



International Journal of Science, Architecture, Technology and Environment

Car Rental System Using the MongoDB-Express-Node.js Stack: Design and Implementation of a Web-Based Peer-to-Peer Vehicle Rental Platform

Dr. Sumlatha Bandari^{1*}, Prasad deo², Prathamesh Amle³, Pratik Ghormare⁴, Avinash Mude⁵, Yash Hatwar⁶, Vedant Wankhede⁷

¹Assistant Professor, Department of Computer Science & Engineering, G H Rasoni University, Amravati, Maharashtra, India.

^{2,3,4,5,6,7}Students, Department of Computer Science & Engineering, G H Rasoni University, Amravati, Maharashtra, India.

*Corresponding author

DOI: <https://doi.org/10.63680/ijate102645.46>

Abstract

Many local vehicle rentals in India are still arranged informally, which gives renters little visibility into availability, pricing, and owner details. This system presents Car Rental Services, a web-based peer-to-peer platform on which owners list vehicles and renters browse and book them by category and city. The backend is built with Node.js, Express.js, and MongoDB through the Mongoose library, and exposes a REST API for authentication, vehicle listings, and bookings. A browser client developed with HTML, CSS, and JavaScript provides phone-number login using a one-time password (OTP), browsing across Economy, Sedan, SUV, and Luxury categories, map-based pick-up selection using Leaflet and OpenStreetMap, and a personal dashboard for bookings and listings. The platform was populated with a seed dataset of 42 distinct listings across Nagpur, Pune, and Mumbai, whose mean daily rates range from Rs. 1,738 for Economy to Rs. 7,175 for Luxury vehicles. The implementation replaces browser-only local Storage persistence with a shared cloud database, making data available across devices. The paper also documents current limitations, including demo-mode OTP delivery and the absence of availability checking, payment, and token-based security, which define the roadmap toward production deployment.

Keywords: Car Rental System, MongoDB, Express.js, Node.js, REST API, OTP Authentication

1. INTRODUCTION

The digitalization of transportation services has changed how people access vehicles, with ride-hailing, car-sharing, and self-drive rental platforms offering alternatives to vehicle ownership [CITE: peer-reviewed or industry source on growth of car-sharing and self-drive rental platforms]. In India, short-term rentals are used for tourism, business travel, weddings, and intercity trips, but a considerable share of local rentals is still arranged through personal contacts and phone calls [CITE: market report or study describing the structure of

India's car rental sector]. Such arrangements make it hard for renters to compare vehicles and rates, and hard for individual owners to reach renters beyond their immediate network.

This study presents the design and implementation of Car Rental Services, a web-based peer-to-peer rental platform. Owners can list a vehicle with its model, category, city, daily rate, description, and photograph. Renters can browse listings by category, select a vehicle, choose a pick-up location on an interactive map, enter trip dates and driver details, and confirm a booking. Four vehicle categories are supported: Economy (hatchback), Sedan, SUV, and Luxury. The demonstration data covers three cities: Nagpur, Pune, and Mumbai.

The backend uses Node.js and Express.js to expose a REST API, with MongoDB as the database accessed through Mongoose [10][13]. Users sign in with a mobile number and a one-time password, and the server creates a user record on first login. The client is built with HTML, CSS, and JavaScript, and uses Leaflet with OpenStreetMap for location selection [7], [8]. An earlier version of the system kept all data in the browser's local Storage, so bookings and listings were visible only on one device; the present version moves persistence to a MongoDB Atlas database shared by all clients. The paper describes the architecture, data model, and API of the system, Analyze the seed dataset it ships with, and discusses limitations and future work.

2. LITERATURE REVIEW

Research on online vehicle rental and car-sharing has examined reservation systems, fleet utilization, and user adoption [CITE: two or three peer-reviewed studies on online car rental or car-sharing reservation systems, with authors and years]. Studies of peer-to-peer sharing platforms stress that trust between strangers, supported by identity verification, ratings, and listing review, is a precondition for adoption [CITE: peer-reviewed study on trust in peer-to-peer or sharing-economy platforms].

On the technology side, JavaScript full-stack frameworks such as MERN and MEAN, built on Node.js and MongoDB, are widely used for data-driven web applications because one language serves both client and server and document databases tolerate evolving schemas [CITE: comparative or survey study of MERN/MEAN stacks]. Consumer applications in India commonly authenticate users with mobile-number OTPs. Security guidance recommends that such secrets be delivered through a separate channel and be short-lived, and not be disclosed to the requesting client in the same response [5], [6] [VERIFY: confirm the exact recommendation in the cited NIST and OWASP editions]. For location selection, open-source mapping libraries such as Leaflet, together with OpenStreetMap data and the Nomination geocoder, offer an alternative to commercial map APIs that need billing accounts [7]–[9].

Prior work and reports therefore address market growth, trust, and technology choices separately. Few sources document a lightweight, open-source implementation that combines owner-side listing, phone-based login, map-based pick-up selection, and per-user dashboards on a JavaScript stack, at a level of architectural detail suitable for replication. This paper addresses that gap.

3. PROPOSED SYSTEM

The proposed system, "Car Rental Services," provides a web-based marketplace in which vehicle owners publish listings and renters search, select, and book vehicles. It integrates a REST API, a cloud-hosted document database, a phone-based login flow, and a map-based location picker into one modular application that can be run on a developer machine with Node.js and a MongoDB Atlas connection string.

3.1. Problem Statement

Renting a vehicle locally in India often means contacting individual owners or agencies one by one. A renter has no single place to compare vehicle category, city, and daily rate, and an owner with one or two vehicles has no channel for reaching renters outside a personal network [CITE: source supporting the claim that individual owners and small operators dominate local rentals]. Without a shared record of listings and bookings, neither side can track what has been offered or reserved.

A first prototype of the system stored its data in the browser's local Storage. Data therefore existed on one browser of one device only, and a vehicle listed by an owner could not be seen by any renter on another device. The core problem addressed in this study is the absence of a shared, persistent backend and a simple, verifiable login flow that allow owners and renters to interact through the same data, together with an interface that lets renters state a pick-up location precisely.

3.2. System Design and Architecture

The architecture follows a layered and modular flow from the browser to the database (Fig.1). The client layer consists of six HTML pages and two shared JavaScript files. The landing page offers a search form and the four category cards; the login page runs a two-step phone and OTP flow; the category page lists vehicles for the chosen category; the booking page collects the pick-up point, dates, and driver name; the dashboard shows booking history and the user's own listings; and the listing page lets owners submit vehicles. The shared file app.js wraps the API calls, manages the session, resolves vehicle photographs through a lookup table of 25 model-to-image mappings, and falls back to a generated illustration of the vehicle body type when no photograph exists. The file map-picker.js implements the location picker.

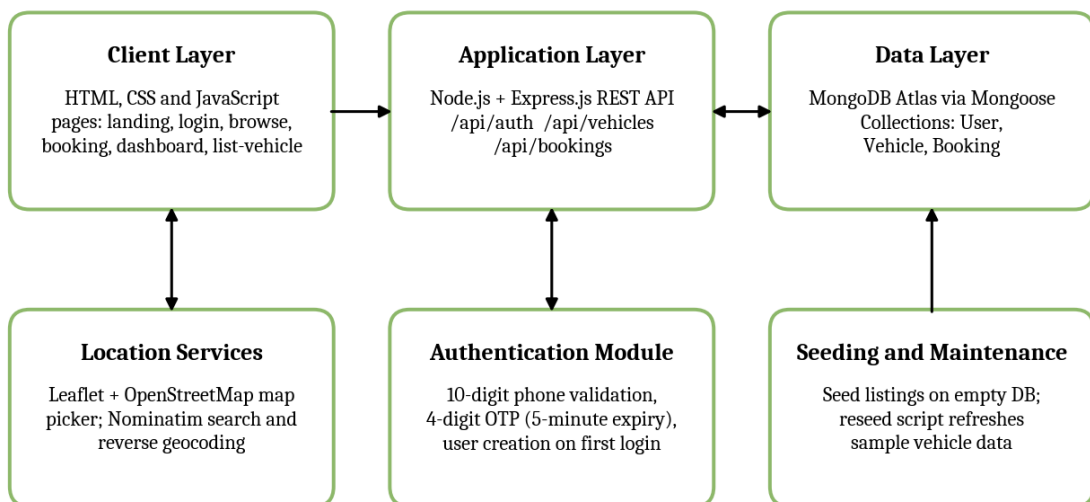


Fig.1: The diagram presents the layered architecture of the Car Rental Services platform, showing the client, application, and data layers together with the location, authentication, and seeding components.

The application layer is an Express 4 server that enables cross-origin requests, parses JSON bodies up to 10 MB so that base64-encoded photographs can be uploaded, and mounts three route modules under /api/auth,

/api/vehicles, and /api/bookings. On start-up it connects to MongoDB using a connection string read from an environment variable and, if the vehicle collection is empty, inserts the seed listings. The data layer is a MongoDB Atlas cluster accessed through Mongoose 8, with schemas for users, vehicles, and bookings [12], [13]. The authentication module validates the phone number and issues and verifies the OTP, and the location services module relies on Leaflet 1.9.4 with OpenStreetMap tiles and the Nomination service for search and reverse geocoding [7][9].

3.3. Database Design

Three collections store the application data (Table 1). Phone numbers identify users and are unique in the User collection. A Vehicle document records the owner's name and phone, the model, category, city, daily rate, description, an optional photograph, and a status that defaults to "Pending Review." A Booking document stores the renter's phone number together with a copy of the selected vehicle's details, the pick-up location, the trip dates, and the driver's name.

Table 1: Database collections and fields used by the platform.

Collection	Fields (required fields marked *)	Notes
User	phone*, created At	phone is unique
Vehicle	ownerName*, ownerPhone*, model*, category*, city*, rate* (Number), description, photoDataUrl, status, createdAt	status defaults to "Pending Review"
Booking	phone*, category, model, rate (Number), owner, city, pickupLocation, pickupDate, returnDate, driverName, createdAt	dates stored as strings

3.4. REST API

The API has eight endpoints (Table 2), plus a root route that reports whether the server is running. Vehicle listings can be filtered by category through a query parameter, and a separate route returns the vehicles listed by one owner. Bookings are returned most recent first.

Table 2: REST API endpoints.

Method	Endpoint	Function
POST	/api/auth/send-otp	Validates a ten-digit phone number, generates a four-digit OTP valid for five minutes
POST	/api/auth/verify-otp	Checks the OTP and expiry, creates the user on first login
GET	/api/auth/user/:phone	Returns the user record (used for "member since" on the dashboard)
GET	/api/vehicles?category=	Lists vehicles, optionally filtered by category, newest first
GET	/api/vehicles/mine?phone=	Lists vehicles submitted by one owner
POST	/api/vehicles	Creates a vehicle listing
POST	/api/bookings	Creates a booking
GET	/api/bookings/:phone	Lists a user's bookings, most recent first

3.5. Implementation Workflow

In the login flow, the client sends the phone number to the server, which generates the OTP, keeps it in an in-memory map with a five-minute expiry, and, because no SMS gateway is connected, returns it in the response so that the interface can display it in demo mode. After the user enters the code, the server compares it, deletes

the stored entry, creates the user if necessary, and the client records the phone number in local Storage as the session. The booking, dashboard, and listing pages redirect to the login page when no session exists.

For listing, the owner's form is validated on the client, the photograph is read as a base64 data URL, and the record is saved with the status "Pending Review." For booking, the client sends the selected vehicle's category, model, rate, owner, and city together with the pick-up location, dates, and driver name. The map picker opens in a modal Centered on Nagpur (latitude 21.1458, longitude 79.0882), lets the user click the map or search for a place, resolves the clicked point to an address through reverse geocoding, and writes the confirmed address into the form. A reseed script removes only the sample listings marked "Verified Owner" and re-inserts the current seed file, leaving user-submitted listings untouched.

4. RESULT AND DISCUSSION

The implemented system supports the complete rental workflow for both renters and owners. A user enters a ten-digit mobile number, receives a four-digit OTP, and on successful verification is signed in. The renter browses listings by category (Fig.2), selects a vehicle, chooses a pick-up point on the map, enters the dates and the driver's name, and confirms the booking, which is stored in the Booking collection and appears in the personal dashboard (Fig.3). Owners submit vehicles through the listing form, and each submission appears in the owner's dashboard with its review status. Because every page reads and writes through the same REST API and database, a booking or listing created on one device is visible on any other, which the earlier local Storage prototype could not do (Table 3).

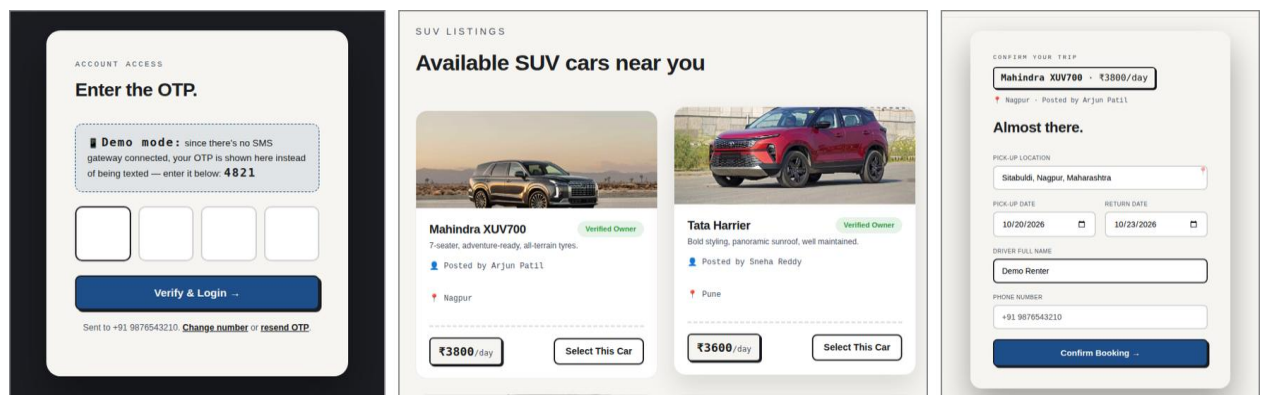


Fig.2: Renter workflow of the system: (a) OTP verification step of the phone-based login, (b) category-wise vehicle listings showing owner, city, and daily rate, and (c) booking form with map-based pick-up location, dates, and driver details. [VERIFY: replace with screenshots captured from your running MongoDB-backed deployment]

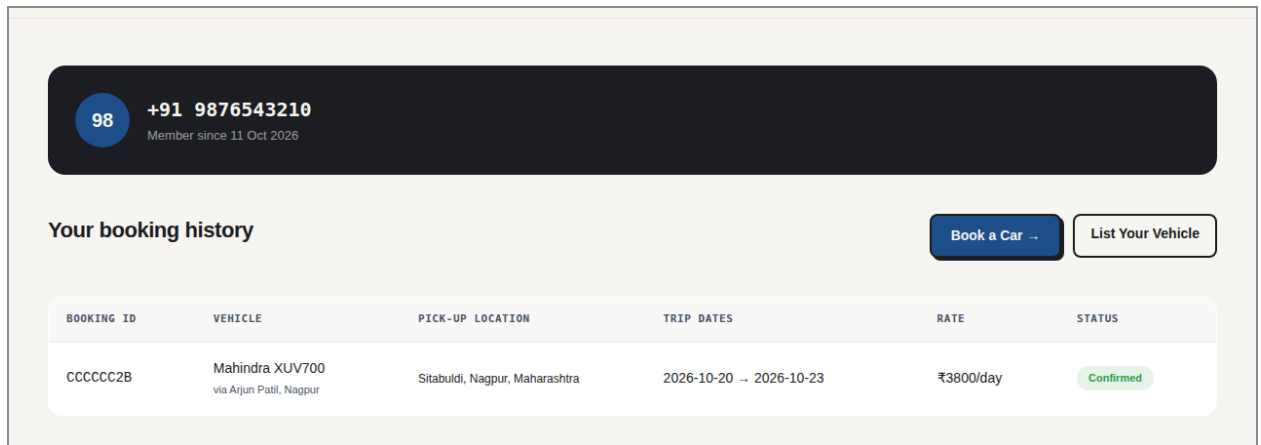


Fig.3: Personal dashboard showing the logged-in user's profile, booking history (booking ID, vehicle, pick-up location, trip dates, rate, and status), and the vehicles listed by the user. [VERIFY: replace with a screenshot from your live deployment]

Table 3: Comparison of the earlier browser-only prototype with the present backend.

Aspect	Earlier prototype	Present backend
Data storage	Browser local Storage	MongoDB database
Visibility of data	One browser on one device	Shared across all devices that call the API
OTP generation	Generated in the browser	Generated on the server (still returned in demo mode)

To populate the platform for demonstration, a seed file of 50 vehicle records was prepared. After removing 8 exact repetitions, 42 distinct listings remain (Table 4, Fig.4). They are spread over Nagpur (16), Pune (13), and Mumbai (13) and over four categories: SUV (14), Economy (12), Luxury (12), and Sedan (4). Mean daily rates rise from Rs. 1,738 for Economy to Rs. 2,550 for Sedan, Rs. 3,589 for SUV, and Rs. 7,175 for Luxury, so the average Luxury rate is about 4.1 times the average Economy rate. Sixteen listings carry the status "Verified Owner" and a bundled photograph, and the other 26 are inserted with the schema default "Pending Review." These figures describe the demonstration data shipped with the system and must not be read as market statistics.

Table 4: Summary of the seed dataset (42 distinct listings).

Category	Listings	Mean rate (Rs./day)	Minimum (Rs./day)	Maximum (Rs./day)
Economy	12	1,738	1,500	2,100
Sedan	4	2,550	2,400	2,700
SUV	14	3,589	2,750	4,500
Luxury	12	7,175	5,800	8,800

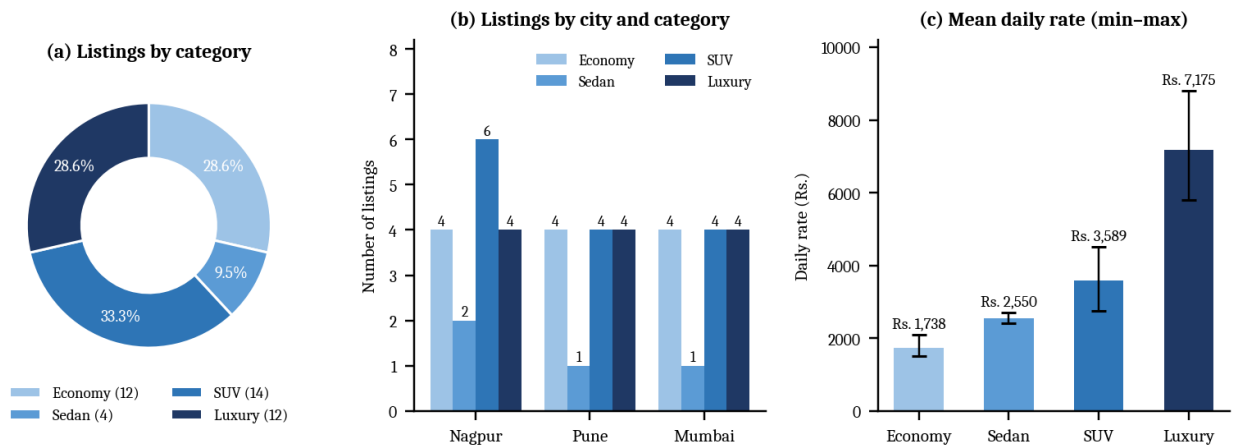


Fig.4: Analysis of the seed dataset of 42 distinct listings: (a) share of listings by vehicle category, (b) number of listings by city and category, and (c) mean daily rental rate by category with minimum–maximum range.

The evaluation reported here is functional. Response-time, load, and usability measurements were not carried out in this work [OPTIONAL: insert measured API latency, test-case results, or a usability survey here if you collect them].

Several limitations remain and define the next stage of development. The OTP is returned in the API response and is held in server memory, so it is lost on restart and cannot be shared between server instances, and it is generated with a non-cryptographic random function; the session consists only of the phone number kept in the browser, and bookings are fetched by phone number without server-side authentication. The system does not check vehicle availability or the order of pick-up and return dates, does not calculate a fare, and has no payment step, and a booking is shown as “Confirmed” immediately. Although listings carry a review status, no route or interface exists for approving them, and the label “Verified Owner” on the sample data is not the result of a verification process. Photographs are stored as base64 strings inside the database, and cross-origin access is open to all origins. Despite these limitations, the system demonstrates a working full-stack pattern in which owner listings, phone-based login, map-based pick-up selection, and per-user dashboards operate on a single shared database.

5. CONCLUSION

The “Car Rental Services” system provides a working, database-backed platform for peer-to-peer vehicle rental. By combining a Node.js and Express.js REST API, a MongoDB database, a phone-based OTP login, and an OpenStreetMap-based location picker, it lets owners publish vehicles and renters browse and book them across four vehicle categories and three cities. Moving from browser storage to a shared database removed the main restriction of the first prototype, and the documented architecture, data model, and API give a basis for replication and extension. The analysis also shows that the system is a prototype: security, availability management, and payment remain to be built before real-world use.

Future enhancements can significantly expand the system. Server-issued JSON Web Tokens and a real SMS gateway would secure login and protect user data. An availability calendar with date validation and fare calculation would prevent double bookings, and an online payment gateway would complete the transaction. An administrator workflow for reviewing and verifying listings, cloud image storage in place of base64 strings,

input validation, and rate limiting would improve safety and scalability. Ratings and reviews, recommendation and dynamic pricing, and a mobile application could increase trust and engagement. Rebuilding the client in React would align the implementation with the full MERN stack and make the interface easier to maintain.

Declaration of Conflicting Interests

The authors declare no potential conflicts of interest with respect to the research, authorship and publication of this article.

Funding

The author received no financial support for the research, authorship and publication of this article.

References

- [1] S. A. Shaheen and A. P. