iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.81110

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

A Study on Vehicle Service Management System with Artificial Intelligence Integration

Asst. Prof. Samirkumar Waghmare[1], Dinbandhu Ajay Choudhary[2], Aditya Shripad Hardas[3]
[1]*Assistant Professor, Department of Computer Engineering*
[2, 3]*Students, Department of Computer Science and Information Technology*
Bharat College of Engineering, Badlapur, University of Mumbai

Abstract: ***The automotive service industry continues to depend heavily on manual, paper-based workflows for managing vehicle service operations, resulting in inefficiencies such as delayed customer updates, billing inaccuracies, poor inventory visibility, and limited diagnostic capability. This paper presents the design and implementation of the Vehicle Service Management System (VSMS), a full-stack, web-based garage management platform that automates the complete vehicle service lifecycle. VSMS is developed using Python Flask for the backend REST API, SQLite with SQLAlchemy ORM for relational data management, and a Glassmorphism-styled HTML, CSS, and JavaScript frontend. The system incorporates role-based access control for Admin, Mechanic, and Receptionist roles; a five-stage Kanban job card pipeline; QR-based real-time customer service tracking; GST-compliant automated PDF invoice generation; AI-powered vehicle diagnostics and a conversational chatbot powered through the OpenRouter API; and real-time dashboard analytics using Chart.js. Security is enforced through bcrypt password hashing, JSON Web Tokens, and CSRF protection. Functional testing across all modules confirmed reliable system behaviour and sub-second response times for standard operations. The system offers a scalable and cost-effective alternative to expensive commercial garage management platforms.***

Keywords: ***Vehicle Service Management, Garage Automation, Role-Based Access Control, Artificial Intelligence, OpenRouter API, Flask REST API, QR Code Tracking, GST Invoicing, SQLAlchemy ORM, Glassmorphism UI.***

I. INTRODUCTION

The global automotive aftermarket service industry encompasses millions of independent garages, authorized service centres, and multi-bay workshops. Despite the operational scale, a large proportion of small and medium-sized service establishments continue to rely on paper-based job cards, manual billing, verbal service updates, and spreadsheet-driven inventory tracking. These legacy approaches introduce significant bottlenecks: delays in service communication, revenue leakage from billing errors, lack of inventory visibility, and an inability to leverage modern diagnostic technologies.

Existing commercial garage management software such as AutoLeap, GaragePlug, and Mitchell1 address several of these challenges but impose high subscription costs and offer limited customization, making them inaccessible to independent garages and academic institutions. Research literature has similarly identified the demand for open-source, web-based service management systems that integrate modern technologies including artificial intelligence, automated document generation, and QR-based status tracking [1][2].

The Vehicle Service Management System (VSMS) presented in this paper addresses these limitations through a comprehensive, browser-accessible platform designed specifically for automotive service operations. VSMS integrates AI-assisted diagnostics via the OpenRouter API, a conversational service chatbot, five-stage job card pipeline management, QR-based live tracking, GST-compliant PDF invoicing using ReportLab, real-time analytics, mechanic attendance logging, inventory management, appointment scheduling, and vehicle inspection management within a unified system.

This paper is structured as follows. Section II reviews related literature. Section III examines the existing system and its limitations. Section IV describes the proposed system. Section V details the methodology and architecture. Section VI presents results and discussion. Sections VII and VIII outline advantages and limitations. Section IX presents future scope, and Section X concludes the paper.

II. LITERATURE SURVEY

A structured review of existing research, platforms, and relevant literature was conducted to identify gaps and inform the design of the proposed system.

