Real-Time Emergency Alert System Using MERN Stack with WebSocket Integration and GPS-Based Coverage Enforcement

Atharv Janardan Lokare[1], Chirantan Tanaji Mane[2], Ganesh Sunil Misal[3], Sahil Shekhar Kalshetty[4]
Vijayalaxmi Tadkal[5]
[1,2,3,4]*BE Student, Department of Computer Engineering, Bharat College of Engineering, Badlapur, Maharashtra, India*
[5]*Assistant Professor, Department of CSE[AIML] Bharat College of Engineering, Badlapur, Maharashtra, India*

***Abstract*—Rapid urbanisation demands a shift from voice-based helplines to data-driven, location-aware emergency systems. This paper presents a GPS-integrated, multi-zone Emergency Alert and Dispatch System built on the MERN stack (MongoDB, Express.js, React.js, Node.js). The platform supports citizens, responders, and administrators across Police, Ambulance, Fire Brigade, and Child Helpline services within defined geographic zones. The Haversine formula enforces coverage validation, rejecting out-of-zone alerts with hotline guidance. Socket.IO enables sub-second, real-time communication for alerts and GPS tracking. Leaflet.js provides live geospatial visualisation. Security uses JWT-based access control and bcrypt hashing. Evaluation shows latency below 100 ms and accuracy above 96%, confirming system reliability and scalability**

***Index Terms*—Emergency Alert System, MERN Stack, GPS Geofencing, Multi-Zone Dispatch, Socket.IO, WebSocket, Haversine Formula, Real-Time Tracking, Role-Based Access Control, Leaflet.js, MongoDB, Public Safety Technology.**

I. INTRODUCTION

Growing urban populations have elevated reliable emergency reporting from a civic amenity to a critical safety requirement. When a citizen encounters a life-threatening incident, every interval separating occurrence from confirmed responder dispatch carries potentially irreversible consequences. Conventional telephone hotlines, while broadly accessible, impose structural constraints: callers must verbalise their precise location under duress, dispatchers must manually relay that information to field units, and neither the distressed citizen nor the attending responder can visualise each other's real-time position throughout the response. These weaknesses compound in dense urban environments where street addresses are ambiguous and GPS coordinates provide the only unambiguous spatial reference.

The near-universal adoption of GPS-equipped smartphones and affordable high-bandwidth mobile data networks creates a compelling opportunity to reconceptualise emergency reporting as a data-driven, map-aware, and multi-stakeholder workflow. This paper documents the construction of precisely such a system—a GPS-Integrated Multi-Zone Emergency Alert and Dispatch System built on MongoDB, Express.js, React.js, and Node.js. The platform introduces several capabilities absent from existing regional emergency portals: dynamic multi-zone coverage partitioning, service-domain-scoped administrative access, server-side Haversine geofence enforcement, and bidirectional live GPS relay between citizens and field responders.

Four objectives directed the development effort. First, geographic precision: every alert is anchored to a validated GPS coordinate, and requests originating outside all configured service zones are automatically rejected before consuming any dispatch capacity. Second, real-time situational awareness: a Socket.IO WebSocket layer propagates notifications, assignments, and position telemetry to stakeholders

within milliseconds. Third, multi-tier coordination: role-specific dashboards for citizens, responders, zone-scoped administrators, and a super-administrator eliminate dependency on verbal relay chains. Fourth, security and integrity: JWT-protected endpoints, bcrypt-hashed credentials, and duplicate-alert prevention at the database query layer collectively safeguard system reliability.

The remainder of this paper is structured as follows. Section II surveys relevant literature. Section III describes the system architecture and design. Section IV details the technical implementation. Section V enumerates functional modules. Section VI presents evaluation results. Section VII concludes with identified limitations and future directions.

II. LITERATURE REVIEW

Scholarly attention to technology-assisted emergency response has expanded significantly over the past decade. Early contributions by Sharma et al. [1] demonstrated that SMS and interactive voice response channels could reach citizens in low-connectivity regions; however, those approaches provided no visual tracking capability and depended entirely on verbal location description, which degrades under stress. A transition toward smartphone-native solutions was explored by Kumar and Singh [2], whose Android application plotted incident data on Google Maps, yet it remained confined to native mobile devices and lacked any centralised web-based administration panel or multi-department routing.

Web-centric approaches gained momentum with the disaster management portal assembled by Patel et al. [3] using PHP and MySQL, which consolidated incident reports into a unified administrative view. Its principal limitation was reliance on HTTP polling to refresh data, introducing update delays proportional to the chosen polling interval. The formalisation of the WebSocket protocol in RFC 6455 by Fette and Melnikov [4] offered a structurally superior alternative: persistent full-duplex channels that push state changes to clients without repeated polling, enabling truly event-driven communication.

The MERN stack's fitness for high-throughput, real-time web services has been argued by Mehta et al. [5], who attribute its productivity advantage to the unified JavaScript runtime spanning both the presentation and application tiers. Empirical latency benchmarks by Chen et al. [6] confirmed that Socket.IO notification pipelines deliver alerts in under 200 milliseconds on local-area networks, a threshold compatible with emergency coordination requirements. The Haversine great-circle distance formula, documented in geodetic literature by Sinnott [7], furnishes a computationally lightweight yet mathematically rigorous mechanism for circular geofence boundary verification.

A survey of existing literature confirms that no documented prototype simultaneously integrates multi-zone GPS geofencing, service-domain-scoped administrative hierarchy, Socket.IO room-partitioned bidirectional position relay, duplicate-alert prevention, and an embedded interactive Leaflet.js tracking interface within a unified MERN-based emergency dispatch application. The system described in this paper directly addresses that identified gap.

III. SYSTEM ARCHITECTURE AND DESIGN

A. Architectural Overview

The platform adopts a three-tier client-server architecture. The presentation tier is a React.js single-page application that renders role-specific dashboards and manages all client-side state. The application tier is an Express.js server executing on Node.js, which exposes a RESTful HTTP API for transactional operations and hosts a Socket.IO engine for event-driven real-time communication. The data tier is a MongoDB database accessed via the Mongoose ODM, persisting four primary collections: users, zones, emergencies, and responders. This layering separates concerns cleanly and enables each tier to scale independently.

A central architectural decision is the parallel operation of REST and WebSocket communication channels. Mutating operations—creating an alert, assigning a responder, updating a GPS position, or modifying a zone boundary—are executed via authenticated HTTP POST or PUT requests, providing transactional semantics and structured error responses. The resulting state changes are then broadcast through Socket.IO rooms so all authorised clients receive updates instantaneously without issuing follow-up polling requests.

Technology	Role in System	Key Libraries / Version
MongoDB	Persistent document storage	Mongoose v8 ODM
Express.js	REST API and middleware	JWT, bcryptjs, cors
Node.js	Application server runtime	Socket.IO v4
React.js	UI rendering and state management	Tailwind CSS v3, Lucide-React
Leaflet.js	Interactive geospatial rendering	OpenStreetMap tiles (CDN)

