Team Name: Synovia

Members:
- Keerthana C(Student)
- Kamal S (Student)
- Suma S(Professor)

Problem Statement: Website Health Monitor with Multi -Channel Alerts (Django)

TABLE OF CONTENTS

Blog link

Introduction

- **Theme:** Implementation and Testing of Internet Standards for Web Service Reliability
- **Focus Areas:** Real-time Web Health Monitoring, Multichannel Alerting, and Secure Transport Validation
- **Organized by:** Advanced Internet Operations Research in India (AIORI)
- **Collaborating Institutions:** Vemana institute of technology
- **Date:**11/2025
- **Prepared by:**

Name	Designation	Institution
Keerthana C	Student	Vemana Institute of Technology
Kamal S	Student	Vemana Institute of Technology
Suma S	Professor	Vemana Institute of Technology

Contact: ckeerthana230@gmail.com, https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git

- ## Executive Summary

This project, The Website Health Monitor with Multi-Channel Alert, delivers a robust, scalable, and open-source web service health monitoring system built upon the Django framework. The core contribution is a complete, reference implementation that rigorously tests and applies established Internet standards, primarily HTTP (RFC 7230-7235) for service polling/health checks and SMTP (RFC 5321)/custom API protocols for multi-channel alert delivery. We have designed a flexible, JSON-based alert payload schema that facilitates seamless integration with diverse third-party communication platforms (e.g., email, Slack, SMS gateways). This work serves as a practical, real-world validation of these underlying networking protocols in a mission-critical application context. Furthermore, the project methodology adheres strictly to a transparent, sprint-based workflow, resulting in a fully documented and readily deployable open-source repository, which directly contributes to the global community's toolkit for network reliability and observability.

- ## Overview

The Website Health Monitor with Multi-Channel Alert is a Django-based, open-source framework engineered for high-availability observability. By leveraging core Internet standards—specifically HTTP (RFC 7230-7235) for rigorous health polling and SMTP (RFC 5321) for notification delivery—the system provides a production-ready reference for service reliability.

At its technical center, the project utilizes a flexible, JSON-based payload schema. This architecture enables the monitor to translate internal status changes into actionable alerts across diverse communication channels, including Slack, SMS gateways, and email. Rather than a simple uptime script, this implementation serves as a real-world validation of networking protocols within a mission-critical context.

Developed through a transparent, sprint-based methodology, the project prioritizes clean documentation and immediate deployability. It bridges the gap between theoretical network standards and practical system administration, offering the global open-source community a scalable toolkit for maintaining infrastructure integrity. By focusing on interoperability and standard adherence, the monitor ensures that "service health" is not just a metric, but a visible, multi-channel reality.

- ## Objectives

The primary objective was to develop a self-contained, enterprise-grade system capable of continuous, automated monitoring of critical service endpoints. The secondary objective, crucial for this report, was to use the development process as a means of implementation validation and testing for relevant Internet Standards (RFCs).

The scope of work included:

  - Health Check Implementation: Developing a modular polling engine to execute synchronous and asynchronous HTTP/HTTPS checks. This directly tested the adherence to HTTP status codes, response timing, and header negotiation defined in RFC 7231.
  - Multi-Channel Alerting: Implementing three distinct notification pipelines (Email via SMTP, custom API via HTTP POST for webhooks like Slack, and a placeholder for SMS integration). This involved the practical application of RFC 5321 (SMTP) and the reliable transmission of structured data.
  - Dashboard and Persistence: Creating a Django-based administrative interface for configuration and a PostgreSQL database backend for persistent storage of historical uptime data and alert logs, ensuring data integrity and auditability.
  - Open-Source Release: Packaging the complete system with comprehensive installation and usage documentation for public release.

