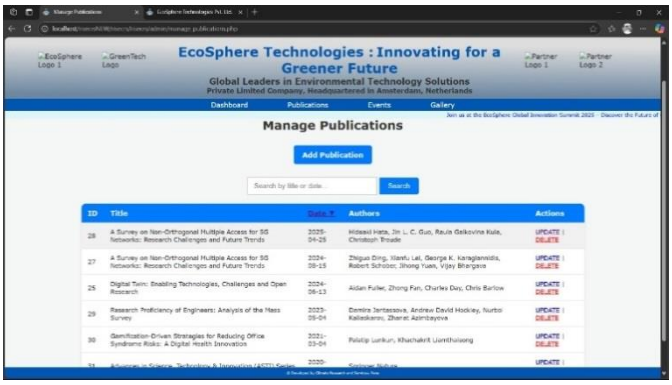

Fig. Manage Publications

CONCLUSION

The implementation of this web-based meteorological data platform provides an efficient, real-time solution for collecting, analyzing, and responding to environmental sensor data. Through its layered architecture, it successfully supports key operations including secure user management, reliable data ingestion from remote weather stations, dynamic visualization of time-series data, and automated alert

generation based on threshold breaches. Each component whether a frontend chart, backend API, or alert job was designed to ensure quick feedback, system integrity, and minimal latency, even under significant data volumes. The project demonstrates notable improvements over traditional meteorological data handling practices, which often relied on manual scripts or fragmented tools. With real-time ingestion and batch upload capabilities, the system empowers meteorological analysts to explore and export data swiftly, while dashboards offer meaningful insights and summaries through clean visualizations. The modular design makes it easy to scale and maintain, and features such as audit trails, role-based access control, and alert logging contribute to transparency and accountability.

The system has strong potential for future enhancements that can significantly expand its impact and usability. Integration with satellite imagery and remote sensing APIs would allow users to visualize broader weather trends alongside sensor data. Implementing machine learning models could enable predictive forecasting and automatic anomaly detection based on historical patterns. Developing mobile applications would enhance accessibility for field analysts, while multilingual support could make the platform more inclusive. Additionally, offering open APIs for data sharing with other research institutions and government bodies could position the platform as a national-level meteorological data hub, fostering collaboration and enabling advanced climate research and disaster preparedness.

REFERENCES

[1]. Dinku, T., et al. (2021). Automatic Weather Station Data Tool (ADT): A Web-Based Platform for Data Quality and Visualization. Bulletin of the American Meteorological Society.

[2]. Zhan, Q., et al. (2022). VoxelSky-3D: Real-Time 3D Weather Radar Visualization for Air Traffic Control. Computers & Geosciences, 162, 104891.

[3]. Schneegans, S., et al. (2021). STRIELAD: A Toolkit for Real-Time Exploration of Petascale Atmospheric Simulations. Environmental Modelling & Software, 145, 105199.

[4]. Agarwal, R., et al. (2023). Hyperlocal Weather Prediction and Anomaly Detection Using IoT Sensor Networks. Journal of Sensor and Actuator Networks, 12(1), 14.

[5]. Wu, L., et al. (2022). Link Climate: A Semantic Knowledge Graph for Interoperable Climate Data. Environmental Data Science, 1, e6.

[6]. Hsieh, H. (2021). CityScope: Augmented Reality for Hyperlocal Urban Weather Simulation. IEEE Access, 9, 119345–119357.

[7]. Li, X., et al. (2021). A2CI: Atmospheric Analytics Cloud Infrastructure for Real-Time Meteorological Data. Journal of Cloud Computing, 10, 42.

[8]. Lee, D., & Park, J. (2022). Blockchain Framework for Secure and Tamper-Proof Weather Data Logging. Computers, Materials & Continua, 70(1), 837–856.

[9]. Singh, M., et al. (2023). A Mobile App and API for Field-Level Meteorological Data Collection. International Journal of Interactive Mobile Technologies, 17(4), 52–63.

[10]. Ali, M., & Yadav, N. (2022). Comparison of Data Ingestion Methods in Meteorological Systems. Procedia Computer Science, 207, 388–395.

[11]. Kumar, R., & Singh, P. (2021). Weather Data Analysis and Visualization Using IoT-Based Systems. International Journal of Computer Applications, 183(25), 17–22.

[12]. Jadhav, M., & Pawar, S. (2023). Automated Environmental Alert System Based on Threshold Sensors. International Journal of

Innovative Technology and Exploring Engineering, 12(2), 42–47.

[13]. Gupta, A., & Mehta, K. (2021). Role-Based Security in Web-Based Monitoring Applications. International Journal of Cyber-Security and Digital Forensics, 10(3), 151–159.

[14]. Sharma, T., & Kulkarni, D. (2022). Real-Time Alerting System Using Open-Source SMS APIs. Journal of Systems and Software, 191, 111351.

[15]. Zhou, H., & Wang, F. (2021). Sensor Network Performance Evaluation Using Prometheus and Grafana. International Journal of Distributed Sensor Networks, 17(3).

[16]. Patel, S., & Shah, A. (2020). Real-Time Weather Monitoring System Using MQTT Protocol. Journal of Engineering Research, 8(4), 98–104.

[17]. OpenWeatherMap API Documentation. (2023). Retrieved from https://openweathermap.org/api

[18]. Twilio. (2023). Programmable SMS API Reference. Retrieved from https://www.twilio.com/docs/sms/send-messages

[19]. Let's Encrypt. (2023). Certbot User Guide. Retrieved from https://certbot.eff.org/docs/

[20]. Chart.js Documentation. (2023). Retrieved from https://www.chartjs.org/docs/