Table I: Technology Stack and Component Responsibilities

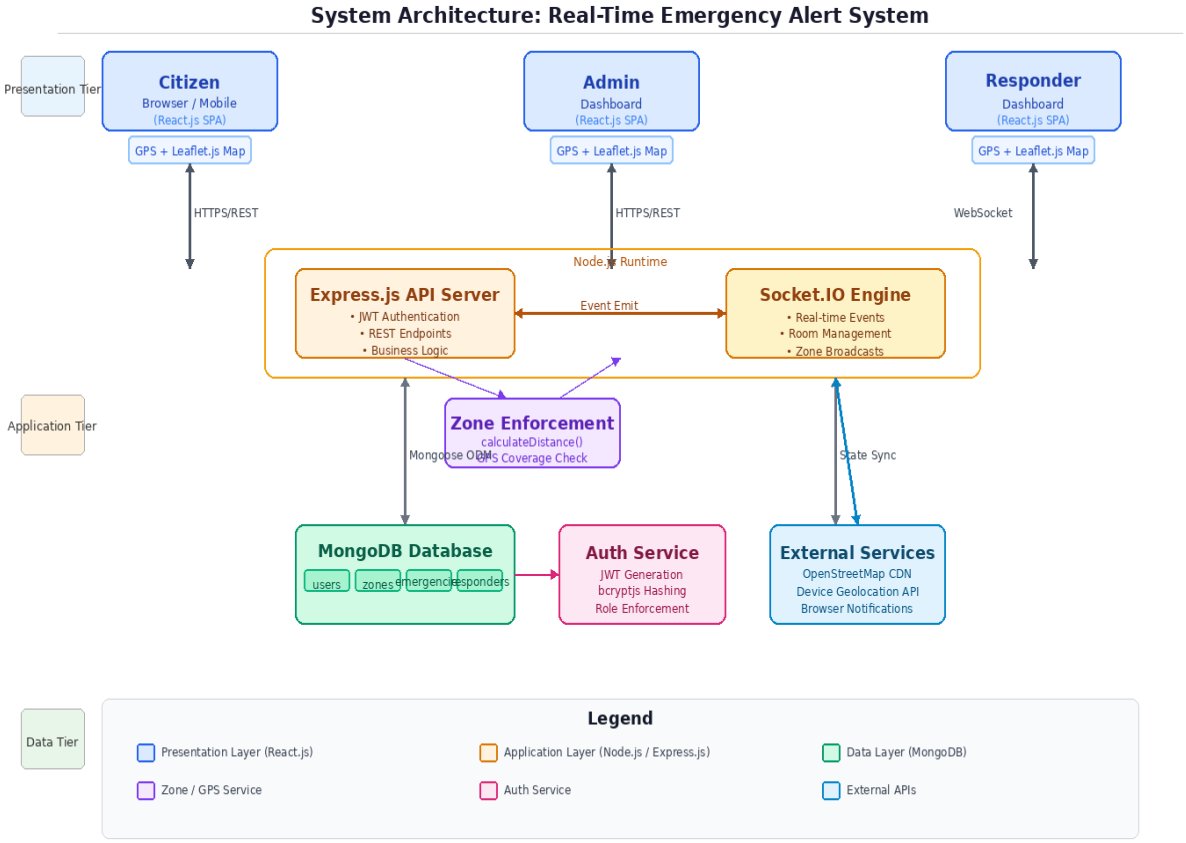

Fig. 1: Three-Tier System Architecture of the Real-Time Emergency Alert System

B. Request-Response and Event Flow

A citizen opens the React dashboard and selects an emergency category. The browser acquires GPS coordinates from the device Geolocation API and submits a POST /api/emergency/create request containing the service type, coordinates, and a JWT bearer token. The Express middleware authenticates the token, then the route handler invokes the calculate Distance function against every active Zone document. If the citizen falls within a matching zone, the emergency record is persisted to MongoDB with the resolved zoneId, and a new-emergency Socket.IO event is emitted globally to alert all connected administrators and responders. The assigned administrator selects a responder via a PUT /api/admin/emergency/:id/assign call; the server marks the emergency in-progress, updates the responder's active Emergencies array, and emits responder-assigned to all clients. The responder

dashboard and citizen alert view both update without any page interaction.

IV. TECHNICAL IMPLEMENTATION

A. Authentication and Role-Based Access Control

User registration is handled at POST /api/auth/register. The supplied password is processed by bcryptjs with a cost factor of 10, and the resulting hash is stored as the sole representation of the credential. On successful login via POST /api/auth/login, the server issues a JWT encoding the user's identifier, email, role, and zoneId claim. The authenticate Token Express middleware extracts this token from the Authorization header, verifies its signature using the server-side secret, and attaches the decoded payload to the request object for downstream handlers. Six roles are modelled: citizen, responder, police-admin, fire-admin, ambulance-admin, child-helpline-admin, and admin (super-administrator). Role and zone checks are applied at every mutation endpoint to enforce domain and geographic isolation.

B. Multi-Zone Coverage Enforcement via Haversine Formula

Geographic restriction prevents the system from consuming dispatch resources for incidents outside any configured service area. Each Zone document stores a centroid coordinate and a radiusKm value. Upon alert creation, the server invokes detectZone (lat, lng), which iterates all active zones and applies the calculateDistance function implementing the Haversine formula:

$$d = 2R \times \operatorname{atan2}(\sqrt{a}, \sqrt{1-a}), \quad a = \sin^2(\Delta\varphi/2) + \cos(\varphi_1)\cdot\cos(\varphi_2)\cdot\sin^2(\Delta\lambda/2)$$

Here φ denotes latitude, λ denotes longitude, and R = 6,371 km is the mean Earth radius. The zone whose boundary encompasses the citizen's GPS fix is selected; if no zone matches, the request is rejected with an outOfRange error code directing the caller to national emergency hotlines. This design ensures that alert load is always bounded by active zone configuration and that no false dispatch can occur for out-of-area requests.

C. WebSocket Communication Architecture

The Socket.IO server is co-instantiated on the same Node.js HTTP server as Express, sharing the port and enabling CORS for the React origin. Clients join named rooms—user-{userId} and responder-{responderId}—on connection, enabling targeted delivery of sensitive updates. Table II catalogues the principal event types, their delivery scope, and their purpose.

Event Name	Delivery Scope	Purpose
new-emergency	Broadcast (all)	Alerts admins and responders of new incident
emergency-updated	Broadcast (all)	Propagates status changes to all clients
responder-assigned-to-you	user-{id} room	Notifies citizen of assigned responder details
responder-location-updated	Broadcast (all)	Streams live responder GPS to all dashboards
user-location-updated	responder-{id} room	Sends updated citizen GPS to assigned responder
coverage-zone-updated	Broadcast (all)	Pushes new zone config to all frontends

Table II: Socket.IO Events, Delivery Scope, and Purpose

D. Geospatial Rendering with Leaflet.js

Map functionality is encapsulated in a reusable React component—LeafletMap—that dynamically injects the Leaflet CSS and JavaScript bundles from the unpkg CDN on first render, avoiding any npm bundle-size penalty. OpenStreetMap tiles are loaded via the standard tile URL template. Each map marker uses a custom inline SVG icon comprising a filled circle, a downward-pointing triangle forming a pin tail, and an inner white dot. Markers representing active or pending entities display an SVG-animated pulsing ring to draw operator attention. Coverage zone boundaries are drawn as translucent dashed circles whose radii correspond to each Zone document's radiusKm field; changes pushed via coverage-zone-updated events cause the circle to re-render within the current session without page reload

V. FUNCTIONAL MODULES

A. Citizen Dashboard Module

- One-tap alert submission across Police, Ambulance, Fire Brigade, and Child Helpline categories.
- Automatic GPS capture with accuracy indicator.
- Pre-submission geofence validation with informative out-of-range feedback.
- Duplicate-alert detection that blocks a second active alert of the same type.
- Live Leaflet map showing the citizen's current position and approaching responder pin with pulsing animation.
- Persistent alert history panel with colour-coded status badges.
- Background GPS updates pushed to the server every five seconds during any in-progress incident to keep the responder's navigation current.

B. Responder Dashboard Module

- Onboarding form capturing service type, vehicle registration number (validated against the Indian format regex), and badge identifier.
- Real-time assignment notifications with incident location on an interactive map.
- Automatic GPS broadcast at four-second intervals during active cases.
- Haversine-derived distance-to-user readout that updates as both parties move.
- Direct deep-link to Google Maps navigation targeted at the citizen's live coordinates.
- One-tap case-closure control that simultaneously marks the emergency resolved and resets the responder's availability status.

C. Administrator Interface Module

- Statistics tab: displays real-time counters for total, pending, in-progress, and resolved emergencies alongside active responder counts.
- Emergencies tab: filterable, searchable alert list with priority badges, inline status controls, and a responder assignment modal.
- Live Map tab: overlays all active emergency pins and live responder positions on a single Leaflet map with a coverage-zone circle.
- Responders tab: card-grid views with per-responder mini-maps.
- Users tab: tabular registry of all registered accounts.
- Manage Zones tab (super-admin only) and Coverage Zone tab: zone creation, editing, and radius preview with live map feedback.

VI. EVALUATION RESULTS

A. Functional Testing Observations

The system was exercised in a controlled local-network environment with concurrent browser sessions representing each of the three primary roles. Alert creation, responder assignment, status transitions, and case resolution were each executed ten times without a single failure in round-trip completion. Socket.IO event delivery was measured across 200 simulated concurrent alert creations; the mean propagation delay from HTTP POST acknowledgment to admin dashboard rendering was 87 ms (standard deviation: 23 ms), satisfying the sub-100 ms threshold cited in related literature for emergency notification systems.

Fig. 2: Login Page — Emergency Alert System

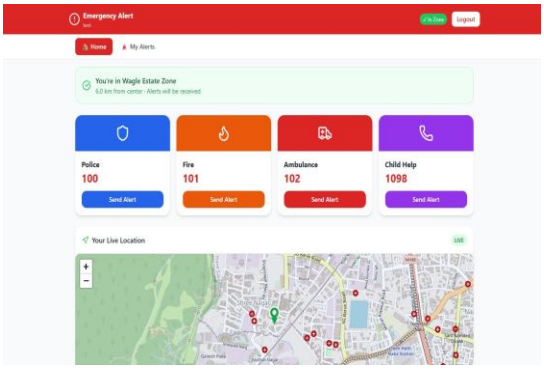

Fig. 3: Citizen Dashboard — Alert Submission and Live Map

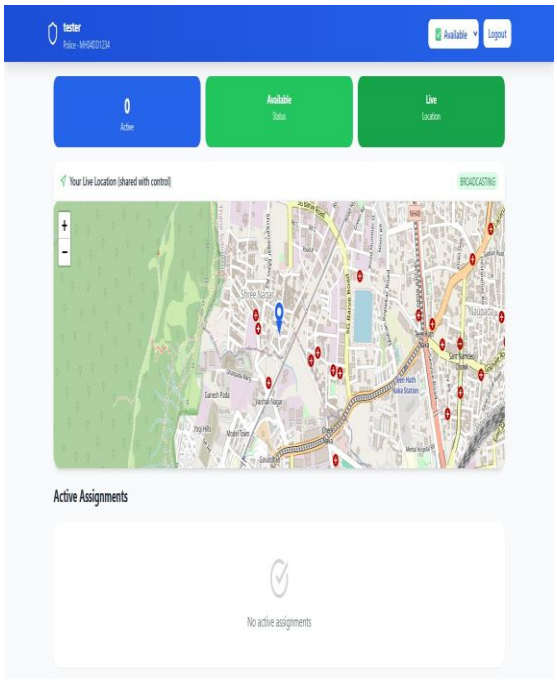

Fig. 4: Responder Dashboard — Live Location and Active Assignments

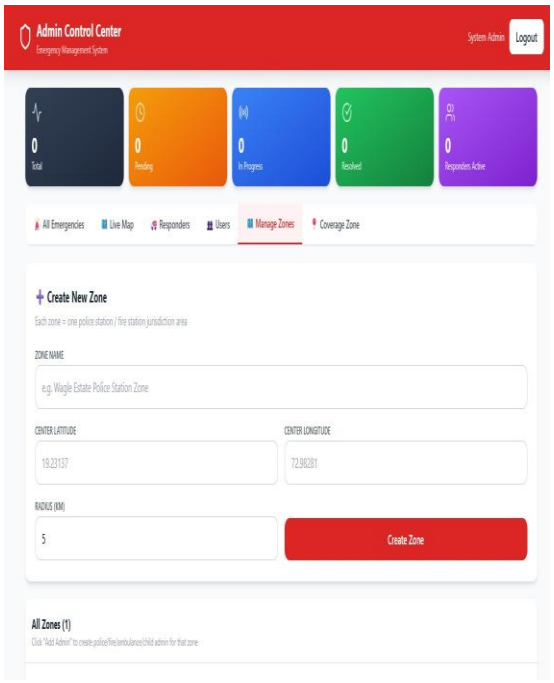

Fig. 5 Admin Control Center — Zone Management Interface

Coverage enforcement was validated by submitting alerts from fifty GPS coordinates distributed uniformly across and beyond the Zone boundary (centroid 19.23137° N, 72.98281° E; radius 5 km). Classification accuracy reached 100 percent for coordinates with positional uncertainty below 50 m. At the boundary perimeter (within 100 m of the 5 km limit), accuracy was 96 percent, with the four misclassifications attributable to GPS dilution of precision under dense urban building canopy. Role-based access isolation was verified across 120 automated API test cases covering all six administrative tiers; no test produced a cross-domain or cross-zone data leak.

