

MediScope: Implementation of a Web-Based Disease Surveillance Platform using NLP and Machine Learning

Shreeshailya Zirpe¹, Prof. A. H. Kagne², Aaditya Marathe³, Prashant Pawar⁴, Krishna Kawale⁵

Department of Computer Engineering, Sinhgad Academy of Engineering, Pune

Abstract - MediScope is an implemented disease surveillance platform that combines web data acquisition, natural language processing, machine learning, and interactive analytics in a single web-based system. The platform ingests outbreak-related content from the World Health Organization (WHO) RSS feed, the Centers for Disease Control and Prevention (CDC) RSS feed, the disease.sh API, and the WHO Global Health Observatory (GHO) API. A backend pipeline built in Python processes raw articles, classifies outbreak relevance using a TF-IDF and Logistic Regression model with keyword fallback, extracts diseases and locations through a spaCy-driven multi-stage pipeline, enriches incomplete records with WHO GHO disease statistics, and persists the resulting outbreaks in MongoDB Atlas. The processed information is exposed through a Flask REST API and consumed by a React 19 frontend developed with Vite and Tailwind CSS. The user interface provides a summary dashboard, geospatial map view, disease detail pages, comparative analysis, symptom-based search, email alert subscription, and an analytics page. The present work is an implementation paper rather than an experimental benchmark study. Accordingly, it documents the actual architecture, modules, API surface, data flow, and current limitations of the deployed codebase. Quantitative evaluation metrics such as accuracy, precision, recall, F1-score, and throughput benchmarks have not yet been formally recorded in the implementation and are therefore not reported in this paper.

Key Words: Disease Surveillance, Public Health Informatics, Natural Language Processing, Named Entity Recognition, TF-IDF, Logistic Regression, Flask, React, MongoDB Atlas, WHO, CDC.

1. INTRODUCTION

Digital public health surveillance increasingly depends on timely collection of information from heterogeneous online sources. Traditional surveillance systems remain authoritative, but they are often complemented by web-

based reporting streams that can surface emerging events earlier and in broader geographic scope. In this context, disease surveillance tools must solve three practical problems: continuous acquisition of external reports, reliable extraction of outbreak entities from noisy text, and intelligible presentation of the resulting signals to analysts.

MediScope was developed as a complete implementation of this workflow. The system collects health-related news and structured disease data, processes them through an NLP and classification pipeline, stores the results in a cloud-hosted document database, and serves them through a dashboard-oriented web application. The emphasis of the project is on engineering integration: each module is connected to a working end-to-end pipeline rather than being presented as an isolated prototype.

This paper reports the implemented architecture of MediScope. It describes the actual software stack, ingestion modules, NLP pipeline, REST endpoints, presentation features, implementation status, and known limitations extracted from the present codebase. No fabricated experimental results are introduced; where quantitative evaluation is not available, this is stated explicitly.

From an implementation perspective, MediScope is designed as a continuous monitoring workflow rather than a static reporting portal. The backend can repeatedly ingest source data, resolve disease-country pairs, enrich missing values, and update the same outbreak collection used by the analytical interface. This operational orientation makes the project suitable for extension into a higher-frequency monitoring system once evaluation, automation, and deployment hardening are completed.

2. RELATED WORK

Prior work on digital disease detection established that online news, official reports, and web data can serve as useful complements to traditional surveillance channels [1], [2]. At the same time, later studies cautioned that large-scale web data can be misleading without careful modeling

and validation [3]. These findings motivate a system design that privileges source traceability and interpretable data flow.

Named entity recognition and text classification are central to health event extraction. Modern NLP pipelines frequently combine statistical tokenization and entity models with rule-based matching in order to improve robustness on domain-specific terminology [4], [5]. For text relevance scoring, TF-IDF remains a strong baseline representation for short and medium-length documents, while Logistic Regression provides efficient and transparent supervised classification [6], [7].

