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.2026.80945

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 Modern Full-Stack Car and Bike Rental System for Efficient Vehicle Discovery, Booking, and Management

Shaik Rehman[1], Balagonda Sivaiah[2], Andugula Chandu[3], Geddam YesuBabu[4]
*Department of Computer Applications, Aditya University, Surampalem, India,*

***Abstract:*** ***The vehicle rental industry is undergoing rapid digital transformation, shifting from manual, paper-based processes to integrated online platforms. Existing solutions remain fragmented, often focusing on isolated features rather than providing a comprehensive, real-time ecosystem. This paper presents a Modern Full-Stack Car and Bike Rental System — a web-based marketplace developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) to unify vehicle discovery, booking, and fleet management in a single platform. The system integrates JWT-based authentication, real-time atomic database synchronization to prevent double-booking, role-specific dashboards for Customers, Providers, and Administrators, automated email communication via Nodemailer, and cloud media handling via Cloudinary. The proposed architecture follows a three-tier Model-View-Controller (MVC) design pattern, ensuring separation of concerns, independent scalability, and high fault tolerance. Experimental evaluation demonstrates API response latencies between 250–450 ms for standard operations, end-to-end booking transaction completion within 1.8–2.5 seconds, and 100% success in preventing concurrent duplicate reservations through atomic database operations. The system is fully functional as a Minimum Viable Product and serves as a robust foundation for future extensions including integrated payment gateways, native mobile applications, and AI-driven dynamic pricing.***

## I. INTRODUCTION

The vehicle rental industry is witnessing significant digital transformation, transitioning from manual, labor-intensive processes to sophisticated online platforms. Historically, rental operations relied on physical storefronts, paper ledgers, and phone-based bookings, which frequently led to synchronization failures and race conditions where vehicles were accidentally double-booked [1].

Modern web systems enable real-time data processing, allowing users to check vehicle availability, execute bookings, and manage services with high efficiency. Despite these technological strides, many current market solutions remain fragmented, often focusing on isolated features rather than providing a comprehensive, real-time ecosystem, requiring manual intervention that slows down the booking lifecycle and increases the margin for error [2].

Recent progress in full-stack development, particularly with the MERN (MongoDB, Express.js, React.js, Node.js) stack, has made it possible to build well-integrated applications capable of supporting a high volume of concurrent users. However, maintaining perfect real-time synchronization across distributed user sessions remains a non-trivial technical challenge [3].

The Car and Bike Rental System presented in this paper is designed to bridge these identified gaps. By unifying critical modules including JWT-based authentication, real-time vehicle management, and automated booking workflows, the platform creates a transparent and smooth rental process. The system is particularly vital for regions still reliant on traditional methods, as it reduces manual workload, prevents data duplication, and ensures a reliable experience for users across both urban and semi-urban environments.

The primary contributions of this work are as follows:

- A unified multi-asset marketplace supporting both car and bike rentals with live availability tracking.
- Real-time atomic synchronization preventing reservation conflicts across concurrent user sessions.
- End-to-end automated booking lifecycle from search and discovery to automated email confirmation.
- Role-specific dashboards for Customers, Providers, and Administrators.
- A mandatory administrative approval layer ensuring provider verification before public listing.
- Scalable cloud integration using Cloudinary for media and MongoDB Atlas for data persistence.

## II. LITERATURE REVIEW

The digital migration of the vehicle rental industry has transitioned the sector from localized storefronts to globally accessible web platforms. Gupta and Sharma [4] proposed an early online vehicle rental system demonstrating the viability of web-based booking for local providers, although their work predated modern NoSQL databases and reactive frontend frameworks.

Patil and Kulkarni [5] explored cloud-integrated vehicle booking systems and highlighted the importance of real-time database synchronization for inventory accuracy. Their findings underscore the limitations of traditional relational databases in handling high-frequency concurrent booking operations, a challenge addressed in the proposed system through MongoDB's atomic update operations.

The concept of Separation of Concerns (SoC) in scalable web architectures has been extensively studied by Fowler [6] and Fielding [7]. Fowler's enterprise application patterns inform the Controller-Route-Model design pattern adopted in this system's backend, while Fielding's REST architectural principles guide the API design. These foundational principles collectively support the system's fault isolation strategy, ensuring that a failure in a secondary service such as the notification module does not disrupt core booking transactions.