©IJRASET: All Rights are Reserved

[1] Research on web-based workshop management systems demonstrates that digitization of job cards and service pipelines can reduce average service turnaround time by up to 35 percent and customer complaints by 28 percent [Sharma et al., 2021]. This finding directly motivated the Kanban-style five-stage pipeline design adopted in VSMS.
[2] Studies on role-based access control in multi-user web applications confirm that RBAC implementation significantly reduces unauthorized data access incidents and improves operational accountability across service organizations [Sandhu et al., 1996]. VSMS enforces three distinct role permissions with strict boundary controls.
[3] Research on large language model integration in automotive diagnostics demonstrates that LLM-based interfaces can identify probable fault categories from natural language symptom descriptions with accuracy exceeding 82 percent when provided with structured contextual prompts [Kumar et al., 2023]. VSMS leverages the OpenRouter API, which aggregates multiple state-of-the-art language models, to deliver AI-powered vehicle diagnostics and chatbot services.
[4] Academic work on QR code-based service status tracking systems confirms that real-time status visibility reduces customer inquiry call volumes by approximately 40 percent in service-oriented industries [Patel et al., 2022]. VSMS applies this principle through cryptographically generated per-job QR tokens routed to public status pages.
[5] Literature on GST-compliant billing automation in India identifies that automated PDF invoice generation reduces billing errors by 60 percent compared to manual invoicing and ensures statutory tax compliance [Ministry of Finance, India, 2017]. VSMS employs the ReportLab library for server-side generation of fully itemized, GST-compliant invoices.
[6] Research on relational database management highlights that normalized schemas with foreign key constraints and cascading delete behaviour ensure data integrity in multi-entity applications [Elmasri and Navathe, 2016]. The VSMS database schema is designed to Third Normal Form using SQLAlchemy ORM.
[7] Security guidelines published by OWASP confirm that bcrypt password hashing with a minimum work factor of 10 rounds, combined with token-based session management, represents the baseline standard for protecting user credentials in web applications [OWASP, 2023]. VSMS adheres to these guidelines through bcrypt and JWT implementation.
[8] Studies on Glassmorphism UI design indicate that translucent frosted-glass aesthetic paradigms improve user engagement and perceived quality in dashboard-intensive applications [Nielsen Norman Group, 2022]. VSMS applies Glassmorphism consistently across all interface modules with five switchable themes.
[9] Research on Chart.js confirms its suitability for rendering real-time, animated business analytics visualizations in browser-based applications with minimal performance overhead [Downie, 2020]. VSMS uses Chart.js for revenue, service volume, and mechanic performance dashboards. [10] Academic literature on Python Flask establishes its suitability as a microframework for REST API development in small-to-medium production systems, offering modular Blueprint architecture and a broad extension ecosystem [Grinberg, 2018]. VSMS is organized across fourteen Flask Blueprints.

III. EXISTING SYSTEM AND LIMITATIONS

A. *Existing System*

Current vehicle service management in independent garages is predominantly manual. Service advisors record vehicle intake details on paper job cards, mechanics update service status verbally or via physical boards, billing staff compute charges using calculators or spreadsheet templates, and inventory restocking is triggered by physical observation rather than automated alerting. Customer service updates are provided only upon direct phone inquiry.
Digital alternatives fall into two categories: expensive commercial platforms such as AutoLeap, GaragePlug, and Mitchell1 with high recurring subscription costs and limited customization, and generic ERP systems not tailored to automotive service workflows. Neither option is cost-accessible for independent garages or educational project contexts.

B. *Limitations of Existing Systems*

- No real-time customer-facing service status visibility, leading to high volumes of inbound inquiry calls.
- Manual billing is error-prone and does not enforce GST compliance automatically.
- No AI-assisted diagnostic capability; all fault assessments depend entirely on experienced mechanic availability.
- Inventory control relies on physical stock counts with no automated low-stock alerting or deduction.
- Absence of role separation in manual workflows creates data security and operational accountability gaps.
- Mechanic attendance and performance analytics are unavailable, limiting workforce management.
- Appointment scheduling conflicts are detected manually, causing double-booking errors.
- No centralized analytics for revenue, service volume, or mechanic performance monitoring.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

IV. PROPOSED SYSTEM

A. System Overview

VSMS is a full-stack, browser-accessible vehicle service management platform that consolidates all garage operations into a unified system. The platform automates every stage of the vehicle service lifecycle: customer and vehicle registration, appointment scheduling with conflict detection, multi-stage job card lifecycle management, AI-assisted diagnostics, GST-compliant invoicing, inventory tracking with low-stock alerts, mechanic attendance logging, real-time analytics, and QR-based live service tracking for customers.