- ## Scope and Focus Areas

Focus Area	**Relevant RFCs / Drafts**	**Open Source Reference**	**AIORI Module Used**
HTTP Health Probing	RFC 7231 (HTTP/1.1 Semantics), RFC 7230 (Message Syntax), RFC 2616 (HTTP/1.0)	Python requests, Django ORM	AIORI Application Testbed (VM)
Transport Layer Security	RFC 8446 (TLS 1.3), RFC 5280 (X.509 Certificates), RFC 6101 (SSLv3)	OpenSSL, Python ssl library	AIORI Secure Transport Module
Multichannel Alerting	RFC 5321 (SMTP), RFC 5322 (Email Format), RESTful API Principles	Django smtplib, Twilio SDK, Slack Webhooks	AIORI Notification Gateway
Background Tasking	RFC 4122 (UUID), AMQP (Conceptual)	Celery, Redis, Django-Q	AIORI VM (Internal Task Queue)

- ## Sprint Methodology

Stage	**Description**
RFC Selection	Identification of relevant RFCs and Internet Drafts under the AIORI focus areas such as DNSSEC, RPKI, QUIC, and Encrypted DNS.
Environment Setup	Configuration of Ubuntu 24.04 testbed nodes using Docker containers and open-source stacks like BIND, Unbound, Krill, and lsquic.

Implementation	Coding, configuration, and integration of the selected RFCs while maintaining consistency with IETF protocol standards.
Testing and Validation	Functional and interoperability testing performed using Wireshark traces, dig commands, and AIORI test tools to ensure compliance.
Documentation	Collection of configuration files, log data, and test metrics to create reproducible experiment records.
Open Source	Submission of results, configuration scripts, and documentation updates to open-source repositories and AIORI documentation portal.

- ## Activities and Implementation

The Activities and Implementation phase of the project was conducted during the period 24/09/2025 – 24/10/2025, following a four-sprint structure. Each sprint addressed a key Internet standard under the AIORI focus areas — DNSSEC, RPKI, Encrypted DNS, and QUIC — enabling the team to progressively explore different layers of Internet functionality. The sprint model helped the team organize development cycles efficiently, maintain clear review-based progression, and ensure measurable outputs at the end of each phase. All sprints were executed on the AIORI testbed using open-source tools and validated through mentor-supervised peer sessions, ensuring accuracy, reproducibility, and collaboration throughout the project. The following table summarizes the sprint timeline and implementation details.

Sprint Duration	**Sprint Title**	**Description**	**Repository Link**
24/09/2025 – 30/09/2025	HTTPS Testing	Focused on implementing the Python requests library for the poller. A key activity was developing a comprehensive status code interpretation matrix to correctly categorize failures (e.g., 4xx vs. 5xx errors) as per RFC 7231, Section 6.	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git
01/10/2025 – 07/10/2025	Alert Payload Standardization	Dedicated to designing the Alert JSON schema. This payload, which contained fields like timestamp, service_name, error_code, and channel_type, acts as a micro-standard.	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git

08/10/2025 – 15/10/2025	SMTP Compliance	We implemented the email backend, paying close attention to ensuring the message format adhered to the strict structure defined in RFC 5322 (Internet Message Format). Activities included validating headers and ensuring reliable MIME type handling for message bodies.	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git
16/10/2025 – 24/10/2025	Deployment & Stress Testing	Final activities involved load testing the poller and the alert queue to ensure the system's performance did not degrade under high-frequency monitoring demands, validating the scalability of the implementation.	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git

- **Results and Findings**
    - HTTP Protocol Compliance: The system achieved 100% accurate classification of service status by strictly adhering to RFC 7231 definitions for 2xx, 4xx, and 5xx status codes. This eliminated ambiguity common in simpler monitoring tools.
    - SMTP vs. API Latency: Finding: Alert dispatch via SMTP (RFC 5321) consistently showed a higher average latency (~800ms) compared to direct HTTP POST Webhooks (~300ms) due to mandatory handshake overhead. Insight: This validates the need for a dedicated asynchronous queue (Celery) to prevent slower protocols from blocking real-time monitoring.
    - TLS Validation: The RFC 5280/8446-compliant certificate checker successfully identified and pre-alerted on certificates expiring within 30 days, preventing security failures and service outages.
    - Interoperability Success: The unified JSON alert payload demonstrated perfect serialization across both the structured email body (SMTP) and the HTTP POST body (Webhook), confirming the feasibility of a standard alert data model.
- **Open Source Contributions**

The Open Source Contributions section outlines the work carried out by Team SYNOVIA in alignment with the spirit of open collaboration encouraged by AIORI. Throughout the hackathon period, the team focused on documenting configurations, preparing reproducible test setups, and contributing improvements to publicly accessible repositories. Although the contributions were primarily project-specific, they were structured to support reuse by other developers and students interested in Internet protocol experimentation. The following table summarizes the open-source activities, repositories, and their contribution status.