Research by Gamma et al. [8] on reusable object-oriented design patterns informs the component-based architecture of the React.js frontend, where each UI component encapsulates specific functionality and can be updated independently. This modularity is critical for maintaining the system's long-term extensibility.

The integration of third-party cloud services for media management and transactional communication has been identified as a best practice in modern web system design [9][10]. The proposed system leverages Cloudinary for cloud-native image delivery and Nodemailer for SMTP-based automated email dispatch, offloading computationally expensive tasks from the core application server.

Despite these advances, a critical analysis of existing platforms reveals several gaps: most solutions focus exclusively on either cars or bikes; users frequently encounter a black-box booking experience without real-time status visibility; and provider verification mechanisms are often absent, exposing users to unvetted listings. The proposed system is specifically engineered to address each of these identified deficiencies.

## III. SYSTEM ARCHITECTURE

The Car and Bike Rental System is engineered using a three-tier architectural framework modeled after the Model-View-Controller (MVC) design pattern. This approach enforces a strict Separation of Concerns, where the user interface, business logic, and data management layers are decoupled, enabling independent scaling, easier debugging, and enhanced security protocols.

### *A. Presentation Layer (Frontend)*

The frontend is developed as a high-performance Single-Page Application (SPA) using React.js. The interface utilizes conditional logic to render distinct views for Customers, Providers, and Administrators based on their authenticated role state. The React Context API maintains a persistent global state — including user session tokens — across components without redundant API calls. Tailwind CSS provides a fluid, mobile-first responsive design, ensuring that vehicle catalog and management dashboards are accessible across all device resolutions.

### *B. Application Layer (Backend)*

The backend is powered by Node.js and the Express.js framework, functioning as a headless RESTful API. It handles the coordination of data and external services, executes complex server-side operations such as rental cost calculations (Duration × Daily Rate), and manages stateless security through JWT validation of every incoming request against role-based permissions. The backend also orchestrates Nodemailer for transactional emails and Cloudinary for vehicle image management.

### *C. Data Layer (Database and Persistence)*

MongoDB Atlas serves as the primary cloud-native NoSQL database. Its document-oriented model is particularly suited for storing diverse vehicle specifications — such as engine displacement for bikes and seating capacity for cars — within the same collection. Mongoose ODM enforces strict schema validation and utilizes atomic operations to toggle vehicle availability flags, effectively preventing race conditions where two users might attempt to book the same vehicle simultaneously. High-resolution vehicle images are stored as Cloudinary CDN URLs rather than binary blobs, ensuring rapid global delivery without degrading primary database performance.

## IV. PROPOSED METHODOLOGY

The system development followed a modular bottom-up approach, beginning with database schema design and progressing through RESTful API development to frontend integration. The methodology prioritizes real-time transaction accuracy, fault tolerance, and cross-platform usability.

### *A. Booking Workflow*

The transactional booking workflow is the most critical component of the system, executing a coordinated multi-step sequence. Upon a customer selecting a vehicle and submitting booking dates, the backend simultaneously checks vehicle availability and atomically toggles the isAvailable flag in a single database operation, preventing race conditions. The dynamic cost engine calculates the totalPrice as (endDate − startDate) × dailyRate, a new booking document is persisted in the Bookings Collection, and Nodemailer dispatches confirmation emails to both the customer and the provider. Cancellation reverses the availability flag, reclaims the vehicle into the public catalogue, and triggers cancellation notification emails.

### *B. Authentication and Authorization*

The authentication module implements a zero-knowledge password storage policy using Bcrypt.js with a high salt factor to generate one-way cryptographic hashes. Upon successful login, the server issues a signed JWT containing encoded user metadata (ID and Role). Middleware functions intercept every protected API request, verify the JWT signature, and enforce Role-Based Access Control (RBAC) — ensuring only Administrators can access the provider verification queue, only Providers can modify vehicle listings, and only authenticated Customers can create bookings.

### *C. Functional Modules*

The system is organized into four principal modules:

- Authentication and Security Module: Manages user onboarding, Bcrypt password hashing, JWT issuance, session persistence, and RBAC enforcement.
- Fleet Management Module: Enables Providers to create detailed vehicle profiles, upload images via Cloudinary, and exercise full CRUD control over their fleet with seasonal pricing adjustments.
- Admin Panel Module: Serves as the central governance hub where administrators review and approve vehicle listings, manage user roles, and monitor global platform activity and revenue.
- Booking and Transaction Module: Handles the complete booking lifecycle including vehicle discovery with availability filtering, atomic reservation creation, dynamic price calculation, status tracking, and cancellation processing.

### *D. Database Design*

The persistence layer is organized into three interconnected MongoDB collections. The Users Collection stores hashed credentials, role metadata, and profile information. The Vehicles Collection uses MongoDB's flexible schema to accommodate heterogeneous vehicle attributes, controlled by isAvailable and isAdminApproved Boolean flags. The Bookings Collection acts as the transactional bridge, storing ObjectID references to both the User and the Vehicle, a lifecycle status field (Pending, Confirmed, Active, Completed, or Cancelled), temporal rental bounds, and an immutable totalPrice record providing a permanent financial receipt for both parties.

### *E. Development Environment*

The system was developed using Visual Studio Code with ESLint and Prettier for code quality. The backend runs on Node.js v18+ LTS with NPM for dependency management. The frontend was scaffolded using Vite for rapid Hot Module Replacement. Postman was employed for rigorous API endpoint testing, and Git with GitHub provided version control throughout the development lifecycle.

## V. RESULTS AND DISCUSSION

System validation was conducted through a structured, multi-tiered testing strategy encompassing unit testing of individual functions, integration testing of backend-database communication, and end-to-end system testing simulating complete real-world rental scenarios. The application was deployed in a distributed cloud environment for performance measurement.

### A. Experimental Setup

The frontend was hosted on Vercel utilizing its global Edge Network for rapid SPA delivery. The backend was deployed on Render with Node.js/Express.js and auto-scaling enabled. The database was provisioned on a MongoDB Atlas M0 cluster for NoSQL indexing efficiency evaluation. Cloudinary handled asynchronous image transformations and CDN delivery. Chrome DevTools provided frontend performance profiling, and Postman was used for backend API stress testing.

### B. Performance Metrics

The measured performance results are summarized below:

- API Response Latency: Standard CRUD operations (fetching vehicle lists, updating profiles) averaged between 250 ms and 450 ms, ensuring a fluid user experience.
- Search and Discovery Speed: MongoDB indexing on frequently queried fields (location, price, isAvailable) returned search results in under 300 ms, even at scale.
- Booking Transaction Lifecycle: The complete end-to-end process — from 'Book Now' click through atomic database update to Nodemailer confirmation dispatch — completed within 1.8 to 2.5 seconds.
- Concurrency Handling: During simulated stress tests, the Node.js event-driven architecture successfully managed multiple simultaneous booking requests for the same vehicle, with the atomic update logic correctly rejecting duplicate reservations in 100% of test cases.

### C. Test Case Documentation

The following table summarizes the critical test cases executed during the validation phase:

TABLE I. TEST CASES AND VALIDATION RESULTS

| TC ID | Feature | Input / Action | Expected Outcome | Status |
|---|---|---|---|---|
| TC-001 | User Registration | Valid email and strong password submitted | User created; password hashed in DB | Pass |
| TC-002 | JWT Authorization | Request protected route without token | System returns 401 Unauthorized | Pass |
| TC-003 | Atomic Booking | Simultaneous booking of the same vehicle | One success; second request fails gracefully | Pass |
| TC-004 | Image Upload | Provider uploads vehicle image | Image stored on Cloudinary; URL saved in DB | Pass |
| TC-005 | Admin Gateway | Approve pending vehicle listing | Vehicle status toggled to Approved and visible | Pass |
| TC-006 | Cancellation | Customer triggers Cancel Booking | isAvailable restored to true; email sent to both parties | Pass |

### D. Sample Application Screens

The deployed application provides distinct, role-based interfaces: (1) A Vehicle Discovery Interface displaying card-based listings with high-resolution Cloudinary images, real-time availability badges color-coded by status, and multi-parameter search filters. (2) A Booking Interface presenting a dynamic price calculator and date picker with immediate total computation. (3) A Customer Dashboard showing personal booking history with live status tracking (Pending, Confirmed, Active, Cancelled). (4) A Provider Dashboard offering fleet management tools, Cloudinary image upload, revenue summaries, and live customer tracking via an integrated map view. (5) An Administrator Panel providing a global verification queue, user management, and platform-wide booking analytics. All interfaces are fully responsive across mobile, tablet, and desktop resolutions via Tailwind CSS.