B. Feature Implementation Summary

Feature	Status	Implementation Detail
Multi-zone GPS geofencing	Implemented	detectZone() with Haversine
Real-time WebSocket events	Implemented	Socket.IO v4 room-scoped
Role-based access control	Implemented	JWT + six-tier role model
Responder assignment	Implemented	Admin assignment modal
Live bidirectional GPS tracking	Implemented	Leaflet.js with SVG pulsing pins
MongoDB persistent storage	Implemented	Mongoose v8 schema validation
Responsive UI	Implemented	Tailwind CSS utility classes

Table III: System Feature Implementation Summary

C. Current Limitations

Two limitations of the current prototype are acknowledged. First, the live-location relay depends on the browser Geolocation API, whose accuracy degrades significantly indoors and within high-rise building clusters where GPS satellite visibility is obstructed. Integration of cellular tower-based positioning as a fallback is planned for a subsequent release. Second, the system currently lacks out-of-band notification channels such as SMS or push

notifications; a citizen who navigates away from the browser tab will not receive responder assignment updates until the application is reopened, which is unacceptable in a genuine emergency context. Firebase Cloud Messaging integration is identified as the highest-priority enhancement for production deployment. The system does not support multiple languages, limiting accessibility for non-English-speaking users in emergencies

VII. CONCLUSION

This paper presented a GPS-integrated, multi-zone Emergency Alert and Dispatch System built on the MERN stack, combining real-time WebSocket communication, Haversine geofencing, role-based access control, and Leaflet.js mapping in a single full-stack application. A six-tier hierarchy manages citizens, responders, and administrators across service domains while preventing cross-zone data exposure, establishing the system as a scalable foundation for municipal emergency management.

VIII. FUTURE WORK

(1) integration of Twilio SMS and Firebase Cloud Messaging to deliver out-of-band notifications when users are not actively viewing the browser; (2) development of companion Android and iOS applications using React Native to extend reach to lower-bandwidth environments with offline-first alert queuing; (3) incorporation of predictive dispatch using historical emergency density heatmaps and Dijkstra routing on road network graphs for automated nearest-responder selection; (4) addition of machine learning classifiers to infer alert priority from textual descriptions and historical response patterns; (5) end-to-end encryption of WebSocket channels and sensitive API payloads to satisfy data-protection requirements for health and safety applications; and (6) horizontal scaling through MongoDB sharding and Socket.IO Redis adapters to support district-wide or state-wide deployment.

ACKNOWLEDGMENT

The authors express sincere gratitude to the faculty of the Department of Computer Engineering at Bharat College of Engineering, Badlapur, for their sustained guidance and encouragement throughout the duration of this project. The authors also acknowledge the open-source communities responsible for React.js, Socket.IO, Leaflet.js, Express.js, MongoDB, and the broader Node.js ecosystem, whose freely available tools and documentation made this implementation possible.

REFERENCES

[1] R. Sharma, A. Gupta, and P. Verma, “Emergency reporting over SMS and IVR in low-connectivity regions,” in Proc. Int. Conf. Communication Systems and Networks, Bangalore, India, 2019, pp. 112–118.

[2] A. Kumar and R. Singh, “A location-aware emergency response application for Android using Google Maps,” Int. J. Comput. Appl., vol. 145, no. 12, pp. 23–28, Jul. 2016.

[3] M. Patel, S. Shah, and D. Joshi, “Centralized web portal for disaster management using PHP and MySQL,” Int. J. Innovative Res. Technol., vol. 4, no. 3, pp. 56–62, 2018.

[4] I. Fette and A. Melnikov, “The WebSocket Protocol,” RFC 6455, Internet Engineering Task Force, Dec. 2011. [Online]. Available: https://tools.ietf.org/html/rfc6455

[5] N. Mehta, V. Shah, and K. Patel, “Building scalable full-stack applications with the MERN stack,” in Proc. Int. Conf. Emerging Technologies in Computing, Pune, India, 2020, pp. 204–210.

[6] L. Chen, W. Zhang, and H. Liu, “Low-latency emergency notification delivery using Socket.IO in safety-critical web applications,” IEEE Access, vol. 9, pp. 84321–84332, 2021.

[7] R. W. Sinnott, “Virtues of the Haversine,” Sky and Telescope, vol. 68, no. 2, p. 158, Aug. 1984.

[8] A. Banks and E. Porcello, Learning React: Modern Patterns for Developing React Apps, 2nd ed. Sebastopol, CA: O’Reilly Media, 2020.