International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Web-Based National Highway Accident Management System

Deepak C, Shashank L, Rahul R

Department of Artificial Intelligence and Machine Learning, Aditya College of Engineering and Technology, Bangalore, India

Under the guidance of Dr. Souvik Das, Assistant Professor, Department of Artificial Intelligence and Machine Learning

Corresponding email: deepakironman2003@gmail.com
Co-author emails: shashanksln5436@gmail.com, rahulr70099@gmail.com

Abstract—Information about accidents on National Highways is often scattered across telephone calls, paper forms and separate organisational records, which makes it hard to review, locate and compare. This paper presents a web-based National Highway Accident Management System that brings accident reporting, map-based visualisation, highway information, summary analytics and emergency-partner onboarding into one browser application. The prototype uses a three-tier MERN architecture (React front end, Node.js and Express API, MongoDB storage) with token-based authentication. Three roles are supported: citizen, emergency partner and administrator. Hospitals and ambulance services do not receive partner privileges by choosing a category at sign-up; they submit an application that stays pending until an administrator records an approval or rejection, and every partner-only and administrator-only operation is checked on the server. A submitted report is treated as a stored record rather than a verified or live incident. The implementation is described through its architecture, workflows, data model and security design, and screenshots of the running prototype show the account-selection page, the report form, the report register, a record detail view and a live map. No performance, accuracy or usability results have been measured, so none are claimed; a sixteen-case functional and security test plan is given as the evaluation to be carried out.

Keywords—highway accident management, role-based access control, web mapping, MERN stack, emergency partner verification, road safety

I. INTRODUCTION

National Highways carry passenger traffic, freight and access to healthcare and education, and they are also where severe crashes concentrate. The Ministry of Road Transport and Highways (MoRTH) reports about 4.61 lakh road accidents and 4.43 lakh injuries in India for 2022, and press coverage of the same report notes that highways, only about 5% of the road network, account for more than half of accidents and more than 60% of fatalities [1]. When a crash occurs, the information that follows it is spread across phone calls, messages, police and hospital registers and the notes of road-owning agencies, and each channel may describe the same event in a different format.

National efforts to consolidate this information exist. The Integrated Road Accident Database (iRAD), developed for MoRTH, lets police, hospitals and road-owning agencies enter crash data into one national database [2], and the National Highways Authority of India (NHAI) publishes highway information and manages the network [3]. Those platforms are institutional. A student-scale or regional system that lets a citizen submit a report, lets an administrator review it, shows stored incidents on a map and lets hospitals and ambulance providers join through a verified route is a different and smaller problem, and it is the one addressed here.

This paper describes a web-based National Highway Accident Management System built as an academic prototype. It is intended to help organise and inspect submitted reports; it does not replace emergency telephone services, police procedure, hospital systems or official road-authority platforms. The contributions are: (i) a three-tier design that separates the browser interface, the API and the database, with the API as the single trust boundary; (ii) a role model in which partner privileges depend on a recorded administrator decision rather than on a role chosen at registration; (iii) a report-handling convention that treats every stored report and every map marker as a submitted record, not a confirmed live incident; and (iv) a documented data model, security design, risk analysis and sixteen-case test plan. The prototype has not yet been measured for performance or accuracy, and the paper says so wherever it matters.

The remainder of the paper is organised as follows. Section II reviews related concepts and states the gap. Section III presents the methodology: requirements, architecture, workflows, data model, security design and implementation environment. Section IV reports the observations available from the running prototype and the test plan. Section V discusses them, and Section VI concludes with limitations and future scope.

A. Problem Definition

The problem is to design and build a web system that lets authorised users report and review National Highway accidents, view reported incidents geographically, see trends and highway information, and onboard hospitals and ambulance services through an administrator approval process. Three difficulties shape the design. First, reports arrive in inconsistent formats and locations are described loosely, so duplicates and gaps are common. Second, a report is a claim; it must not be presented as a verified fact until someone has reviewed it. Third, emergency partners need elevated access, which creates a trust and privacy risk if any organisation can claim the role.