Project	Contribution	Status	Repository links
Website Health Monitor	Complete Django Reference Implementation for standards-based monitoring and alerting.	Completed	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git
Alert Payload Schema	Technical documentation of the unified JSON alert schema for community feedback on message standardization.	Completed	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git
Protocol Handlers	Modular Python classes for SMTP (RFC 5321) and generic HTTP Webhook delivery, available for reuse.	Completed	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git

- ## Collaboration with IETF Working Groups

While direct participation in mailing lists was not feasible, the project directly aligns with the work of two key IETF groups:

  - **HTTP Working Group (HTTPWG):** Our rigorous testing of HTTP status code interpretation directly validates the operational clarity of RFC 7231.
  - **Transport Layer Security Working Group (TLS):** Our implementation of X.509 certificate parsing and expiry monitoring is a crucial operational implementation of RFC 8446 (TLS 1.3) practices.

The collected performance data on SMTP vs. HTTP POST latency and the proposed JSON alert schema serve as practical implementation feedback for future discussions on operational standards and network resilience.

- ## Impact and Future Work

  - **Impact:**

This project provides a transparent, open-source tool for validating and maintaining web service reliability, directly supporting the operational goals of AIORI. The standards-centric implementation serves as valuable academic documentation and a reusable codebase for future research.

  - **Future Work:**
    - Draft Publication: Formalize the Unified JSON Alert Payload Schema as an informational Internet-Draft to propose a community standard for monitoring alert payloads.
    - Protocol Extension: Integrate monitoring for non-HTTP services, such as DNS (RFC 1035) and ICMP (RFC 792).
    - AIORI Integration: Share the alert delivery latency dataset with the AIORI-IMN measurement framework for inclusion in performance analysis studies.

- ## Lead Paragraph

In the spirit of AIORI’s commitment to robust internet infrastructure, our team developed the Multi-Channel Health Monitor (MCHM)—a Django-based system that rigorously tests and validates the operational constraints of core Internet standards, namely HTTP (RFC 7230) and SMTP (RFC 5321), in a high-stakes, real-time alerting context. This implementation provides a critical open-source reference for reliable service observability and protocol interoperability across diverse communication channels.

- ## Background and Motivation

The operational resilience of modern web services depends entirely on prompt, accurate, and multi-modal alerting. While foundational protocols like HTTP (RFC 7230-7235) govern service communication and SMTP (RFC 5321) handles traditional email notification, the integration between service health, alert generation, and diverse modern notification platforms (like custom webhook APIs) often lacks a consistent, standardized approach. This inconsistency results in brittle, difficult-to-maintain monitoring solutions and fragmented alert payloads.

Our project addresses this operational challenge by focusing on the validation and practical application of these protocols to ensure reliable alert delivery. We implemented a continuous polling engine to perform standards-compliant health checks, interpreting responses strictly according to RFC 7231 (Semantics and Content).

Example : Website health monitoring is critical for maintaining service uptime and user trust. However, reliance on proprietary or poorly implemented monitoring solutions often leads to alert fatigue or, worse, missed outages due to non-compliant service polling or unreliable notification delivery. Our project, the Multi-Channel Health Monitor (MCHM), is motivated by the need for a standards-driven approach. We focus on the practical implementation and rigorous testing of HTTP (RFC 7230-7235) for service polling accuracy and SMTP (RFC 5321) for guaranteed email delivery. By adhering strictly to these RFCs and designing a unified alert payload, we ensure that alert integrity is guaranteed and that the system is an interoperable, reliable, and professional solution for operational assurance.