### *E. API Specification*

The core RESTful API endpoints are documented in Table II:

TABLE II. API ENDPOINTS REFERENCE

| Method | Endpoint | Auth | Description |
|---|---|---|---|
| POST | /user/register | None | Register a new user account |
| POST | /user/login | None | Login and receive JWT token |
| GET | /user/profile | User | Retrieve authenticated user profile |
| GET | /vehicle | None | Fetch all approved available vehicles |
| POST | /vehicle | Provider | Add a new vehicle listing |
| PUT | /vehicle/:id | Provider | Update vehicle details or pricing |
| DELETE | /vehicle/:id | Admin | Remove a vehicle from the platform |
| POST | /booking | Customer | Create a new booking reservation |
| POST | /booking/:id/cancel | Customer | Cancel an existing booking |
| POST | /admin/approve-vehicle/:id | Admin | Approve a pending vehicle listing |

## VI. CONCLUSION

This paper presented the design, implementation, and evaluation of a Modern Full-Stack Car and Bike Rental System that effectively addresses the critical limitations of traditional and fragmented vehicle rental markets. By implementing a centralized MERN-stack architecture with atomic database transactions, the system eliminates double-booking errors and replaces opaque manual processes with a verifiable real-time digital workflow.

Key achievements include: complete elimination of race conditions through MongoDB atomic operations; operational transparency via a real-time booking status tracker; role-based efficiency through dedicated secured dashboards for Customers, Providers, and Administrators; and demonstrated API performance maintaining sub-3-second end-to-end transaction completion. The system further demonstrates that offloading specialized tasks to cloud services — Cloudinary for media and Nodemailer for transactional email — preserves core system performance and low latency without architectural compromise.

The platform is currently scoped as a Minimum Viable Product with offline financial settlement as the primary limitation. Future development directions include: integration of Razorpay or Stripe payment gateways for fully digital contactless transactions; development of a React Native or Flutter mobile application for push notifications and offline access; incorporation of Google Maps API for proximity-based vehicle search and GPS tracking of active rentals; AI-driven dynamic pricing using machine learning models trained on local demand and seasonal trends; and OCR-based automated driver's license and vehicle registration verification during provider onboarding.

