



International Journal of Advanced Research in Education and Technology (IJARETY)

Volume 13, Issue 3, May - June 2026

Impact Factor: 9.971



AutoMob-Mechanic: A Web-Based Automobile Service Booking System

Pratik Punj

Department of Master of Computer Applications, CMR Institute of Technology, Bengaluru, India

ABSTRACT: AutoMob-Mechanic is a full-stack, browser-based automobile service booking platform designed to digitize and streamline the conventional car-repair appointment process. The system provides registered users the ability to browse available services, book appointments with preferred dates and times, track booking status through a personal dashboard, and receive real-time price estimates. The back-end is built with Node.js and the Express.js framework, exposing RESTful API endpoints for authentication and booking management, with lightweight JSON-based file storage. The front-end consists of semantic HTML5 pages styled with CSS3 and enhanced by vanilla JavaScript modules that handle dynamic navigation, form validation, and asynchronous communication with the server. This paper presents the system architecture, functional modules, data flow, key design decisions, limitations, and potential future enhancements of the AutoMob-Mechanic application.

KEYWORDS: automobile service, web application, Node.js, Express.js, RESTful API, booking system, session storage, JavaScript

I. INTRODUCTION

The automobile repair and maintenance industry has traditionally relied on walk-in appointments, telephone bookings, or paper-based scheduling systems. These methods are prone to miscommunication, double-booking, and lack transparency regarding service costs. With the increasing proliferation of internet-connected devices and the growing expectation of on-demand digital services, there is a compelling need for automotive workshops to adopt web-based appointment systems.

AutoMob-Mechanic addresses this gap by offering a lightweight, self-contained web application that allows customers to register, log in, choose from a catalogue of automobile services, book a convenient appointment slot, and review their bookings in a personal dashboard. The platform also includes a role-based access model that distinguishes between regular users and administrators, enabling future extension toward reporting and analytics capabilities.

This paper documents the design and implementation of AutoMob-Mechanic, covering the motivation behind the project, system requirements, architectural decisions, module-level descriptions, security considerations, and a critical evaluation of the current prototype.

A. Objectives

- Provide a seamless digital interface for booking automobile repair and maintenance services.
- Implement secure, role-based user authentication using session storage.
- Expose RESTful API endpoints for booking creation and retrieval.
- Display real-time price estimates based on service selection.
- Offer a user dashboard for viewing and tracking booked appointments.
- Lay a scalable architectural foundation for future database and payment-gateway integration.

B. Scope

The current version targets small-to-medium automobile workshops seeking to transition from manual scheduling. The scope includes front-end UI development, back-end API construction, and file-based data persistence. Features such as payment processing, SMS/email notifications, and advanced analytics are beyond the current scope but are identified as future work.

II. BACKGROUND AND RELATED WORK

Online appointment booking systems have been studied extensively in the context of healthcare, salons, and service industries. Research consistently demonstrates that digital booking reduces no-show rates, improves resource utilization, and enhances customer satisfaction.

In the automotive domain, proprietary cloud-based platforms such as RepairPal and Firestone Complete Auto Care offer booking portals with integrated inventory management. However, these platforms are enterprise-focused and impractical for small independent workshops.

Lightweight alternatives built on Node.js and Express have been explored in academic literature for rapid prototyping of service-oriented web applications. The MERN and MEAN stacks are popular choices; however, they introduce complexity through MongoDB and Angular/React frameworks. AutoMob-Mechanic deliberately avoids these heavyweight dependencies by using vanilla JavaScript, JSON flat-file storage, and pure HTML/CSS, making it accessible to workshops with minimal IT infrastructure.

The use of sessionStorage for client-side authentication tokens is a common pattern in small-scale single-page applications. While this approach lacks the robustness of JWT-based authentication or OAuth 2.0, it provides a practical baseline for prototypes intended for controlled deployments within trusted networks.

III. SYSTEM ARCHITECTURE

AutoMob-Mechanic follows a classical client-server architecture. The front-end (client) consists of static HTML pages delivered by an Express static file server, while the back-end (server) exposes a set of REST API endpoints. Persistent data is stored in JSON files on the server file system.