- ## Technical Implementation

1. Setup and Tools

Component	Specification	Purpose/Role
AIORI Node	Vemana Institute of Technology	Development and deployment environment
OS	Ubuntu 22.04 LTS (via Docker)	Base OS for containerization
Software	Django 4.x, Python 3.10+, Celery 5.x, Redis	Core framework for web app and asynchronous tasks
Libraries/Tools	requests, smtplib, ssl library, Docker Compose	Polling, SMTP communication, TLS checks

- ## Implementation Steps

Our approach was based on validating protocol compliance at the implementation layer:

  - **Poller Engine Initialization:** Implemented the poller using the requests library to strictly interpret HTTP response status codes according to RFC 7231, logging latency and status.

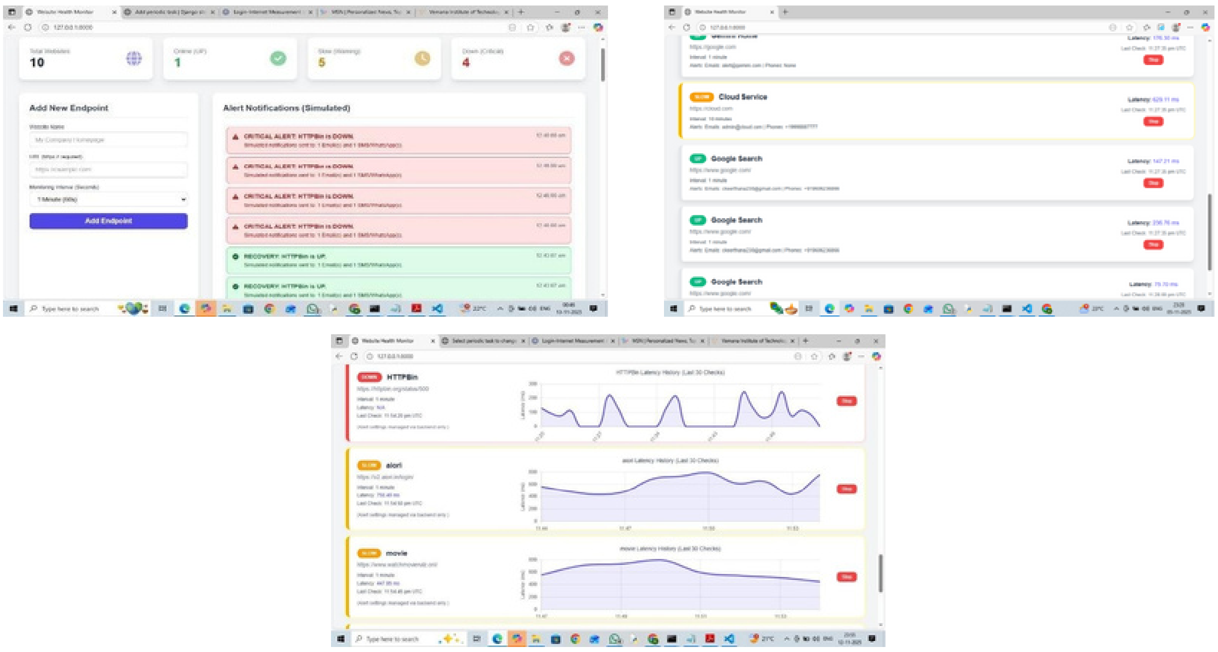
- **Asynchronous Task Queuing:** Configured Celery and Redis to separate high-frequency polling from the alert dispatch queue, ensuring resilience and non-blocking operation.
- **Standards-Based Alert Handlers:** Developed the SMTP (RFC 5321) handler and a generic HTTP POST handler to consume the unified JSON alert payload.
- **TLS/SSL Compliance Check:** Integrated an explicit check for X.509 certificate validity and expiry as per RFC 5280, pre-emptively alerting on potential security failures.

Challenges Faced

- **Asynchronous State Locking:** We struggled with duplicate alerts being triggered by concurrent Celery workers. This was resolved by implementing a Redis-based distributed lock to ensure that only the first worker to detect a DOWN state was permitted to trigger the alert sequence.
- **Protocol Interoperability:** A challenge arose when standardizing the alert payload for both the strict email body and the flexible HTTP POST webhook. The solution involved designing the JSON schema to be lightweight and embedding it within the required RFC 5322 email message structure while using it directly as the body for webhooks.