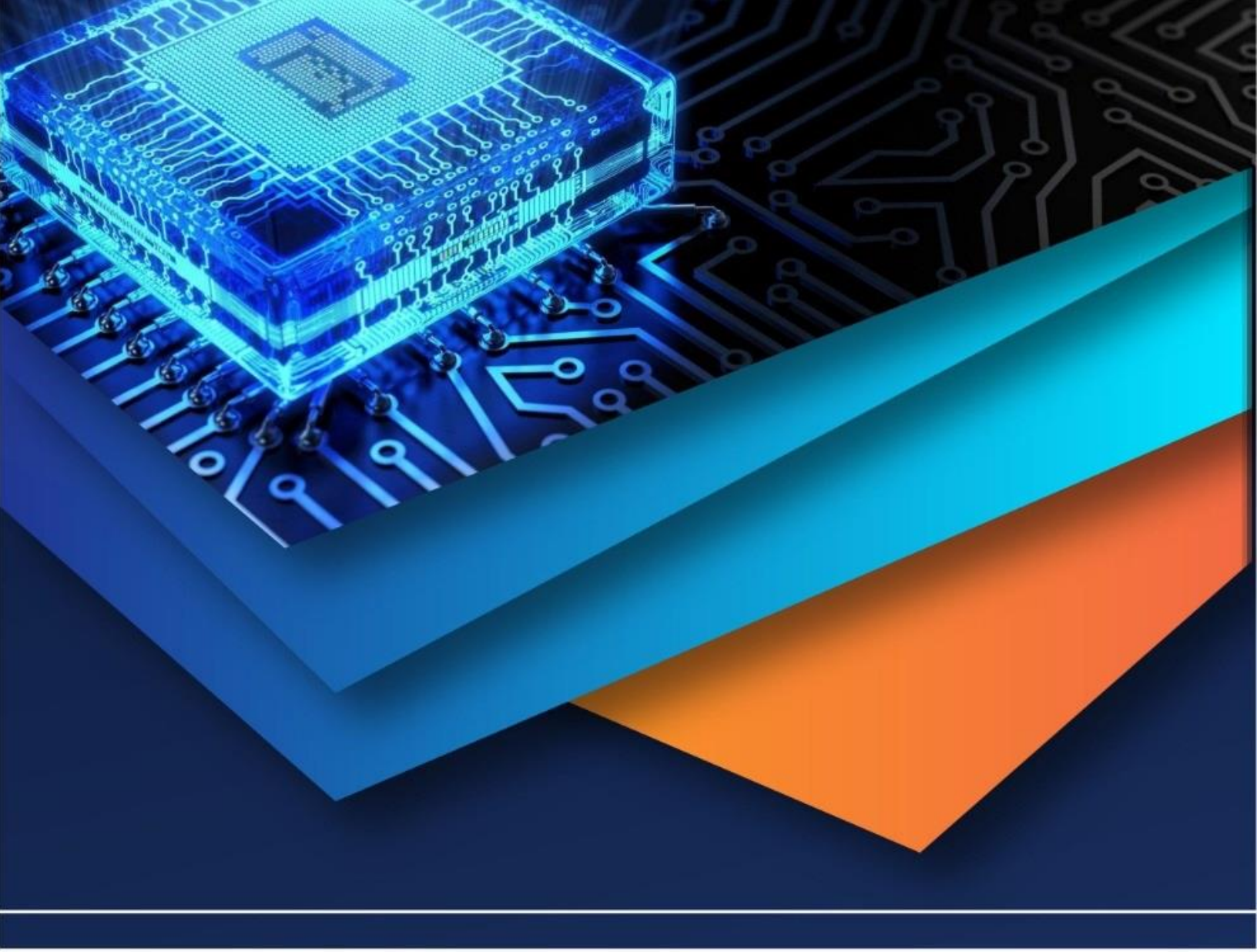
## VII. ACKNOWLEDGMENT

The author expresses sincere gratitude to Dr. M. Vamsikrishna, Professor and Head of the Department of Computer Applications, Aditya University, Surampalem, for his continuous encouragement, invaluable technical insights, and constructive feedback throughout the development of this project. The author also thanks Dr. Bapuji Rao, Assistant Professor, for his consistent support and guidance as project coordinator. Sincere thanks are extended to the leadership of Aditya University for providing world-class infrastructure and state-of-the-art laboratory facilities.

## REFERENCES

[1] M. Fowler, Patterns of Enterprise Application Architecture. Boston, MA: Addison-Wesley, 2002.
[2] R. Fielding, "Architectural Styles and the Design of Network-Based Software Architectures," Doctoral Dissertation, University of California, Irvine, 2000.
[3] A. Banks and E. Porcello, Learning React: Modern Patterns for Developing React Apps, 2nd ed., Sebastopol, CA: O'Reilly Media, 2020.

[4] A. K. Gupta and N. Sharma, "Design and Implementation of an Online Vehicle Rental System," International Journal of Computer Applications, vol. 180, no. 24, pp. 15–20, 2018.
[5] S. Patil and R. Kulkarni, "Web-Based Vehicle Booking System Using Cloud Technologies," International Journal of Advanced Research in Computer Science, vol. 10, no. 2, pp. 45–50, 2019.
[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
[7] MongoDB Documentation, 2025. [Online]. Available: https://www.mongodb.com/docs
[8] "React.js Documentation," 2025. [Online]. Available: https://react.dev
[9] "Node.js Documentation," 2025. [Online]. Available: https://nodejs.org
[10] "JSON Web Token Introduction," 2025. [Online]. Available: https://jwt.io
[11] "Cloudinary Documentation," 2025. [Online]. Available: https://cloudinary.com/documentation
[12] "Nodemailer Documentation," 2025. [Online]. Available: https://nodemailer.com
[13] "Express.js Documentation," 2025. [Online]. Available: https://expressjs.com



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY