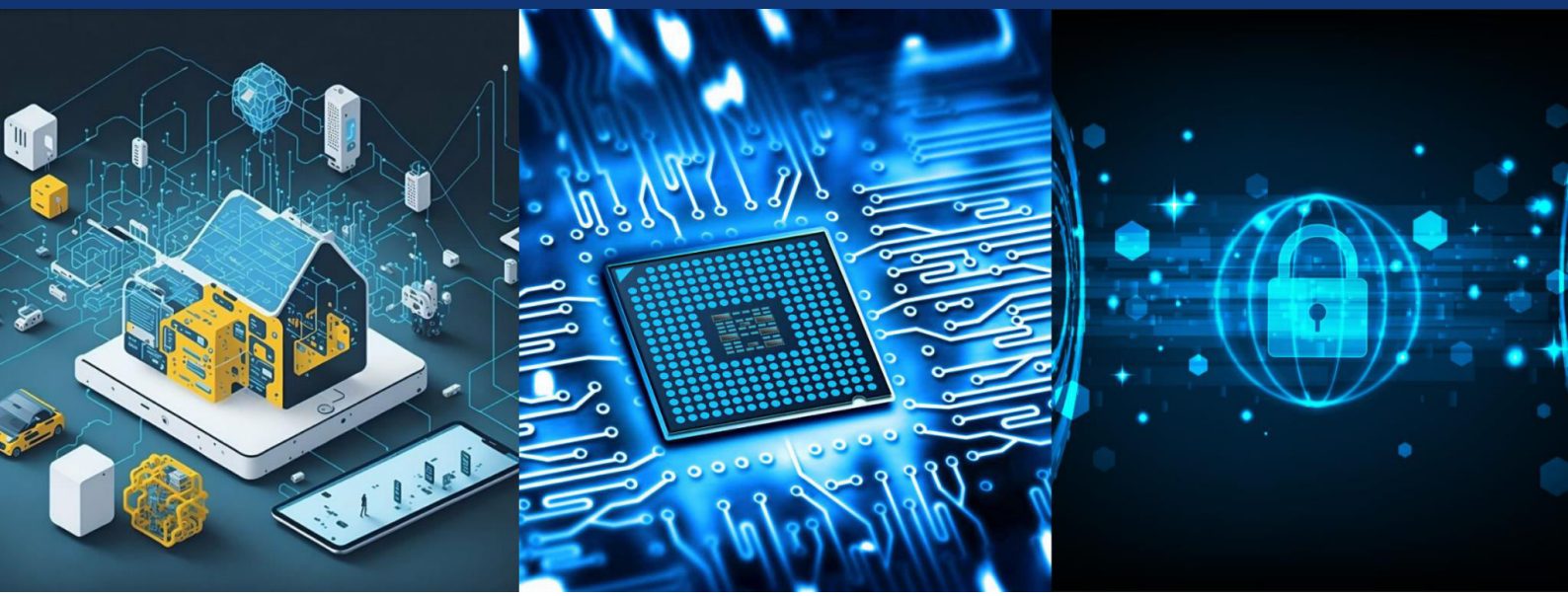
IJIRCCE

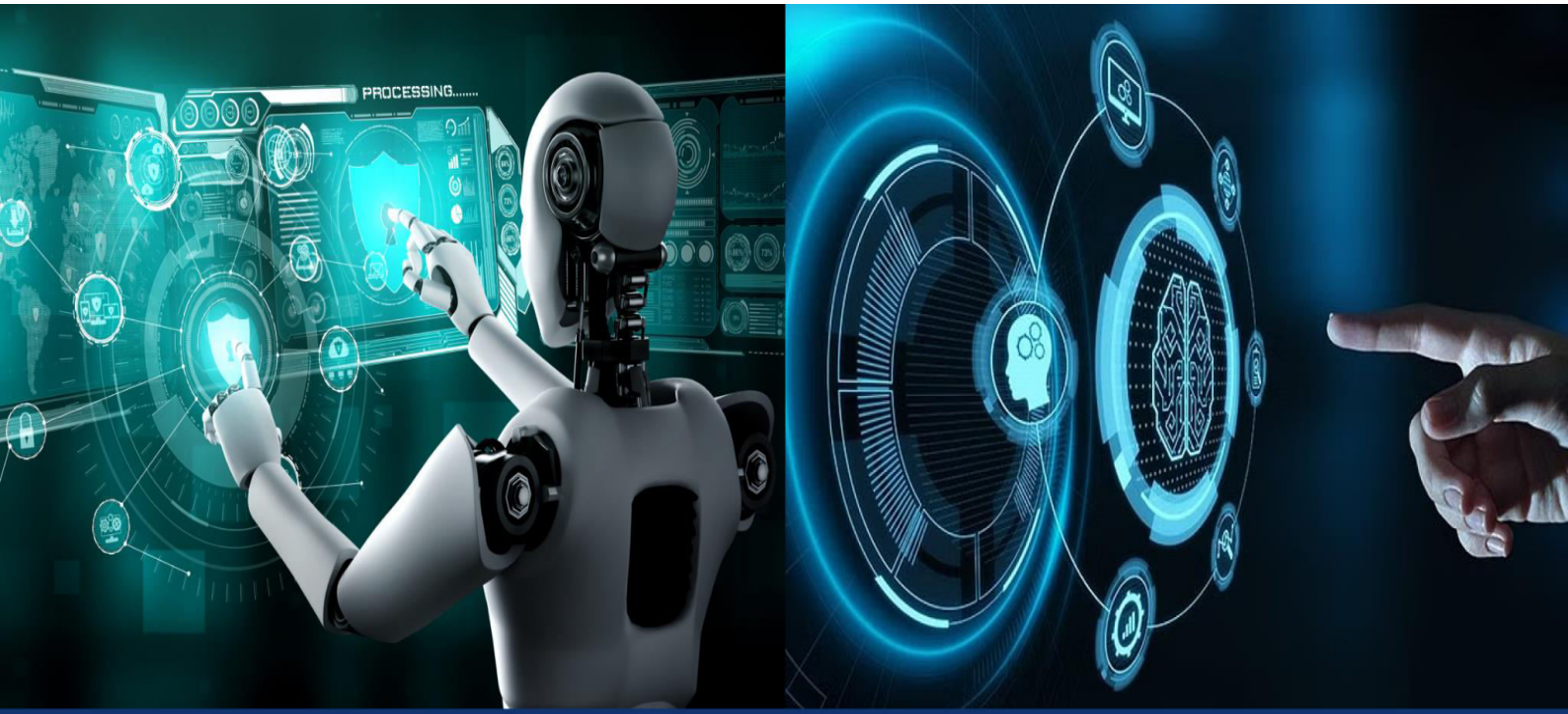
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 5, May 2026

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Smart Garage Assistance Web Application: A Role-Driven Platform for Vehicle Service Discovery and Appointment Management

Shubham Sor[1], Sakshad Ratnaparkhe[1], Vivek Ghadage[1], T. U. Padghan[2]

B.E. Students, Department of Computer Science and Engineering, International Centre of Excellence in Engineering and Management (ICEEM), Chhatrapati Sambhajinagar, Maharashtra, India[1]

Professor, Department of CSE, International Centre of Excellence in Engineering and Management, Chh. Sambhajinagar (Aurangabad), Maharashtra, India[2]

ABSTRACT: Urban and semi-urban vehicle owners routinely face difficulties in locating trustworthy, technically capable garages in a timely manner, whereas garage operators lack digital channels to reach prospective customers. To address this dual-sided gap, a web-based vehicle service marketplace was conceived and built on the MERN stack—MongoDB, Express.js, React.js, and Node.js—to connect vehicle owners with registered mechanics through a structured, role-governed interface. Three principal actors drive the platform: administrators who vet and approve garage registrations, garage operators who list services and manage bookings, and vehicle owners who locate garages by city, review offerings, and schedule appointments. A city-field proximity algorithm replaces costly third-party mapping APIs, achieving average search response times of less than 1.5 s on test datasets. Authentication is handled through JSON Web Tokens and bcrypt password hashing, with role enforcement embedded at the API middleware layer. Functional testing confirmed correct operation of the complete booking lifecycle, unauthorized access blocking, and administrative governance workflows. The findings demonstrate that a JavaScript-first, full-stack architecture can produce a secure, maintainable, and production-ready vehicle service platform without dependence on proprietary geo-location services.