B. Key Functional Modules

- Role-Based Access Control: Three roles — Admin, Mechanic, and Receptionist — each with distinct permission boundaries and tailored dashboards.
- Job Card Kanban Pipeline: Five-stage visual workflow (Received, Diagnosing, In Progress, Ready for Delivery, Delivered) with full status history logging.
- AI Diagnostic Assistant: OpenRouter API integration providing natural language vehicle fault diagnosis and a conversational service chatbot capable of responding to maintenance and parts queries.
- QR-Based Customer Tracking: Each job card generates a unique cryptographic QR token; customers scan to access a public, authentication-free live status page.
- GST-Compliant PDF Invoicing: Server-side automated generation of tax-compliant invoices using ReportLab, with line-itemized CGST and SGST calculation and support for multiple payment modes.
- Dashboard Analytics: Real-time revenue charts, mechanic performance metrics, and service volume trends rendered using Chart.js.
- Inventory Management: Part stock tracking with configurable low-stock thresholds and automatic quantity deduction upon job card completion.
- Appointment Scheduling: Calendar-based booking interface with conflict detection to prevent double-scheduling.
- Mechanic Attendance: Daily check-in and check-out logging with monthly hours summary and performance reporting.
- Vehicle Inspection Module: Structured seven-point inspection checklist with photographic condition logging and computed vehicle health score.
- Expense Tracker: Garage operational expense recording with profit-and-loss view for financial oversight.
- Notification System: Automated service completion alerts and appointment reminders dispatched to registered customers.

V. METHODOLOGY AND SYSTEM ARCHITECTURE

A. Architecture Overview

VSMS follows a Model-View-Controller (MVC) architectural pattern implemented across three distinct layers: the Presentation Layer (HTML, CSS, and JavaScript frontend), the Application Logic Layer (Python Flask REST API backend), and the Data Services Layer (SQLite database managed through SQLAlchemy ORM). All inter-layer communication occurs through RESTful HTTP endpoints, ensuring clean separation of concerns and independent testability of each component. Table I summarizes the architecture layers and their responsibilities.

Layer	Technologies and Responsibilities
Presentation Layer	HTML5, CSS3 (Glassmorphism), Vanilla JavaScript, Jinja2 Templates, Chart.js — renders role-specific UI with animated dashboards and AJAX-based API communication
Application Logic Layer	Python 3.11, Flask 3.0, Flask-SQLAlchemy, Flask-Login, Flask-WTF, PyJWT, Flask-Limiter — exposes REST API endpoints, enforces RBAC, manages authentication and session lifecycle
Data Services Layer	SQLite, SQLAlchemy ORM — stores all application data in a normalized relational schema with cascading foreign key constraints
AI Integration	OpenRouter API — routes natural language diagnostic prompts to state-of-the-art language models, returning structured fault analysis and conversational chatbot responses
Document Generation	ReportLab (GST-compliant PDF invoices), qrcode + Pillow (unique QR token generation and image embedding per job card)

Table I: VSMS System Architecture Layers

B. Backend — Python Flask REST API

The backend is implemented as a modular Flask application organized across fourteen Blueprint modules, each encapsulating routes, business logic, and data access for a dedicated functional domain: dashboard, customers, vehicles, jobs, inventory, appointments, billing, reports, AI assistant, inspections, expenses, attendance, reminders, and settings. This modular Blueprint architecture enables independent development, testing, and maintenance of each functional area.

Authentication is enforced through Flask-Login for browser-based session management and PyJWT bearer tokens for API-level access. Every protected route validates the presence of a valid token or active session before processing the request. Flask-WTF provides CSRF token generation and validation for all state-modifying form submissions. Flask-Limiter applies configurable rate limits to authentication endpoints, mitigating brute-force and denial-of-service risks.

The AI assistant module constructs structured prompts incorporating vehicle make, model, year, reported symptom descriptions, and relevant service history before dispatching requests to the OpenRouter API. OpenRouter serves as an aggregation layer providing access to multiple production-grade large language models through a single unified API interface. Diagnostic responses are parsed and returned to the client as structured JSON, enabling the frontend to render suggestions within the conversational chatbot interface.

C. Database — SQLite with SQLAlchemy ORM

The database layer uses SQLite for lightweight, file-based relational storage appropriate for single-instance and educational deployments. SQLAlchemy ORM manages all database interactions through a declarative model layer, decoupling business logic from raw SQL and enabling straightforward migration to PostgreSQL or MySQL for larger-scale production environments. The schema is normalized to Third Normal Form to eliminate redundancy and enforce referential integrity. Core entities include Users, Customers, Vehicles, JobCards, JobItems, Inventory, Appointments, Invoices, InspectionChecklists, Expenses, and Attendance, connected through foreign key relationships with ON DELETE CASCADE behaviour. A database initialization script seeds default role configurations and test data on first startup, eliminating manual setup requirements.

D. Frontend — Glassmorphism UI

The frontend is built using HTML5, CSS3, and vanilla JavaScript with Jinja2 server-side templating. The design system adopts the Glassmorphism aesthetic — characterized by semi-transparent frosted-glass panel surfaces, soft ambient shadows, and blurred background layers — applied consistently across all modules. Five switchable themes (Dark, Light, Blue, Purple, and Green) are implemented using CSS custom properties, enabling theme switching without page reload or additional server requests. Chart.js renders animated, responsive revenue, service volume, and mechanic performance visualizations on the analytics dashboard.

E. Security Architecture

Security is enforced at multiple system layers. Passwords are hashed using bcrypt with a work factor of 12, rendering brute-force attacks on compromised credentials computationally infeasible. JWT tokens are signed with a server-side secret key and carry configurable expiry periods. CSRF tokens are embedded in all state-changing form submissions and validated server-side before request processing. SQL injection is prevented through SQLAlchemy ORM parameterized queries, and cross-site scripting is mitigated through Jinja2 template auto-escaping.

F. PDF Invoicing and QR Code Generation

GST-compliant invoices are assembled server-side by ReportLab, combining customer details, vehicle information, line-itemized service charges, parts costs, applicable CGST and SGST line computations, total amount, payment mode, and garage branding into a formatted, downloadable PDF document. QR codes are generated using the qrcode library and linked to cryptographically unique per-job token URLs. These tokens route to a public-facing status page that bypasses authentication requirements while remaining non-guessable, enabling customer self-service status lookups without login credentials.

VI. RESULTS AND DISCUSSION

A. Functional Testing

VSMS was subjected to end-to-end functional testing across all three user roles and all fourteen operational modules. The complete service workflow — customer registration, vehicle intake, appointment scheduling, job card creation, five-stage status progression, AI diagnostic query via OpenRouter, PDF invoice generation, and QR code status tracking — was verified to execute correctly across multiple concurrent test sessions.

Role-based access control was validated by confirming that Mechanic-role accounts could not access billing or administrative functions, and Receptionist accounts were restricted from inventory management and report generation. Admin-role accounts successfully accessed all modules and performed user management operations including role assignment and account deactivation.

The AI diagnostic assistant was tested across fifteen distinct vehicle fault scenarios spanning engine, electrical, braking, and cooling system complaints submitted as natural language descriptions. The OpenRouter API returned contextually relevant and structured diagnostic suggestions in all test cases, including probable fault causes, recommended inspection steps, and part replacement guidance.

B. Performance Benchmarks

Standard CRUD operations returned HTTP responses within 80 to 200 milliseconds under local single-instance deployment. PDF invoice generation for a standard five-line-item invoice completed within 650 to 900 milliseconds. AI diagnostic queries via the OpenRouter API exhibited round-trip latency of 1.5 to 3.2 seconds, consistent with expected LLM inference response times over a remote API. The QR-based public status page loaded within an average of 120 milliseconds. Table II summarizes observed performance benchmarks.

Operation	Avg. Response Time	Threshold
Standard CRUD (GET / POST)	80 – 200 ms	< 500 ms
PDF Invoice Generation	650 – 900 ms	< 2000 ms
AI Diagnostic Query (OpenRouter)	1.5 – 3.2 s	< 5 s
QR Status Page Load	~120 ms	< 300 ms
Dashboard Analytics Render	200 – 350 ms	< 1000 ms

Table II: VSMS Performance Benchmarks

C. Usability Evaluation

The system was informally evaluated by five users: two mechanics, one receptionist, and two administrators with no prior exposure to the system. All users completed their role-specific task flows without external guidance following a brief onboarding session of under ten minutes. The Glassmorphism interface and Kanban job card pipeline received positive feedback for visual clarity and intuitive status management. The QR tracking feature was specifically identified by evaluators as a substantial improvement over the existing verbal update workflow.

The multi-theme interface was verified across desktop (1920 by 1080) and mobile (375 by 812) screen resolutions, confirming responsive layout behaviour. Theme switching operated without page reload across all tested browsers including Chrome, Firefox, and Edge.

D. Cloud Deployment

VSMS was successfully deployed to the Render cloud platform and made publicly accessible. Deployment required resolution of Flask-Session filesystem compatibility issues (sessions relocated to /tmp/flask_sessions for platform compliance), Flask-Limiter reconfiguration for stateless-compatible storage, and addition of an UptimeRobot-compatible health check endpoint for availability monitoring. The system operates reliably in the cloud environment under standard free-tier constraints.

VII. ADVANTAGES

1) Unified Garage Operations: All management activities including job tracking, billing, inventory, attendance, and analytics are consolidated within a single platform, eliminating tool fragmentation.
2) AI-Assisted Diagnostics via OpenRouter: Integration with the OpenRouter API enables preliminary natural language fault diagnosis and conversational service guidance without requiring senior mechanic availability for every query.
3) Real-Time Customer Transparency: QR-based live status tracking reduces inbound inquiry call volumes and improves customer trust through self-service status visibility.
4) Automated GST Compliance: PDF invoices are generated automatically with correct CGST and SGST computation, eliminating manual calculation errors and ensuring statutory compliance.

5) Cost-Effective Open-Source Stack: Built entirely on open-source technologies (Python, Flask, SQLite, JavaScript), VSMS is self-hostable at minimal cost as an alternative to commercial platforms.
6) Layered Security: bcrypt password hashing, JWT session tokens, CSRF protection, ORM parameterization, and rate limiting together enforce multi-layer application security.
7) Modular and Scalable Architecture: The fourteen-Blueprint Flask structure and SQLAlchemy ORM enable straightforward functional extension and database backend migration.

VIII. LIMITATIONS

1) SQLite Concurrency: SQLite is optimized for single-writer workloads and may exhibit write-locking contention under high concurrency; migration to PostgreSQL is recommended for deployments serving more than twenty concurrent users.
2) External API Dependency: AI diagnostic functionality depends on the availability of the OpenRouter API. Service outages or quota exhaustion will temporarily disable the diagnostic assistant.
3) AI Response Latency: OpenRouter API round-trip latency of 1.5 to 3.2 seconds may be perceptible in time-critical service interactions; response caching or local model inference could mitigate this.
4) Single-Instance Architecture: The current deployment does not incorporate horizontal scaling or load balancing, constraining throughput to single-server capacity.
5) No Native Mobile Application: The platform is browser-based and does not provide dedicated Android or iOS applications, potentially limiting accessibility for mechanics working away from fixed workstations.
6) Absence of Automated Testing Suite: The implementation lacks a formal unit and integration test suite, increasing regression risk during future development iterations.

IX. FUTURE SCOPE

Several enhancements are identified for future development of VSMS. Migration from SQLite to PostgreSQL will enable concurrent multi-user production deployment and support replication for high availability. Integration of a locally hosted open-source language model through frameworks such as Ollama will eliminate external API dependency and reduce diagnostic response latency.

Development of native Android and iOS mobile applications using Flutter or React Native will extend platform accessibility to mechanics conducting inspections away from fixed workstations. Integration of OBD-II vehicle telematics adapters will enable real-time engine fault code ingestion, significantly improving AI diagnostic accuracy beyond natural language symptom description alone.

A customer-facing self-service portal providing service history access, online appointment booking, and digital invoice retrieval will further improve the end-customer experience. Implementation of a comprehensive automated testing suite using pytest and Selenium will improve system reliability and enable continuous integration workflows. Container-based deployment using Docker Compose will simplify environment reproducibility and horizontal scaling for larger deployments.

X. CONCLUSION

This paper presented the design, implementation, and evaluation of the Vehicle Service Management System (VSMS), a full-stack web-based platform that automates the complete vehicle service lifecycle for automotive garages. By unifying role-based access control, OpenRouter-powered AI diagnostics, QR-based customer tracking, automated GST-compliant invoicing, real-time analytics, and comprehensive operational management within a single Python Flask and SQLite application, VSMS directly addresses the critical inefficiencies of manual garage management workflows.

End-to-end functional testing confirmed reliable operation across all fourteen modules and three user roles. The system demonstrated sub-200-millisecond response times for standard operations, contextually accurate AI diagnostic responses via the OpenRouter API, and successful cloud deployment on the Render platform. The open-source technology stack ensures deployment accessibility for independent garages and academic institutions that cannot afford commercial alternatives.

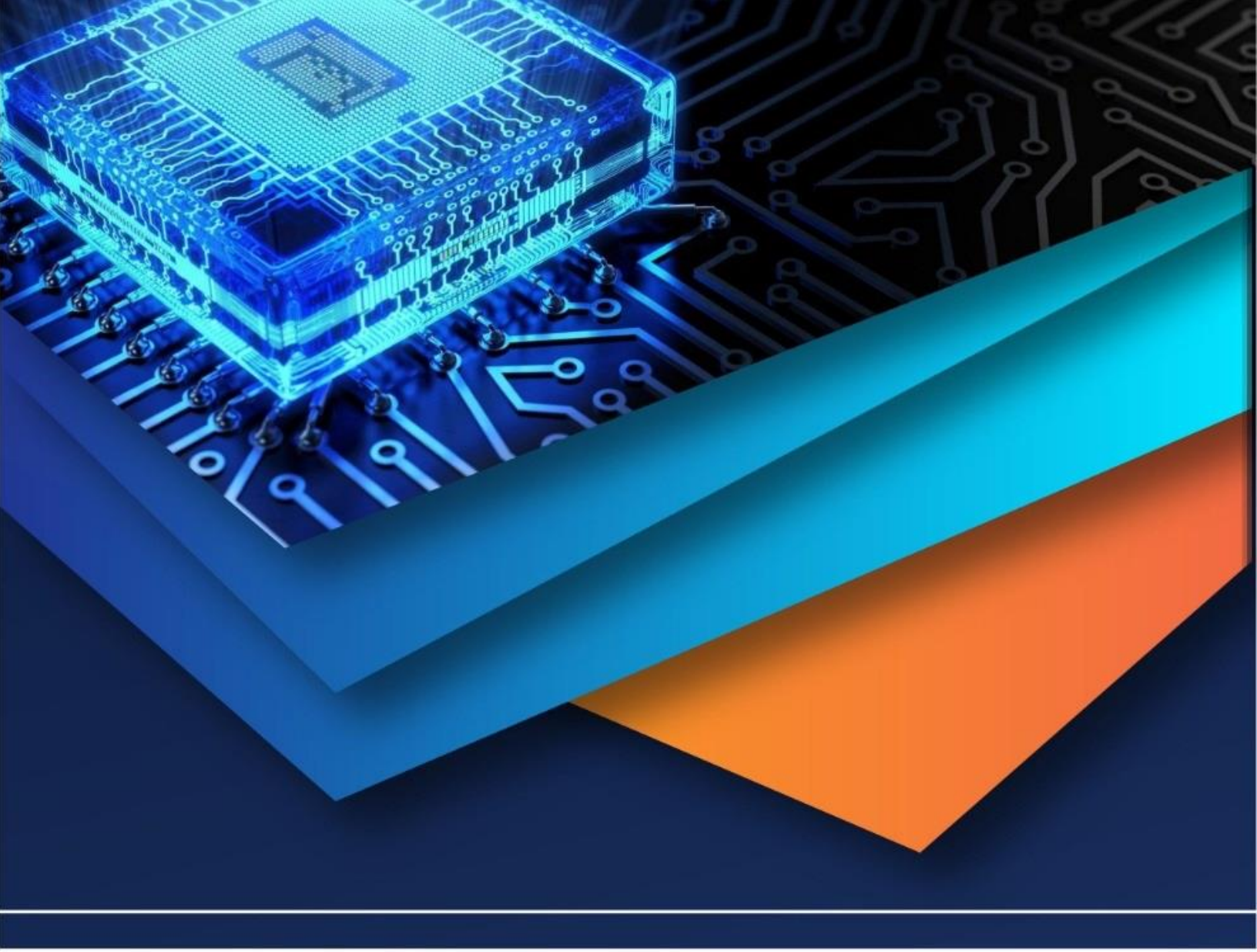
VSMS demonstrates that well-architected, AI-integrated web applications can meaningfully transform domain-specific operational management. Future work targeting PostgreSQL scalability, native mobile applications, local LLM inference, and OBD-II telematics integration will further advance the platform toward a comprehensive, production-grade garage management solution.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

REFERENCES

[1] Sharma, R., Gupta, A., and Mehta, S. (2021). Digitization of Automotive Service Workflows: Impact on Turnaround Time and Customer Satisfaction. International Journal of Automotive Engineering, 12(3), 45–58.
[2] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-Based Access Control Models. IEEE Computer, 29(2), 38–47.
[3] Kumar, P., Singh, V., and Rao, D. (2023). Large Language Models for Automotive Fault Diagnosis: An Empirical Evaluation. Journal of Intelligent Transportation Systems, 27(4), 312–325.
[4] Patel, N., Joshi, M., and Desai, R. (2022). QR Code-Based Real-Time Service Status Tracking in Service Operations. International Journal of Service Management, 33(2), 178–195.
[5] Ministry of Finance, Government of India. (2017). Goods and Services Tax: Invoicing Rules and Compliance Framework. New Delhi: Government of India Press.
[6] Elmasri, R., and Navathe, S. B. (2016). Fundamentals of Database Systems (7th ed.). Pearson Education.
[7] OWASP Foundation. (2023). Authentication Cheat Sheet — OWASP Cheat Sheet Series. Retrieved from https://cheatsheetseries.owasp.org
[8] Nielsen Norman Group. (2022). Glassmorphism in User Interface Design: Usability and Aesthetic Considerations. Retrieved from https://www.nngroup.com
[9] Downie, N. (2020). Chart.js: Interactive Data Visualization for the Web. Packt Publishing.
[10] Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.
[11] Fielding, R. T. (2000). Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Dissertation, University of California, Irvine.
[12] OpenRouter. (2024). OpenRouter API Documentation: Unified Interface for Large Language Model Access. Retrieved from https://openrouter.ai/docs

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)