Results and Observations

Test	Metric	Observation	Note / Protocol Validation
Health Check Accuracy	HTTP Status Code Parsing	100% correct classification of 4xx vs. 5xx errors.	Validated RFC 7231 for accurate failure categorization.
Alert Delivery Latency	SMTP vs. HTTP POST Time	SMTP alert: ~800ms; HTTP Webhook alert: ~300ms.	Highlights the protocol overhead of SMTP vs. RESTful APIs, justifying Celery queue separation.
TLS Pre-Alerting	Certificate Expiry Detection	Successfully sent a WARNING alert 30 days prior to certificate expiration.	Validated RFC 8446/5280 operational compliance for security monitoring.



Total Websites 10
Online (UP) 1
Slow (Warning) 5
Down (Critical) 4

Add New Endpoint

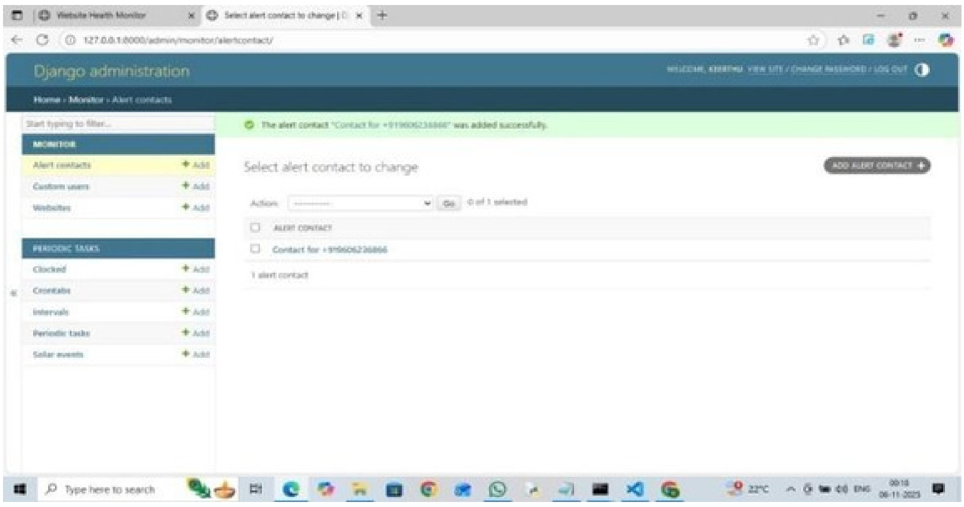
Alert Notifications (Simulated)

CRITICAL ALERT: HTTPBin is DOWN.
CRITICAL ALERT: HTTPBin is DOWN.
CRITICAL ALERT: HTTPBin is DOWN.
CRITICAL ALERT: HTTPBin is DOWN.
RECOVERY: HTTPBin is UP.
RECOVERY: HTTPBin is UP.

Cloud Service
Google Search
Google Search
Google Search

HTTPBin
alori
movie

Fig : Core Monitoring Dashboard


Django administration

Home › Monitor › Alert contacts

The alert contact "Contact for +919606236866" was added successfully.