B. Scope

The system covers account registration and login, accident report submission with optional location and photograph, a register of reported accidents, accident trend summaries, highway information, map-based display of stored incident coordinates, a hospital information panel, an administrator function and the partner application and approval workflow. It does not provide ambulance dispatch, guaranteed live tracking of vehicles, integration with government emergency call centres or hospital admission systems, predictive forecasting, payment processing or legally authoritative accident certification. These require external agreements, trusted live data and further security controls.

II. LITERATURE REVIEW

A. Digital Accident Reporting

Digital reporting replaces paper and ad hoc messages with structured forms that capture time, place, incident category, vehicles, injuries and a description. Fixed fields make later filtering and analysis possible, and validation keeps malformed values out of the dataset. National systems such as iRAD apply this idea at institutional scale by giving every contributing agency the same data-entry structure [2]. A status field, for example Submitted, Under Review, Verified or Rejected, separates a new claim from a reviewed one, which matters because a submitted report cannot be assumed to describe a confirmed accident.

B. Web Mapping

A Geographic Information System stores and displays information tied to coordinates, and browser mapping libraries place markers on a map from data returned by an API. Open map data such as OpenStreetMap [4] and a lightweight library such as Leaflet [5] make this possible without proprietary services, subject to the data licence and tile-usage policy. A map is only as reliable as its coordinates. Approximate or user-entered positions should be labelled as such, and a record without coordinates should stay in a list rather than be placed at an invented location.

C. Role-Based Access Control and Web Security

Role-based access control assigns permissions to roles rather than to individuals [6]. In a web application the role shown in the browser is not protection by itself: the server must verify the authenticated identity and the permission on every protected request, because any client-side check can be bypassed. The OWASP Top Ten lists broken access control and injection among the most common web risks [7], and its guidance on file upload recommends restricting file type and size, generating safe filenames and storing uploads away from executable paths [8]. Token-based sessions commonly use JSON Web Tokens [9], which carry signed claims; a token that embeds a role can go stale when the role changes, so the server should return the authoritative role and approval state after login.

D. Document Databases

MongoDB stores records as flexible documents, which suits accident reports, user accounts and partner applications whose fields are mostly structured but may carry nested values [10]. Its flexibility moves the burden of consistency to the application layer: schema validation, indexes on frequently queried fields and careful updates keep the data usable as the number of records grows.

E. Research Gap

The concepts above are well established individually. What the reviewed sources do not give a student project is a single, small, inspectable system that joins citizen reporting, map display, administrator review and verified onboarding of emergency partners, while stating clearly what a stored report is and is not. This statement is limited to the sources consulted. The system presented here tries to supply that integration with three explicit rules: a report is a record, not a verified incident; a partner role is granted only by a recorded administrator decision; and every role decision is enforced by the backend. Table I places these concepts beside the corresponding design response.

TABLE I. CONCEPTS REVIEWED AND THE DESIGN RESPONSE

Concept	What the literature stresses	Design response
Digital reporting	Structured fields, validation, status values	Report form with required fields and status values; no report treated as confirmed
Web mapping	Marker accuracy depends on coordinates	Markers only for valid coordinates; others stay in the list
Access control	Server must check every request	Authentication and role middleware on protected routes
Partner trust	Unverified organisations are a risk	Application stays pending until an administrator decides
Data protection	Collect little, restrict access, validate uploads	File type and size limits; role-limited detail views; audit fields

III. METHODOLOGY

A. Development Approach

Development followed an iterative approach. Work was divided into requirement analysis, interface design, API and database development, module integration, testing and refinement, and each iteration took a small group of features and checked how the front end, API and database interact.