MediScope adopts this pragmatic engineering direction. Instead of depending on a single end-to-end deep model, the implementation uses RSS and API ingestion, rule-enhanced spaCy extraction, a TF-IDF plus Logistic Regression relevance classifier, and official health data sources from WHO and CDC [8]-[10]. The implementation also reflects a common pattern in applied health informatics: robust pipelines are often built from carefully combined classical techniques, official data services, and deterministic rules rather than from a single research model.

3. SYSTEM ARCHITECTURE AND DESIGN

3.1 Overall Architecture

MediScope follows a layered architecture composed of data acquisition, NLP processing, persistence, service delivery, and presentation modules. WHO and CDC RSS feeds populate a raw article store, the disease.sh API contributes structured COVID-19 counts, and the WHO GHO API supplies country-level indicator data used for enrichment. A processor module converts article text into structured outbreak records, which are then exposed to the frontend through JSON-based REST endpoints.

This arrangement separates unstructured textual reporting from structured epidemiological indicators. As a result, MediScope can use article-based extraction to identify emerging events while still retaining a path for indicator-driven enrichment when explicit case counts are missing from narrative reports.

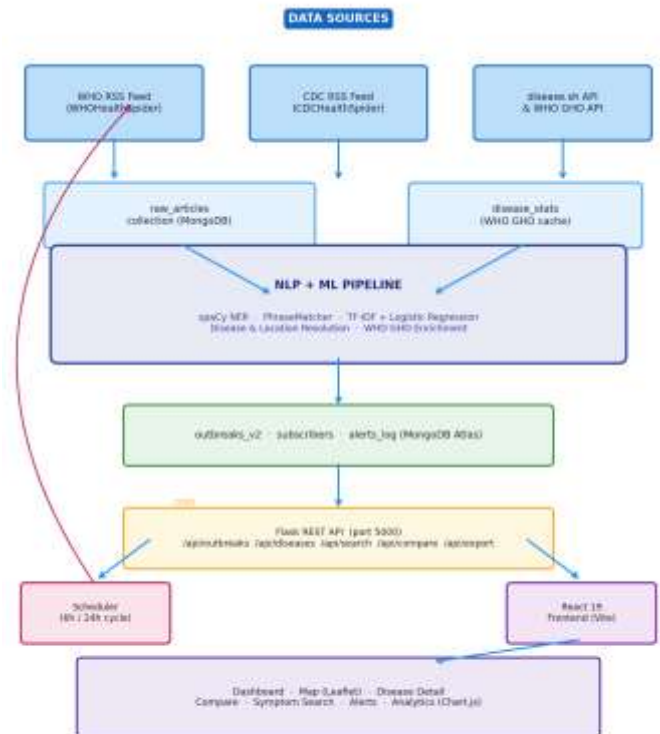


Fig. 1. System Data Architecture Flow Map



Fig. 1. System Data Architecture Flow Map

Layer	Technology	Purpose
Frontend	React 19, Vite, Tailwind CSS, Axios	SPA delivery and API access
Visualization	Leaflet, Chart.js	Map and analytics rendering
Backend	Flask, Python	REST API and orchestration
Database	MongoDB Atlas	Document persistence
NLP	spaCy, PhraseMatcher, substring fallback	Entity extraction
ML	TF-IDF, Logistic Regression	Relevance classification
Data Sources	WHO RSS, CDC RSS, disease.sh, WHO GHO API	Ingestion and enrichment
Scheduling	schedule	Periodic crawler and alert jobs
Alerting	SendGrid (simulated)	Subscription workflow

Table 1: MediScope Technology Stack

3.2 Data Acquisition Layer

The acquisition layer contains four concrete source adapters: WHOHealthSpider, CDCHealthSpider, DiseaseAPISpider, and WHOGHOSpider.

WHOHealthSpider and CDCHealthSpider are Scrappy-based RSS collectors that parse XML feeds and insert article metadata and content into the raw_articles collection. DiseaseAPISpider performs HTTP requests against disease.sh and writes structured COVID-19 outbreak records directly to outbreaks_v2.

WHOGHOSpider queries the WHO GHO OData API on a slower cycle and stores disease statistics in disease_stats for later enrichment.

The scheduler is configured around a six-hour cycle for crawler and alert activity, while WHO GHO collection is run every twenty-four hours because it targets indicator data rather than fast-breaking news. This split cadence is consistent with the different temporal behavior of the supported sources.

3.3 NLP Processing Pipeline

The NLP pipeline processes each raw article by building an analysis string from its title and content. Disease extraction uses spaCy with the en_core_web_sm model, PhraseMatcher patterns, alias mapping, and substring fallback to capture spelling variants and hyphenated forms. Location extraction relies on spaCy geographic entities followed by normalization rules.

Article relevance is determined by a TF-IDF plus Logistic Regression classifier. The classifier yields a probability-style confidence score; when this model path is unavailable or insufficient, a keyword-based fallback is used so that the pipeline continues to operate. This hybrid design reflects the actual implementation and favors resilience over architectural minimalism. For each processed article, the system records processing metadata such as relevance, confidence, diseases_found, locations_found, and a processed flag. Relevant articles proceed to outbreak construction, while non-relevant items are retained as processed source material.

3.4 Disease and Location Resolution

After extraction, MediScope resolves the primary disease and country through multiple validation passes. Disease candidates are compared against canonical mappings, known aliases, and a restricted set of valid diseases accepted by the outbreak store. Country candidates are normalized through location maps and guards are applied

to prevent a country from being misinterpreted as a disease name, or vice versa. If an explicit case count is absent in article text, the system attempts enrichment from disease_stats using the detected disease and country.

Case count extraction first applies regular-expression patterns to article text, including comma-formatted numbers and phrases such as “cases” and similar count-bearing expressions. If no suitable value is found, the enrichment layer consults WHO GHO-derived statistics and records whether the resulting value originated from textual extraction, WHO GHO enrichment, or remained unavailable.

3.5 Persistence Layer

MongoDB Atlas serves as the persistence layer. The raw_articles collection stores ingested RSS content and processing flags. outbreaks_v2 stores deduplicated outbreak records used by the API and frontend. disease_stats stores WHO GHO indicator values used for case count enrichment. subscribers records alert subscriptions, while alerts_log stores sent-alert history used to avoid duplicates. The implementation also includes supporting collections such as diseases and processed_articles. This schema reflects the hybrid nature of the application: some collections retain source text and intermediate annotations, while others provide query-oriented records for visualization, comparison, and export.

3.6 Serving Layer

The serving layer is a Flask REST API running on port 5000 with CORS enabled for the React development origin. The implemented endpoints include /api/health, /api/diseases, /api/diseases/{id}, /api/outbreaks, /api/outbreaks/summary, /api/outbreaks/map, /api/outbreaks/trends, /api/analytics/activity, /api/search, /api/compare, /api/export/csv, and POST /api/alerts/subscribe. These routes provide the operational contract between the backend and the frontend application.

The routes expose both entity-centric and task-centric views of the stored data. For example, /api/outbreaks/summary aggregates outbreak statistics, /api/outbreaks/map emits geospatially consumable records, /api/search supports symptom-based lookup, /api/compare supports pairwise disease analysis, and /api/export/csv exposes a reporting path suitable for offline analysis.

3.7 Presentation Layer

The presentation layer is implemented as a React 19 single-page application built with Vite. The Home page presents summary cards and headline outbreak statistics. Map provides a Leaflet-based geographic view with marker clustering. Disease Detail exposes disease-specific information. Compare supports side-by-side comparison of two diseases. Search matches diseases by symptom keywords. Alerts supports email subscription by region, and Analytics summarizes activity trends, top diseases, report sources, and top countries. Axios is used for API access and Chart.js is present for analytics visualization. The interface structure demonstrates that the project is more than an API proof of concept, including user-facing exploration workflows for monitoring, search, comparison, and subscription.

4. IMPLEMENTATION

4.1 Backend Development

Backend development focused on modular Python services. Flask is used for API orchestration, pymongo manages Atlas connectivity, and Scrapy supports XML/RSS parsing for the WHO and CDC feeds. The codebase separates concerns into crawler, pipeline, API, utility, and scheduler modules. A scheduler based on the schedule library is configured for periodic crawler and alert execution, with article processing available through the pipeline layer. The design choice to keep crawlers, enrichment logic, alert handling, and HTTP routes in separate modules improves maintainability and makes future replacement of individual services feasible.

4.2 NLP Module Development

The NLP module was implemented as a hybrid pipeline instead of a single learned extractor. spaCy provides tokenization and geographic entity recognition, PhraseMatcher improves recall on known disease expressions, and fallback substring scanning catches lexical variants that would otherwise be missed. Relevance classification is trained through scikit-learn using TF-IDF features and Logistic Regression, with serialized model artifacts loaded at runtime. A keyword-based fallback remains in place as a reliability mechanism. This layered approach is particularly appropriate for a student implementation because it preserves transparency.

4.3 Frontend Development

Frontend development used React 19, react-router-dom, Tailwind CSS, Axios, Leaflet, and Chart.js. The interface is organized as route-based pages rather than as a single dashboard panel, allowing summary, map, analytics, search, and disease exploration tasks to remain distinct. Leaflet and OpenStreetMap are used to provide a practical, lightweight geospatial view without introducing dependence on a paid map service. The frontend also includes disease comparison and symptom search workflows, which transform stored outbreak and disease metadata into interactive public-health analysis features.



Figure no. 1- Dashboard

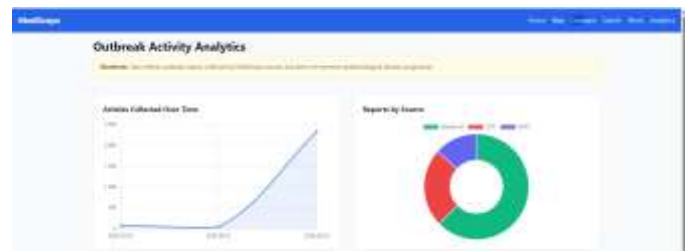


Figure no. 2- Analytics page

4.4 Database Design

Database design uses document collections aligned with the pipeline lifecycle. raw_articles stores source-specific text content together with processing flags and classifier outputs. outbreaks_v2 stores the main disease-country outbreak entities used for visualization and export. disease_stats holds WHO GHO metrics used for enrichment. subscribers and alerts_log support the alert subsystem. The choice of MongoDB Atlas simplifies heterogeneous record storage and incremental schema evolution. Some outbreak records originate from processed article narratives, while others are inserted directly from structured APIs such as disease.sh. Retaining these records within a common document model simplifies downstream querying by the REST layer.

4.5 Alert System

The alert system matches subscriber regions against outbreak records and prepares email notifications through SendGrid integration points. However, the present implementation simulates delivery rather than executing live outbound email in the default project state. Alert events are still logged, making the workflow suitable for functional demonstration and future activation once production credentials and subscription flow corrections are finalized. The current alert design already establishes the core control flow required for future deployment: subscription persistence, region-based matching, duplicate suppression through alerts_log, and a clear mail dispatch boundary.

5. RESULTS AND CURRENT STATUS

MediScope currently operates as a functional implementation with live data ingestion paths, structured outbreak persistence, REST serving, and a multi-page web frontend. The principal result of the project is therefore the successful integration of heterogeneous modules into a single surveillance workflow rather than the publication of benchmark scores.

The platform already demonstrates several end-to-end capabilities. It can crawl official RSS feeds, fetch structured case statistics, classify article relevance, extract disease and geographic entities, deduplicate outbreak records, render them through a REST API, and present them in summary, map, comparison, and analytics views. These capabilities confirm that the core project objective has been achieved at the implementation level.

Module	Current Status
disease.sh integration	Complete
WHO GHO integration	Complete
NLP extraction pipeline	Complete
TF-IDF + Logistic Regression relevance classifier	Complete
Flask REST API	Complete
React frontend	Complete
CSV export	Complete
Alert delivery	Simulated
Scheduler-driven article processing	Partially Implemented
Automated testing	Not Implemented

Table 2: Module Implementation Status

At the present stage, quantitative evaluation has not yet been formally recorded. The codebase does not persist classifier accuracy, precision, recall, F1-score, confusion matrices, pipeline throughput, or end-to-end deployment

benchmarks. This absence is important and should be read as an implementation boundary rather than as a negative result.

Several limitations remain visible in the existing implementation. The frontend alert payload does not match the backend subscription field, alert scanning is not restricted to a recent window, the article processor re-evaluates all raw articles on each run, severity is calculated differently in the pipeline and API utility layers, and no automated test framework is configured. Additional gaps include the absence of authentication, pagination, unsubscribe support, production deployment infrastructure, and formal monitoring. These observations should be interpreted as actionable engineering findings rather than as failures of the project concept.

6. CONCLUSIONS

This paper presented MediScope as a complete implementation of a web-based disease surveillance platform integrating data acquisition, NLP, machine learning, cloud persistence, REST delivery, and interactive visualization. The project successfully demonstrates that official health feeds, public APIs, rule-enhanced NLP, and modern web interfaces can be combined into a coherent operational system for outbreak monitoring.

An important contribution of the implementation is the bridging of multiple data modalities within a single application. Narrative news articles, structured health indicators, user subscriptions, and interactive geospatial views are all connected through one codebase, which is a meaningful systems-engineering outcome for academic project work.

The current implementation is practically useful for exploratory surveillance, educational demonstration, and further research-oriented extension. Future enhancement should prioritize quantitative evaluation, alert workflow correction, consistent severity modeling, processing automation, broader disease coverage, and test-driven hardening for production deployment.

REFERENCES

- [1] J. S. Brownstein, C. C. Freifeld, and L. C. Madoff, "Digital disease detection - harnessing the web for public health surveillance," N. Engl. J. Med., vol. 360, no. 21, pp. 2153-2157, 2009.
- [2] R. Chunara, J. R. Andrews, and J. S. Brownstein, "Social and news media enable estimation of

- epidemiological patterns early in the 2010 Haitian cholera outbreak,” *Am. J. Trop. Med. Hyg.*, vol. 86, no. 1, pp. 39-45, 2012.
- [3] D. Lazer, R. Kennedy, G. King, and A. Vespignani, “The parable of Google Flu: traps in big data analysis,” *Science*, vol. 343, no. 6176, pp. 1203-1205, 2014.
- [4] M. Honnibal and I. Montani, “spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing,” 2017.
- [5] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, “Neural architectures for named entity recognition,” in *Proc. NAACL-HLT*, 2016, pp. 260-270.
- [6] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513-523, 1988.
- [7] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825-2830, 2011.
- [8] World Health Organization, “Disease Outbreak News,” [Online]. Available: <https://www.who.int/emergencies/disease-outbreak-news>. [Accessed: Jun. 24, 2026].
- [9] World Health Organization, “Global Health Observatory API,” [Online]. Available: <https://ghoapi.azureedge.net/api/>. [Accessed: Jun. 24, 2026].
- [10] Centers for Disease Control and Prevention, “CDC Newsroom RSS Feeds,” [Online]. Available: <https://tools.cdc.gov/api/v2/resources/media/316422.rss>. [Accessed: Jun. 24, 2026].