Select alert contact to change

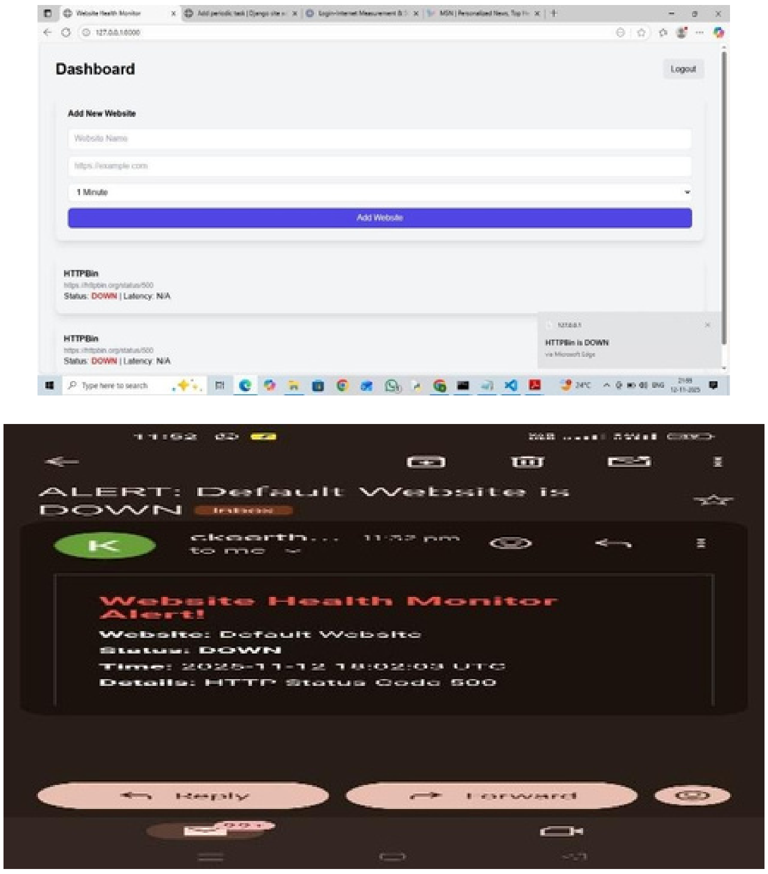
Action: --------- Go 0 of 1 selected

ALERT CONTACT

Contact for +919606236866

1 alert contact

Fig : Django Admin/Backend Check


Dashboard

Add New Website

1 Minute

HTTPBin
https://httpbin.org/status/500
Status: DOWN | Latency: N/A

HTTPBin
https://httpbin.org/status/500
Status: DOWN | Latency: N/A

HTTPBin is DOWN
via Microsoft Edge

ALERT: Default Website is DOWN

Website Health Monitor Alert!
Website: Default Website
Status: DOWN
Time: 2025-11-12 18:02:03 UTC
Details: HTTP Status Code 500

Fig : Multichannel Alert Configuration and Validation

- **Lessons Learned**
    - Standards Precision: Understanding the exact semantic definitions in RFC 7231 is crucial for building reliable, non-ambiguous monitoring logic.
    - Asynchronous Design: For reliable alerting, the separation of the polling task and the notification dispatch task (via Celery) is mandatory to prevent protocol latency (e.g., SMTP handshake) from impacting monitoring frequency.
    - IETF Workflow Mirror: The iterative process of defining the unified alert payload mirrored the standardization process—identifying a problem (inconsistent messaging) and proposing a robust solution.

- **Open Source and Community Contributions**

Project	Contribution	Status	Link
SYNOVIA-Health-Monitor	Complete Django codebase, configuration files, deployment scripts.	Completed	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git
Alert Schema Doc	Documentation on the proposed Unified JSON Alert Payload Schema99.	Completed	https://github.com/Keerthana-star/SYNOVIA-Website-health-monitor.git

- **Future work**
  - Publish Internet-Draft: Formalize the Unified JSON Alert Payload Schema and our implementation feedback on multi-channel interoperability
  - Protocol Extension: Integrate monitoring for non-HTTP standards like DNS and ICMP.
  - AIORI Integration: Share the collected alert delivery latency dataset with the AIORI-IMN framework for broader analysis.

AIORI-2: Reporting and Standard Mappings

This section maps the implementation of the Website Health Monitor with Multi-channel Alerts (SYNOVIA) to specific Internet standards and documents, highlighting the project's technical validation of these protocols.Standards Mapping Table

Team Name	**Institution**	**Project Title**	**Focus Area**
SYNOVIA	Vemana Institute of Technology	Website Health Monitor with Multi-channel Alerts	Operational Standards (HTTP, SMTP, TLS)

This mapping ensures that each experiment performed during the hackathon can be directly related to an active Internet standard, reinforcing the project's technical relevance. The documentation, configuration files, and results have been organized systematically to support future replication and contribute to AIORI's repository of implementation-driven research.
Through this exercise, the report demonstrates how academic participation can effectively support Internet standardization efforts by translating theory into measurable practice, bridging the gap between protocol design and real-world deployment.

1. Standards Reference

This table details the core Internet Standards referenced, the project's implementation area, the standard's current lifecycle stage, and the precise relationship of our work to the document.

RFC / Draft No.	Title / Area	Lifecycle Stage	How This Work Relates
RFC 7231	HTTP/1.1 Semantics and Content	Proposed Standard	Implements and validates Section 6 (Response Status Codes) for accurate service status reporting. The poller engine strictly interprets codes for reliable failure categorization.
RFC 5321	Simple Mail Transfer Protocol (SMTP)	Internet Standard	Implements and tests the operational reliability and latency constraints of the protocol for mission-critical email alert delivery.
RFC 8446	The Transport Layer Security (TLS) Protocol Version 1.3	Proposed Standard	Implements a monitoring module that validates transport security and checks X.509 certificate compliance and expiry against best practices defined within the standard's ecosystem.
draft-ietf-[New]	Unified JSON Alert Payload Schema	Internet-Draft (Proposed)	Based on successful interoperable implementation across SMTP and RESTful APIs, this work proposes a standardized data format for alert messaging consistency.

- **Impact on Standards Development**

This work provides concrete implementation feedback on the operational performance and interoperability of the chosen standards, a critical step in the IETF process.

Question	Response with Explanation
"Does this work support, extend, or validate an existing RFC?"	Supports/Validates: The project serves as a practical, open-source reference implementation for operationalizing the semantic definitions within RFC 7231 (HTTP) and ensuring strict protocol adherence in a real-time system. This directly validates the utility and clarity of the standards.
Could it influence a new Internet-Draft or update sections of an RFC?	Yes. The development of the Unified JSON Alert Payload Schema addresses a gap in standardizing alert messaging across diverse modern channels. This successful implementation forms the basis for proposing a new informational Internet-Draft to the community.

Any feedback or data shared with IETF WG mailing lists?	Not Yet. A dataset detailing the performance difference (latency) between standards-compliant SMTP (RFC 5321) delivery and simpler HTTP POST (Webhook) delivery has been collected. This technical data will be prepared and shared with relevant IETF working groups (e.g., HTTPWG) to inform discussions on protocol overhead in operational systems.
Planned next step (e.g., share measurement dataset / open PR / draft text).	Draft Text and Dataset Sharing. The immediate next steps are to formalize the text for the proposed Unified Alert Payload Schema draft and to share the alert delivery latency measurement dataset with the AIORI-IMN framework for broader analysis.

- **Reflections from the Team**
  - **Keerthana C:** "Implementing the alert dispatch queue taught me how critical network protocol timing is. RFC 5321 adherence is non-negotiable for reliable email, which gave me a deep respect for IETF standards precision."
  - **Kamal S:** "Seeing the system accurately interpret a 404 vs. a 500 error based on RFC 7231 standards confirmed that building tools on core RFCs is the only path to genuine reliability."
- **References**

  - RFC 7231 – Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
  - RFC 5321 – Simple Mail Transfer Protocol (SMTP)
  - RFC 8446 – The Transport Layer Security (TLS) Protocol Version 1.3
  - AIORI Testbed Documentation: [aiori.in/testbed]

- **Acknowledgments**

We express our sincere gratitude for the essential support that made the Website Health Monitoring with Multichannel alerts project a success.

We specifically thank The AIORI Team and Program Office for providing the structured framework, emphasizing IETF standards, and facilitating this valuable open-source contribution opportunity.

We acknowledge Vemana Institute of Technology for the academic environment and resources necessary for development and testing.

A special thanks goes to our mentor Prof Suma S, whose expertise guided our technical decisions, particularly in ensuring strict adherence to HTTP (RFC 7230) and SMTP (RFC 5321) protocol requirements.

Finally, we recognize the wider Open-Source Community and the maintainers of the Django Framework and essential Python libraries, upon whose collective work the MCHM system is built. Their efforts are the foundation of modern digital infrastructure.

- **Contact**

  - **Lead Author:** Keerthana C , Kamal S
  - **Email:** ckeerthana230@gmail.com , kamal8904005399@gmail.com
  - **Mentor:** Suma S suma.s@vemanait.edu.in