B. Requirements

Twelve functional requirements were set. A user shall be able to register and sign in (FR1) and sign out through protected navigation (FR12). An authenticated citizen shall be able to submit an accident report (FR2), required fields shall be validated before storage (FR3) and records shall be stored in MongoDB (FR4) and shown to authorised users (FR5). Markers shall be displayed for records with valid coordinates (FR6), and trend and summary views (FR7) and National Highway information pages (FR8) shall be provided. A user shall be able to apply for emergency-partner access (FR9), an administrator shall be able to approve or reject applications (FR10), and pending or rejected applicants shall be prevented from using partner-only features (FR11).

The non-functional requirements cover usability, security, performance under demonstration load, reliability through controlled error messages, maintainability through modular code, scalability through indexes, pagination and server-side filtering, and privacy through role-limited visibility of personal and verification data.

C. System Architecture

The system uses a three-tier architecture (Fig. 1). The presentation tier is a React application in the browser. The application tier is a Node.js and Express REST API that performs authentication, validation, business rules and role checks. The data tier is MongoDB, which holds users, accident reports, highway information and partner applications [10]–[13]. The browser never connects to the database directly, so the API is the trust boundary. Map tiles and hospital data come from external services and are treated separately from the core database.

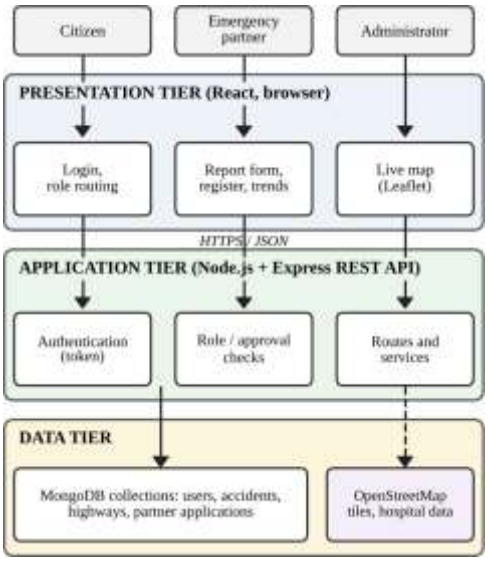


Fig. 1. Three-tier architecture and the three user roles.

D. User Roles

A citizen can register, sign in, view permitted dashboard information and submit accident reports. An emergency partner, meaning a hospital or an ambulance or emergency-transport provider, can submit an application and, after approval, use the partner functions assigned by the system. An administrator reviews accident information and partner applications, records decisions and manages authorised administrative functions. A user who has applied for partner access remains a citizen until the approval is recorded and the backend grants the partner role.

E. Reporting Workflow

A signed-in citizen opens the report form and enters the highway, incident date and time, location or landmark, incident type, number of injuries and optional coordinates and photograph (Fig. 2). The front end checks required fields and sends the request with the authentication token. The back end verifies the token, validates the payload, stores a report document with its creation time and reporter reference, and returns a reference that identifies the record in later review. An administrator reviews the report and updates its status. The record then appears in the register and in dashboard and map queries, and a map marker is drawn only if the record carries valid coordinates.

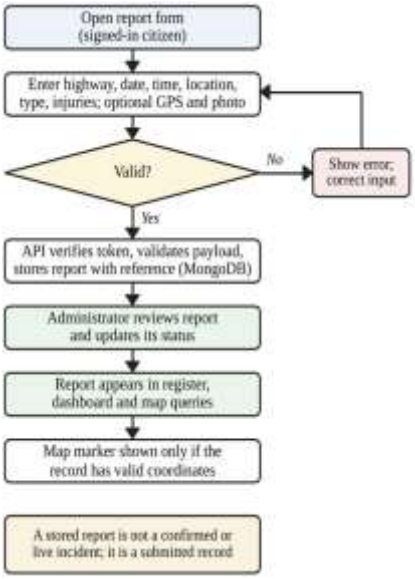


Fig. 2. Accident reporting workflow.

F. Partner Application Lifecycle

An applicant chooses Hospital or Ambulance / Emergency Transport and supplies the organisation name, registration or authorisation number, contact person, official contact number and a verification document. The back end stores the application with a Pending status. An administrator reviews the details and the document and records Approved or Rejected with an optional remark; the decision stores the administrator identifier and a timestamp for traceability. Only an approved application grants the corresponding partner role. The applicant can view the status and any remark, and partner-only screens are available only after approval.

G. Data Model

Four groups of MongoDB documents are used (Table II). Users hold identity, role, requested role and partner approval state. Accidents hold the reporter reference, incident date and time, highway, location, coordinates, type, description, injury and fatality counts where captured, status and timestamps. Highways hold route number, name, region and optional reference coordinates. Partner applications hold the applicant reference, category, organisation details, document reference, status, remarks, decision time and the deciding administrator. In the data dictionary the partner approval state takes the values not-required, pending, approved or rejected, and the report status takes the values Submitted, Under Review, Verified or Rejected.

TABLE II. PRINCIPAL DOCUMENT TYPES AND KEY FIELDS

Collection	Key fields
Users	name, email, phone, password hash, role, requestedRole, partnerApproval, organisation information, timestamps
Accidents	reporter, incident date and time, highway, location, coordinates, type, description, injuries, fatalities, status, timestamps
Highways	route number, name, state or region, segment description, optional reference coordinates
Partner applications	applicant, category, organisation name, registration number, contact person, official phone, document reference, status, adminRemarks, decidedAt, decidedBy

H. Authentication and Authorisation

Authentication confirms who the user is and authorisation decides what that user may do. After login the back end issues a signed token and a minimal profile that includes the role and approval status [9]. Protected routes pass through authentication middleware that identifies the user and authorisation middleware that checks the role and, for partners, the approval state. Hiding a menu item in React is a convenience, not a security boundary. Secrets and database credentials are held in environment variables and kept out of source control.

I. Map and Location Design

Markers are created from valid latitude and longitude values returned by the API, using Leaflet with OpenStreetMap tiles [4], [5]. A marker carries a short incident summary and links to a permitted detail view. Records without coordinates remain accessible in the list, and the interface states that markers represent submitted records and does not claim continuous live tracking, because no real-time location feed is connected. A hospital panel lists healthcare facilities near the map view, and a restricted panel for marking a victim as admitted is shown only to authorised partner roles.

J. Security Design and Risk Analysis

The security design follows the practices in Section II-C [7], [8]: HTTPS in deployment; password hashing with a recognised algorithm; validation and sanitisation of input; rate limits on authentication routes; ownership and role checks on the server; restricted upload type and size with safe filenames and storage outside executable paths; non-public verification documents; administrative audit fields; database backups; and error responses that reveal no secrets or stack traces. A risk analysis identified twelve risks, summarised in Table III.

TABLE III. PRINCIPAL RISKS AND PLANNED MITIGATIONS

Risk	Impact	Mitigation
Inaccurate or fabricated reports	Unreliable records	Mandatory fields, validation, administrator verification before a report is treated as verified
Wrong coordinates	Misplaced markers	Manual selection, coordinate validation, user correction before submit
Duplicate reports	Distorted statistics	Duplicate detection on location, date, time and details; administrator merge or reject
Unauthorised access	Data exposure	Authentication, server-side role checks, password hashing, HTTPS
Unsafe uploads	Security and storage issues	Type and size limits, server validation, safe filenames
Server or database failure	Interruption or data loss	Backups, error handling, monitoring
Map service failure	Map unavailable	Graceful error handling; list view keeps essential location data
Outdated hospital data	Misleading information	Administrative update and periodic verification

K. Implementation Environment

The front end is written in React and JavaScript with Vite for local development [11]. The back end uses Node.js and Express [12], [13] and MongoDB for storage, inspected through MongoDB Compass. Visual Studio Code is the editor, and API requests are tested with Thunder Client or Postman. The backend connection string is read from an environment variable, a clear start-up error is logged if the connection fails, and registration, login and data routes must use the same database. Deployment requires configuring the front-end origin, API base URL, token secret, upload storage and map settings, with CORS limited to the intended origin.

IV. RESULTS

This section reports only what the prototype showed and what has been planned. No quantitative measurement has been carried out, and the planned tests have not been presented as passed.

A. Observed Prototype Behaviour

Screenshots of the running prototype show five working views. The account-selection page separates citizen access from the emergency-partner path. The Report Accident form accepts highway, date, time, location, incident type and injuries. The Reported Accidents register lists stored reports with their highway, place, type, injuries and status. A record detail view shows the stored values, the uploaded photograph and a section for victim and patient records. The Live Map renders stored incidents over OpenStreetMap tiles with a hospital panel and a role-restricted panel for admitting victims. The register in the screenshots holds a single test record, a hit-and-run on NH-44 near Yelahanka with one injury and the status Reported, so the observations demonstrate the end-to-end path from form to register to detail view but not behaviour at volume. Table IV summarises what each view shows.

TABLE IV. VIEWS OBSERVED IN THE PROTOTYPE

View	What the screenshot shows	Evidence limit
Account selection	Citizen and emergency-partner paths on one page	Single screenshot
Report form	Fields for highway, date, time, location, type, injuries	Form only; validation not shown
Report register	Stored report listed with highway, place, type, injuries, status	One test record
Record detail	Stored values, photograph, victim and patient section	One test record
Live map	OSM tiles, hospital panel, restricted admit panel	Marker behaviour at volume untested

B. Planned Evaluation

A sixteen-case test plan was prepared (Table V) and is to be run on a separate test database with non-sensitive sample data. The functional cases cover registration, login, report submission and validation, the register, the map, trends and the partner workflow. The security cases verify that expired or invalid tokens are rejected, that a citizen cannot call administrator endpoints, that a pending partner cannot reach partner-only routes, that a user cannot change another user's restricted record and that uploaded documents cannot be executed or fetched publicly. Candidate metrics are the successful-submission rate, API response time under a defined load, the share of records with valid coordinates, the duplicate-report rate, the correctness of dashboard aggregates against a known dataset and the time to review a partner application. None of these has been measured yet.

TABLE V. FUNCTIONAL AND SECURITY TEST PLAN (NOT YET EXECUTED)

ID	Test	Expected outcome
T1	Open the application	Account selection or welcome screen shown
T2–T3	Register a valid citizen; register an existing email	User created once; duplicate rejected with a clear message
T4–T5	Login with valid credentials; with unknown email	Session issued; user-not-found, no false success
T6–T7	Submit a valid report; an incomplete report	Record saved and returned; validation error, nothing stored
T8–T10	Open register, map and trends	Permitted records listed; markers at stored coordinates; aggregates match test data
T11–T12	Submit partner application; use partner features while pending	Status Pending; access denied
T13–T14	Administrator approves; administrator rejects	Metadata stored and role granted; status stored and privileges stay disabled
T15	Citizen calls an admin API	Forbidden response
T16	Upload an unsupported or oversized document	Upload rejected

V. DISCUSSION

A. What the Prototype Demonstrates

The prototype shows that reporting, review, map display and partner onboarding can be joined in one browser application over a document database. Common forms and a central store address the fragmentation described in Section I, and the map and register make stored reports easier to inspect than separate lists. The most important design choice is that the API, not the interface, decides who may do what: the partner role depends on a recorded administrator decision, and role-aware navigation only reflects what the server already allows.

B. Record Versus Verified Incident

A stored report is a claim made by a user. Treating it as a record with a status, rather than a confirmed or live incident, protects administrators and users from acting on unreviewed information and keeps the map honest. A marker shows where a report says an incident happened, and it may be old, wrong or duplicated. Operational use would need a trusted real-time source and an agreed verification procedure.

C. Privacy and Ethics

Accident records can contain personal and patient details, and they may be distressing or inaccurate. The system therefore limits detail views by role, keeps verification documents non-public, collects location only when needed, uses neutral wording and should provide a way to correct a record. It should not be presented as an official government service unless formally authorised, and retention rules must be defined before any real data is held.

D. Practical Lessons

Integration issues were the main practical difficulty: mismatched route names, stale tokens that still carry an old role, inconsistent database names, wrong API base URLs, missing model fields and middleware applied in the wrong order. A form is not complete until its route, validation, model, retrieval query and user feedback all work together, and authentication must be tested against real database records. Navigation and backend security have to be designed together.

VI. CONCLUSION

This paper presented a web-based National Highway Accident Management System that combines citizen reporting, an accident register, map visualisation, highway information, summary analytics and an administrator-controlled onboarding workflow for hospitals and ambulance services. The React, Node.js and Express, and MongoDB architecture keeps the interface, business rules and storage separate, and enforces roles at the API. Screenshots show the main views working on a test record. No performance, accuracy or usability results exist yet, and the prototype is an information-management aid, not a replacement for emergency services or authoritative accident records.

A. Limitations

1) No quantitative evaluation has been carried out; the sixteen-case plan has not been executed and the screenshots show a single test record.
2) Usefulness depends on timely, accurate and complete reports, and a map marker is not proof that an incident is current or officially verified.
3) The system cannot guarantee emergency response, ambulance availability or hospital bed availability without trusted integrations and operational agreements.
4) Duplicate detection, pagination, audit trails and document review are planned rather than demonstrated.
5) Geographic accuracy, data retention and privacy controls need continued review before any real deployment.

B. Future Scope

The following directions follow from the limitations; none is a current capability. They are arranged in the phases of the project roadmap.

1) Stabilise authentication, database configuration and core report operations, then complete map

accuracy, highway references, filtering and analytics validation.

2) Complete partner application review with document security, a decision audit trail and notifications.
3) Run usability, security and load testing with representative users and report the metrics in Section IV-B.
4) Evaluate integration with verified road-authority data, ambulance dispatch and hospital availability feeds, and prepare a controlled pilot.
5) Add duplicate detection, multilingual and accessible interfaces and a mobile application; predictive analytics would be considered only after a sufficiently large validated dataset exists.

ACKNOWLEDGMENT

The authors thank Dr. Souvik Das, Assistant Professor, Department of Artificial Intelligence and Machine Learning, and Aditya College of Engineering and Technology, Bangalore, for guidance and support throughout this project.

REFERENCES

[1] Ministry of Road Transport and Highways, Government of India, "Road Accidents in India 2022," Transport Research Wing, New Delhi. [Online]. Available: https://morth.nic.in/

[2] Ministry of Road Transport and Highways, "Integrated Road Accident Database (iRAD)," developed by RBG Labs, IIT Madras. [Online]. Available: https://coers.iitm.ac.in/irad/

[3] National Highways Authority of India, official website and publications. [Online]. Available: https://nhai.gov.in/

[4] OpenStreetMap Contributors, "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org/

[5] V. Agafonkin, "Leaflet: an open-source JavaScript library for interactive maps." [Online]. Available: https://leafletjs.com/

[6] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," Computer, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[7] OWASP Foundation, "OWASP Top Ten Web Application Security Risks." [Online]. Available: https://owasp.org/www-project-top-ten/

[8] OWASP Foundation, "File Upload Cheat Sheet." [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

[9] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7519

[10] MongoDB, Inc., "MongoDB Manual," documentation. [Online]. Available: https://www.mongodb.com/docs/

[11] Meta Platforms, Inc., "React Documentation." [Online]. Available: https://react.dev/

[12] OpenJS Foundation, "Node.js Documentation." [Online]. Available: https://nodejs.org/docs/

[13] Express.js, "Express Web Framework Documentation." [Online]. Available: https://expressjs.com/