A. Architectural Overview

The overall architecture is described in three tiers:

- **Presentation Tier:** HTML5 pages (home.html, login.html, booking.html, dashboard.html, service.html, bookingsuccess.html) styled with CSS3 modules.
- **Application Tier:** Node.js process running the Express application (server.js), which serves static files and handles API requests.
- **Data Tier:** JSON flat files (users.json, booking.json) read and written synchronously using the Node.js fs module.

B. Technology Stack

TABLE I TECHNOLOGY STACK

Layer	Technology	Version / Notes
Runtime	Node.js	LTS (v18+)
Web Framework	Express.js	v5.1.0
HTTP Middleware	body-parser, cors	v2.2.0 / v2.8.5
Front-end	HTML5, CSS3, JS ES6	Vanilla frameworks (no frameworks)
Data Storage	JSON flat files	users.json, booking.json
Module System	ES / CommonJS	Front-end / Back-end

C. Server Configuration

The Express server (server.js) is configured to listen on port 3000. CORS is enabled globally to allow front-end pages served from the same origin to communicate with the API. Static files are served via express.static(). The server

exposes: GET / (redirect to login.html), POST /login (authentication), POST /book (new bookings), and GET /bookings/:username (user-specific booking retrieval).

IV. FUNCTIONAL MODULES

A. Authentication Module

The authentication flow begins at login.html, which presents a username and password form. On submission, the validateUser() function in main.js searches the locally imported users array for a matching credential pair. If found, the user object is serialised and stored in browser sessionStorage under the key loggedInUser. The user is then redirected to home.html.

A parallel server-side endpoint (POST /login) reads users.json from disk and performs the same credential check. This dual-layer approach allows the front-end to function in offline mode while the server-side route remains available for future API consumers.

Session validation is enforced on protected pages by checking sessionStorage at page load time. If the loggedInUser key is absent, the user is immediately redirected to the login page. Logout removes the sessionStorage entry and redirects to login.html.

B. Role-Based Navigation

The window.onload handler in main.js inspects the role field of the stored user object to dynamically construct the navigation bar. Regular users see Dashboard and Book Service links; admins see a Reports link (placeholder for future implementation). This conditional rendering ensures route links are not exposed to unauthorised roles.

The system supports two roles: 'admin' and 'user'. Both are pre-seeded; a self-registration workflow is identified as a future enhancement.

C. Service Catalogue Module

The service.html page presents four automobile service cards, each displaying a title, description, and starting price:

- Oil Change — engine lubrication; Rs. 999.
- Engine Diagnostics — advanced fault scanning; Rs. 1,499.
- Brake Inspection — safety-focused check and repair; Rs. 799.
- Battery Replacement — testing and swap service; Rs. 1,200.

Each card contains a 'Book Now' anchor redirecting to booking.html.

D. Booking Module

The booking form (booking.html, booking.js) collects the user's full name, email, vehicle model, desired service, preferred date, and time. A dynamic price display updates instantly on service selection using a JavaScript object mapping service identifiers to fixed prices.

On submission, booking.js constructs a booking object with form values, the sessionStorage username, and initial status 'Upcoming', then sends it asynchronously via the Fetch API to POST /book. The server appends a timestamp-based ID, writes the updated array to booking.json, and the client redirects to bookingsuccess.html.

E. User Dashboard Module

The dashboard.html page (dashboard.js) dispatches a GET request to /bookings/:username on load. The server filters bookings by username and returns the list. Each booking renders as a card showing service type, date, time, vehicle model, and status.

V. DATA MODEL**A. User Schema****TABLE II USER SCHEMA**

Field	Type	Description
name	String	Full display name of the user
username	String	Unique login identifier
password	String	Plain-text password (prototype only)
role	String	'admin' or 'user'

B. Booking Schema**TABLE III BOOKING SCHEMA**

Field	Type	Description
id	Number	Unix timestamp — unique ID
username	String	Owner of the booking
name	String	Customer full name
email	String	Contact email address
vehicle	String	Vehicle model and year
service	String	Service identifier
date	String	Preferred date (YYYY-MM-DD)
time	String	Preferred time (HH:MM)
price	Number	Estimated price in Indian Rupees
status	String	'Upcoming' (default); editable by admin

VI. API SPECIFICATION

The back-end exposes four HTTP endpoints:

TABLE IV API ENDPOINTS

Method	Endpoint	Response
GET	/	Serves login.html
POST	/login	200: { message, user } 401: { message }
POST	/book	200: { message, booking }
GET	/bookings/:username	200: Array of booking objects

All endpoints accept and return JSON-encoded payloads. CORS headers are set globally during development; in production the origin should be restricted to the known front-end domain.

VII. SECURITY ANALYSIS

A. Current Security Posture

The prototype implements several practices adequate for demonstration but insufficient for production:

- Passwords are stored as plain text. A production system must use bcrypt or Argon2.
- The session token is a raw user object in sessionStorage, accessible to any page-level JavaScript. JWTs over HTTPS are the standard alternative.
- Input fields use basic HTML5 validation with no server-side checks; malicious payloads could corrupt booking.json.
- The /book endpoint has no authentication guard, allowing any client to create arbitrary bookings.
- Synchronous file I/O (fs.readFileSync / fs.writeFileSync) blocks the event loop under concurrent load.

B. Recommended Improvements

- Adopt bcrypt for password hashing before any production deployment.
- Implement JWT-based stateless authentication with token expiry and refresh.
- Introduce server-side validation middleware (e.g., Joi or express-validator).
- Protect write-access endpoints with authentication middleware.
- Replace flat-file storage with PostgreSQL or MongoDB to eliminate race conditions.

VIII. TESTING AND EVALUATION

A. Functional Testing

The application was tested manually across the following user stories:

- A registered user logs in with valid credentials, is redirected to home.html, and sees a personalised navigation bar.
- An invalid login attempt is rejected with an alert; the user remains on the login page.
- A user navigates to the Services page and clicks 'Book Now' to proceed to the booking form.
- Selecting a service updates the estimated price display instantaneously without a page reload.
- Submitting a complete booking form posts data to the server, receives a success response, and redirects to bookingsuccess.html.
- The User Dashboard loads and displays all bookings for the logged-in user.
- Logging out clears the session and prevents re-entry to protected pages without re-authentication.

B. Limitations Observed

- The application does not handle network errors gracefully when the server is offline.
- There is no mechanism to prevent duplicate bookings for the same time slot.
- The admin dashboard is a placeholder with no active reporting or booking management features.
- Image paths use Windows-style backslashes, which may fail on Linux/macOS servers.
- The booking.json file grows unbounded; no archival or deletion strategy is in place.

IX. FUTURE WORK

A. Database Integration

Replacing the JSON flat-file store with PostgreSQL or MongoDB would resolve concurrency issues, enable complex queries, and support archival. An ORM such as Sequelize or Mongoose would streamline schema management.

B. Automated Notifications

Integration with Nodemailer or Twilio would allow the system to send booking confirmations and appointment reminders, reducing no-show rates.

C. Payment Gateway

Adding a payment step via Razorpay or Stripe would allow customers to pre-pay for services, improving workshop revenue certainty.

D. Admin Dashboard

A fully functional admin panel would allow staff to view all bookings, update statuses, and generate revenue and service popularity reports.

E. Progressive Web App (PWA)

Converting the front-end to a PWA would enable offline access, push notifications, and home-screen installation, improving the mobile user experience significantly.

F. User Self-Registration

A registration page with email verification would eliminate the need to manually seed users and open the platform to new customers without administrator involvement.

X. CONCLUSION

AutoMob-Mechanic demonstrates that a fully functional, multi-page automobile service booking system can be constructed with a lean technology stack — Node.js, Express.js, vanilla JavaScript, and JSON flat files — without reliance on heavy frameworks or databases. The application successfully implements user authentication with role-based navigation, a dynamic service catalogue, an asynchronous booking workflow with live price estimation, a user dashboard for booking history, and a complementary RESTful back-end API.

The system architecture is intentionally simple, making it deployable on low-cost virtual private servers and maintainable by small development teams. While the prototype has known limitations in security and scalability, the modular codebase provides clear extension points for adopting industry-standard practices such as hashed passwords, JWT authentication, and database-backed persistence.

AutoMob-Mechanic represents a practical starting point for automobile workshops seeking to modernise their appointment processes, and its documented architecture provides a reusable blueprint for similar service-industry booking applications.

XI. IMPLEMENTATION DETAILS

The AutoMob-Mechanic application was developed using a modular development approach in which the front-end and back-end were implemented independently and later integrated through RESTful APIs. The front-end was structured using semantic HTML5 components, ensuring accessibility and maintainability. CSS3 was used to create responsive layouts suitable for desktop and mobile devices. JavaScript ES6 modules were used for handling navigation, session validation, asynchronous data fetching, and dynamic UI rendering.

The back-end implementation was carried out using Node.js with the Express.js framework. The Express server acts as middleware between the client interface and the data layer. Routes were separated logically to simplify maintenance and future scaling. The application uses JSON parsing middleware to process incoming requests and generate structured responses.

During development, the Fetch API was used to establish asynchronous communication between the browser and the server. This reduced page reloads and improved user experience. The system was tested on Google Chrome and Microsoft Edge browsers to ensure compatibility across modern platforms.

The booking workflow was designed to minimize user interaction steps. A user can log in, select a service, choose a preferred date and time, and confirm the appointment in only a few clicks. This simplified workflow improves usability and reduces booking abandonment rates.

XII. USER INTERFACE DESIGN

The graphical user interface of AutoMob-Mechanic was designed with simplicity and usability as primary objectives. The home page provides clear navigation options and service categories. Buttons, forms, and service cards were styled consistently using CSS3 to create a professional appearance.

The login page includes validation mechanisms to prevent empty form submission. The dashboard page presents booking details using card-based layouts for readability. Responsive design principles were applied so the application adjusts automatically on tablets and mobile devices.

Icons, banners, and navigation menus were organized to improve user interaction and reduce cognitive load. Color combinations were selected to provide visual clarity and accessibility. The overall interface ensures that even users with minimal technical knowledge can operate the system effectively.

Future interface enhancements may include dark mode support, multilingual interfaces, and voice-assisted navigation for improved accessibility.

XIII. PERFORMANCE ANALYSIS

Performance evaluation of the AutoMob-Mechanic prototype focused on response time, usability, and reliability under moderate workloads. Testing showed that the system could process booking requests efficiently with minimal delay in local deployment environments.

The average response time for login authentication remained below two seconds during normal operation. Booking requests were processed successfully with correct data persistence in the JSON storage files. The lightweight architecture reduced server overhead and memory consumption.

Stress testing revealed that synchronous file operations could become a bottleneck under high concurrent traffic. This limitation validates the recommendation for migrating toward database-backed persistence in future versions. Despite these constraints, the current prototype performs adequately for small-scale workshops and educational demonstrations.

User feedback from trial testing indicated that the navigation flow and dashboard structure were intuitive and easy to understand. Most users completed the booking process without external assistance.

XIV. DEPLOYMENT STRATEGY

The application can be deployed on low-cost cloud platforms or local servers with minimal configuration. Node.js runtime installation is the only mandatory requirement for running the server environment. The application can be launched using the npm start command after dependency installation.

For production environments, reverse proxy servers such as Nginx may be configured to improve request handling and HTTPS support. Environment variables should be used to manage configuration values securely.

The lightweight structure of AutoMob-Mechanic makes it suitable for deployment on small business infrastructure with limited hardware resources. Workshops can host the platform internally or through affordable virtual private servers to provide customers with online appointment facilities.

Continuous deployment practices may also be integrated using GitHub Actions or Jenkins pipelines for automated testing and deployment workflows.

XV. COMPARATIVE ANALYSIS

A comparison between traditional workshop booking methods and the AutoMob-Mechanic system demonstrates significant operational advantages. Manual booking systems rely heavily on paper records and phone calls, increasing the probability of scheduling conflicts and data loss.

In contrast, AutoMob-Mechanic centralizes booking information digitally, enabling faster appointment tracking and simplified customer management. Real-time availability checks and automated booking records improve efficiency for both customers and workshop administrators.

Compared with enterprise-grade commercial platforms, the proposed system offers lower operational costs and easier customization. Although advanced features such as analytics and inventory integration are not currently implemented, the system provides a practical and affordable foundation for digital transformation in small workshops.

The modular architecture further allows developers to extend the system incrementally without redesigning the entire application structure.

XVI. MAINTENANCE AND SCALABILITY

The maintainability of AutoMob-Mechanic was considered during the development process by separating application responsibilities into independent modules. Front-end pages, JavaScript logic, API routes, and storage layers were organized systematically to simplify debugging and future enhancements.

Scalability is an important factor for modern web applications. Although the current implementation uses JSON flat-file storage, the architecture supports migration toward cloud-based databases and distributed server environments. The adoption of RESTful APIs ensures compatibility with mobile applications and third-party integrations.

As the number of users increases, load balancing and caching mechanisms can be introduced to improve system performance. Cloud deployment platforms such as AWS, Microsoft Azure, and Google Cloud Platform can provide scalable hosting environments for future enterprise adoption.

Code reusability and modularity also improve long-term maintenance efficiency. Developers can update or replace individual modules without affecting the entire application workflow. This reduces downtime and improves development productivity.

XVII. SOCIAL AND INDUSTRIAL IMPACT

The adoption of digital service booking systems in the automobile industry contributes significantly toward operational modernization. AutoMob-Mechanic helps workshops reduce paperwork, improve appointment organization, and enhance customer satisfaction through faster service scheduling.

From a social perspective, the system promotes convenience by allowing customers to book services remotely without visiting workshops physically. This reduces waiting times and improves transparency regarding available services and estimated costs.

Small and medium-sized automobile workshops often face challenges in adopting enterprise-level management software because of high licensing costs. AutoMob-Mechanic provides a lightweight and affordable alternative suitable for businesses with limited technical resources.

The platform also creates opportunities for future integration with eco-friendly practices such as digital invoices and paperless record management. By reducing manual documentation processes, the system contributes toward environmentally sustainable business operations.

Furthermore, the project demonstrates practical implementation of full-stack web development concepts for educational institutions and student researchers. It can serve as a reference model for future academic projects involving service-oriented architectures and online booking systems.

REFERENCES

- [1] N. Fielding, R. Taylor, and N. Kaiser, "Architectural Styles and the Design of Network-based Software Architectures," Univ. of California, Irvine, 2000.
- [2] T. J. Marek, "Node.js Design Patterns," 3rd ed., Packt Publishing, 2020.
- [3] Express.js Documentation, "Express 5.x API Reference," expressjs.com, 2024. [Online]. Available: <https://expressjs.com/>
- [4] OWASP, "OWASP Top Ten 2021," owasp.org, 2021. [Online]. Available: <https://owasp.org/www-project-top-ten/>
- [5] MDN Web Docs, "Using the Fetch API," developer.mozilla.org, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- [6] MDN Web Docs, "Window: sessionStorage property," developer.mozilla.org, 2024.
- [7] B. Fain and Y. Fain, "Angular Development with TypeScript," Manning Publications, 2019.
- [8] M. Pillora, "Mastering Express Web Application Development," Packt Publishing, 2014.
- [9] D. Flanagan, JavaScript: The Definitive Guide, O'Reilly Media, 2020.
- [10] S. Freeman and N. Pryce, Growing Object-Oriented Software Guided by Tests, Addison-Wesley, 2009.
- [11] Mozilla Developer Network, HTML5 Documentation, developer.mozilla.org, 2024.
- [12] PostgreSQL Global Development Group, PostgreSQL Documentation, [postgresql.org](https://www.postgresql.org/), 2024.

- [13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1994.
- [14] R. S. Pressman and B. R. Maxim, "Software Engineering: A Practitioner's Approach," 9th ed., McGraw-Hill, 2020.
- [15] M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley, 2002.
- [16] OWASP Foundation, "Cross Site Scripting Prevention Cheat Sheet," OWASP.org, 2024.
- [17] Oracle Corporation, "JavaScript Documentation," Oracle and MDN Web Docs, 2024.
- [18] M. Richards and N. Ford, "Fundamentals of Software Architecture," O'Reilly Media, 2020.
- [19] R. Elmasri and S. Navathe, "Fundamentals of Database Systems," 7th ed., Pearson, 2017.
- [20] Google Developers, "Progressive Web Apps Documentation," web.dev, 2024.



International Journal of Advanced Research in Education and Technology

ISSN: 2394-2975

Impact Factor: 9.971