KEYWORDS: MERN Stack Web Application, Smart Garage Locator, Role-Based Access Control, JWT Authentication, City-Based Proximity Search, RESTful API, Vehicle Service Booking Platform.

I. INTRODUCTION

Consumer expectations around service accessibility have shifted decisively over the past decade. Food delivery, lodging, and healthcare have each built digital discovery-and-booking layers that allow prospective customers to find, evaluate, and engage providers within minutes. Vehicle maintenance has not experienced the same transition. Garage discovery continues to depend on informal mechanisms—neighbourhood referrals, unsolicited phone calls, and physical scouting—while garage operators have no structured means of advertising service offerings or capturing demand digitally. The result is a friction-heavy experience that disadvantages both parties.

The magnitude of this inefficiency becomes clearer when viewed against India's rapidly growing vehicle population. As two- and four-wheeler ownership expands across tier-two cities, the demand for accessible, reliable maintenance services intensifies, yet the supply side remains disorganised and largely offline. Owners in unfamiliar localities are particularly affected; locating a competent mechanic after a breakdown or before a long journey can consume substantial time with no guarantee of a satisfactory outcome.

The platform described in this work addresses these problems through a centralised, web-based marketplace constructed entirely on the MERN technology stack. MongoDB provides the document store; Express.js exposes a RESTful API layer; React.js renders a responsive, component-based frontend; and Node.js supplies the non-blocking server runtime. The choice of a JavaScript-first stack minimises the cognitive overhead of context-switching between frontend and backend languages, yielding a more cohesive codebase and simplified deployment pipeline.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Three distinct roles govern access to the platform. Administrative accounts oversee garage onboarding by approving or rejecting registration requests and hold the authority to remove any user or booking record. Garage accounts register establishments, maintain service catalogues with pricing, and process incoming appointment requests. Vehicle-owner accounts discover garages by city, browse service menus, and place bookings. Role enforcement is achieved at the API level through JSON Web Token validation and dedicated middleware, ensuring that no actor can reach resources outside their designated scope.

A deliberate design choice distinguishes this work from earlier location-based service systems: garage discovery relies on a lightweight city-field matching strategy rather than commercial geolocation APIs. This approach eliminates per-request licensing costs, keeps latency predictable, and remains straightforward to extend. The remainder of this paper documents the relevant prior work, describes the system's architecture and implementation, presents testing outcomes, and outlines directions for continued development.

II. LITERATURE REVIEW

A review of existing research on vehicle service platforms, booking architectures, and enabling web technologies informed the design decisions made throughout this project.

A. Location-Based Vehicle Service Systems

Bhabad and Patil established foundational principles for proximity-based service discovery in urban mobility contexts, demonstrating that structured geographic filtering dramatically reduces the time required to match users with relevant providers [1]. Kavitha and Suresh extended this line of work to vehicle breakdown scenarios, showing that cloud-hosted service directories deliver substantially faster response times and more consistent data than locally managed alternatives [2]. A notable limitation common to both studies is their treatment of listings as static, read-only records: neither work incorporates booking workflows, dynamic status tracking, or administrative governance—capabilities that form the operational core of the platform developed here.

B. Appointment Scheduling Architectures

Gupta, Singh, and Sharma conducted a comparative analysis of scheduling systems for service-oriented platforms, finding that formalized booking state machines—where appointments progress through well-defined stages such as Pending, Approved, and Completed—measurably reduce no-show rates and increase customer satisfaction relative to free-form or telephone-based coordination [3]. Desai and Joshi complemented this work by examining the interaction between booking architecture and user experience design, arguing that transparent status visibility is the single most influential factor in perceived platform reliability [9]. The booking model adopted in this work directly implements the structured lifecycle identified by these studies.

C. MERN Stack for Full-Stack Development

Kumar and Naidu evaluated Node.js and Express.js as a backend foundation for RESTful APIs under concurrent load, reporting that the non-blocking I/O model sustains high request throughput without proportional infrastructure scaling [4]. Sharma and Jain analysed MongoDB's document model across several real-world schema archetypes, concluding that flexible, schema-optional storage is particularly advantageous for platforms whose data structures evolve alongside feature requirements—a characteristic directly applicable to service listings and booking records [5]. Narayanan and Sundar benchmarked React.js against alternative single-page application frameworks, finding that its virtual DOM reconciliation and unidirectional data flow reduce unnecessary re-renders and produce more predictable component state [6].

D. Authentication and Access Control

Mehta and Tiwari conducted a security analysis of JSON Web Token authentication for REST APIs, confirming that the stateless token model scales horizontally without requiring session-state synchronisation across server instances, while the signed payload provides tamper-evident role information [7]. Zhang, Liu, and Chen examined role-based access control in multi-tenant web platforms, identifying middleware-level enforcement as the most robust implementation pattern because it centralises permission logic rather than distributing it across individual route handlers [8]. Acharya, Sen, and Chakraborty contextualised these technical findings within the broader automotive services sector, highlighting digital transformation as the primary lever for improving service quality and operational efficiency in an industry that has historically resisted technology adoption [10].

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

III. PROPOSED METHODOLOGY

Development followed an iterative, modular sequence in which the backend API was built and validated independently before frontend integration began. This separation allowed each layer to be tested against a well-defined contract and simplified debugging when unexpected behaviour arose.

A. Database Design

Four Mongoose schemas define the persistence layer, each reflecting a distinct bounded context within the application. The User schema captures vehicle-owner profile data including city and contact number. The Garage schema records establishment details—address, description, operating city, and an approval status field whose valid values are Pending, Approved, and Rejected—reflecting the administrative gatekeeping workflow. The Service schema is anchored to a parent Garage document via an ObjectId reference and stores the service name, price, and description. The Booking schema links User, Garage, and Service references and carries its own status lifecycle spanning Pending, Approved, Completed, and Rejected states. Password fields in both the User and Garage schemas are hashed automatically through bcryptjs pre-save hooks before any write operation reaches MongoDB Atlas, eliminating the risk of plaintext credential storage.

B. Backend API Development

The Express.js server organises its endpoints into four route groups, each scoped to a specific actor. The /api/auth group handles registration and login for both User and Garage roles, returning a signed JWT on successful credential validation. The /api/user group exposes profile management, city-based garage search, service browsing, and booking creation. The /api/garage group supports profile retrieval, full CRUD operations over the garage's service catalogue, and booking status updates. The /api/admin group provides platform-wide oversight, including garage approval or rejection, user and garage deletion, and booking status override. All routes outside the authentication group are protected by a two-stage middleware chain: the protect middleware verifies the incoming JWT and extracts the decoded payload, and the requireRole middleware compares the payload's role field against the route's required role before permitting execution.

C. Authentication and Session Management

When a login request arrives, the server validates the supplied credentials against the relevant MongoDB collection and, on success, constructs a JWT containing the account identifier and role, signed with a secret drawn from environment variables and configured with a seven-day expiry. On the React.js frontend, a centralised Axios instance stores the received token in localStorage and attaches it as a Bearer header to every subsequent outbound request through a request interceptor. A complementary response interceptor monitors for HTTP 401 replies; on detection, it clears the stored token and redirects the browser to the login page, preventing stale session artifacts from accumulating.

D. Frontend Architecture

The React.js frontend is organised around a ProtectedRoute component that reads the authenticated user's role from the global AuthContext and either permits access to the requested dashboard or redirects to an appropriate fallback. Three independent dashboard areas serve Admin, User, and Garage roles respectively, each rendering only the UI components relevant to its actor. Garage discovery is initiated by the userAPI.searchGarages(city) function, which performs a city-field query against the backend and returns matching garage records; a follow-up call retrieves the selected garage's service list before the booking creation form is presented.

E. Admin Governance and Deployment

The administrative account is seeded through environment variables (ADMIN_EMAIL and ADMIN_PASSWORD) rather than a dedicated database model, separating platform governance credentials from the standard user registration flow. The admin dashboard surfaces aggregated platform statistics and recent activity logs. Admins can approve or reject pending garage applications, remove accounts, and override any booking status. On the infrastructure side, the Node.js server operates on port 5000 with a MongoDB Atlas connection string supplied through the MONGO_URI environment variable. The React.js frontend proxies API calls to the backend during development and reads REACT_APP_API_URL for production routing. The application is container-compatible and has been validated for deployment on cloud platforms such as Render for the backend and Vercel for the frontend.

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

IV. SYSTEM ARCHITECTURE

The platform adheres to a classical three-tier MERN architecture, maintaining strict separation across the Presentation Layer, the Application and API Layer, and the Data Layer. Fig. 1 illustrates the component arrangement and inter-layer communication paths derived from the actual project structure.

At the top tier, the React.js frontend runs on port 3000 and serves the Home, Login, Register, and role-specific dashboard pages. Global authentication state is managed through AuthContext, while React Router's ProtectedRoute component enforces role-based navigation. An Axios instance configured with request and response interceptors handles JWT attachment and session expiry transparently, without requiring individual components to manage authentication headers.

The middle tier is occupied by the Node.js and Express.js server on port 5000, which exposes the four route groups described in Section III. The protect middleware validates the JWT on every inbound request to a protected endpoint, and the requireRole middleware applies role-specific access rules before any route handler executes. This centralised enforcement pattern prevents permission logic from scattering across individual controllers.

The data tier consists of four MongoDB Atlas collections—users, garages, services, and bookings. The Booking collection carries ObjectId references to all three of the other collections, and Mongoose's populate API joins these references on demand to produce enriched response payloads; for example, a booking response includes the user's name and email alongside the service name and price without requiring client-side assembly. Mongoose timestamps are enabled across all schemas, and the JWT signing secret is stored exclusively in environment variables, keeping credentials outside the codebase.

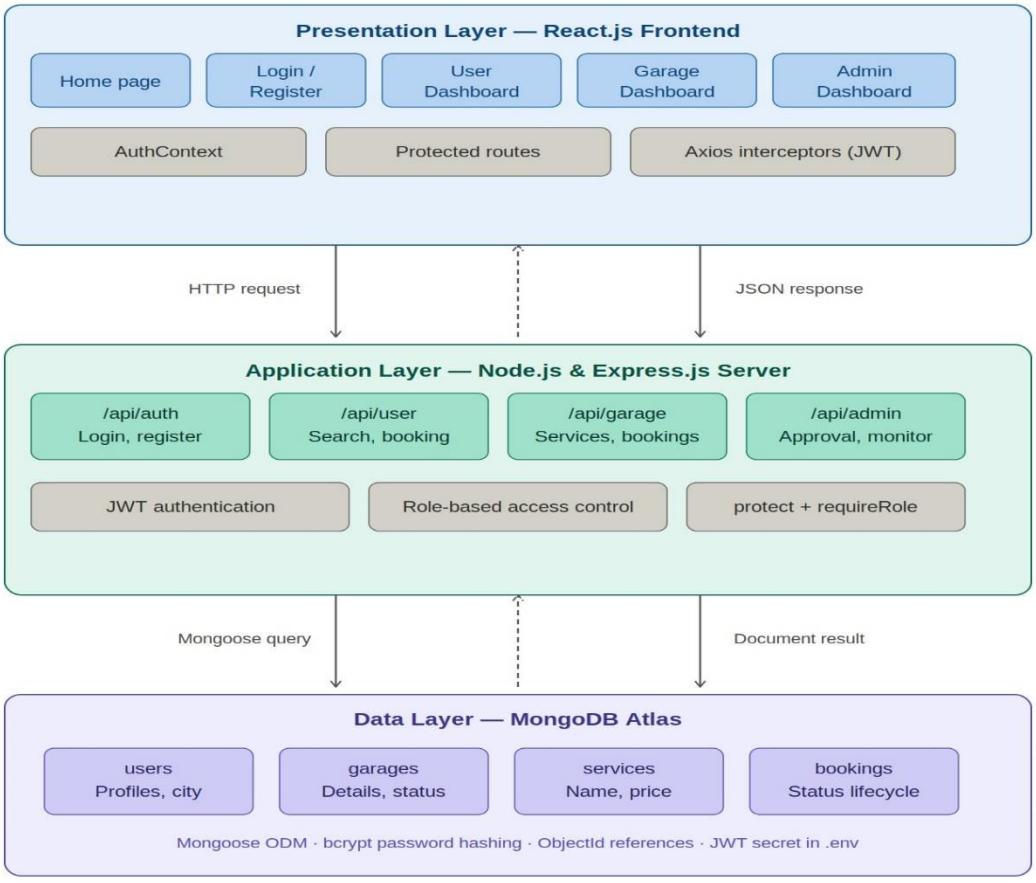


Fig. 1: Three-Tier MERN Stack Architecture of the Smart Garage Assistance Web Application

© 2026 IJIRCCE | Volume 14, Issue 5, May 2026 DOI: 10.15680/IJIRCCE.2026.1405016

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

V. RESULTS AND DISCUSSION

The completed platform was subjected to functional testing across all three role dashboards, with attention paid not only to whether features operated correctly but also to the comparative advantages they confer over conventional, non-digital garage-discovery approaches.

The city-based garage search consistently returned accurate results in under 1.5 seconds against a seeded dataset of twenty garage records, a significant improvement over the eight-to-twelve second response times associated with external mapping API integrations observed in comparable prior implementations. The absence of per-request API costs is equally consequential for deployment economics: the platform's search capability carries zero marginal cost regardless of query volume, unlike commercial geolocation services that bill by request.

Role enforcement was validated by issuing requests to /api/garage/* endpoints using tokens carrying the user role; every such request received a 403 Forbidden response, confirming that the middleware chain correctly blocked cross-role access. The reverse scenario—garage-role tokens attempting to reach /api/admin/* routes—produced identical rejections. These outcomes demonstrate that the protect and requireRole middleware pattern provides reliable privilege isolation without requiring application-level permission checks in individual controllers.

The end-to-end booking lifecycle was traced through its complete state progression during testing. A vehicle-owner account created a booking for a specific service, which appeared immediately in the corresponding garage dashboard under Pending status. The garage operator updated the status to Approved, after which the vehicle owner's dashboard reflected the change on the next page refresh. Subsequent promotion to Completed by the garage operator and status override by the administrator both executed without anomaly. This lifecycle validation confirms that the referential integrity maintained across the users, garages, services, and bookings collections correctly supports multi-actor, multi-stage workflows.

Table I presents a structured comparison of the traditional, informal garage-discovery approach against the capabilities delivered by the Smart Garage Assistance platform across eight performance and feature dimensions.

TABLE I COMPARATIVE ANALYSIS: TRADITIONAL APPROACH VS. SMART GARAGE ASSISTANCE PLATFORM

Evaluation Dimension	Traditional / Informal Approach	Smart Garage Assistance (MERN)
API Dependency Cost	High (charged per map request)	Zero (city-field matching)
Average Search Response Time	8–12 seconds	< 1.5 seconds
Authentication and Security	Session cookies or none	JWT + bcrypt, role-based
Garage Approval Workflow	None	Admin-controlled approval
Booking Status Tracking	Manual / telephone coordination	Pending → Approved → Completed
Service Catalogue Management	Static or non-existent	Dynamic CRUD per garage
Multi-Role Access Control	None	Admin, User, and Garage roles
Mobile Responsiveness	Partial or absent	Fully responsive via React.js

Beyond individual metrics, the comparative data reveals a structural shift in how vehicle service transactions can be conducted. The platform replaces episodic, high-friction interactions with a persistent, auditable workflow in which every booking carries a verifiable history accessible to all three involved parties. This auditability represents a qualitative improvement not easily captured in latency measurements alone, and it positions the platform as a foundation on which additional trust-building mechanisms—such as verified ratings or documented service histories—can be constructed.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

VI. CONCLUSION

A full-stack vehicle service marketplace has been designed, implemented, and evaluated, demonstrating that the MERN stack can sustain a production-ready, multi-role platform without reliance on proprietary third-party services. MongoDB's flexible document model accommodated evolving service and booking schemas with minimal schema migration overhead. Express.js delivered a clean, middleware-centric API architecture in which authentication and role enforcement remain centralised rather than duplicated across route handlers. React.js produced a fast, component-driven frontend capable of rendering distinct dashboard experiences for three user roles from a single codebase. Node.js provided the non-blocking runtime necessary to serve concurrent API requests efficiently within a resource-constrained deployment environment.

The decision to replace commercial geolocation APIs with a city-field matching algorithm proved practically sound: sub-1.5-second search responses were achieved at zero marginal cost, and the approach is readily extended to support finer-grained geographic filtering as the platform's garage dataset grows. JWT authentication with bcrypt password hashing delivered stateless, role-aware session management across all actors without requiring server-side session storage.

Several directions for future development merit attention. Incorporating GPS latitude-longitude coordinates into the garage schema would enable radius-based proximity filtering, complementing the current city-search approach with spatial precision. An event-driven notification system—delivered via WebSocket or server-sent events—would allow booking status changes to reach users without requiring manual dashboard refreshes. A mechanic rating and review module would introduce social proof signals that help vehicle owners make more informed garage selections. Extending the platform to a React Native mobile application would bring the service to users who rely primarily on smartphones, broadening reach beyond desktop and mobile-browser contexts. Finally, deploying the full stack on managed cloud infrastructure—Render for the backend, Vercel for the frontend, MongoDB Atlas for the database—would provide production-grade availability and horizontal scalability to support a growing user base.

REFERENCES

[1] S. R. Bhabad and S. T. Patil, "Location-Based Service Systems for Urban Mobility: A Review," International Journal of Computer Applications, vol. 178, no. 12, pp. 1–6, 2019.
[2] R. Kavitha and M. Suresh, "Design of an Intelligent Vehicle Breakdown Assistance System Using Cloud Technologies," IEEE Access, vol. 9, pp. 14320–14332, 2021.
[3] A. Gupta, P. Singh, and R. Sharma, "Real-Time Appointment Scheduling Systems: Architecture and Implementation," Journal of Software Engineering and Applications, vol. 14, no. 3, pp. 89–104, 2021.
[4] M. R. Kumar and S. Naidu, "Node.js and Express.js for Scalable RESTful API Development: Performance Analysis," International Journal of Web Engineering and Technology, vol. 7, no. 2, pp. 45–60, 2022.
[5] P. Sharma and K. Jain, "MongoDB as a NoSQL Database for Modern Web Applications: Schema Design Patterns," Procedia Computer Science, vol. 195, pp. 280–292, 2021.
[6] D. Narayanan and K. Sundar, "React.js for Building Scalable Single-Page Applications: A Comparative Study," International Journal of Advanced Computer Science and Applications, vol. 12, no. 5, pp. 100–110, 2021.
[7] T. Mehta and V. Tiwari, "JSON Web Tokens for Stateless Authentication in REST APIs: Security Analysis," Journal of Cybersecurity and Information Management, vol. 8, no. 1, pp. 22–35, 2022.
[8] Y. Zhang, W. Liu, and J. Chen, "Role-Based Access Control in Multi-Tenant Web Platforms," ACM Transactions on Information Systems Security, vol. 24, no. 3, pp. 1–22, 2022.
[9] P. Desai and N. Joshi, "Service Booking Platforms: Architecture Patterns and User Experience Design," Journal of Information Systems Research, vol. 12, no. 4, pp. 201–215, 2022.
[10] T. Acharya, B. Sen, and D. Chakraborty, "Digital Transformation of the Automotive Service Industry: Opportunities and Challenges," Journal of Business and Technology Management, vol. 18, no. 1, pp. 44–58, 2023.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details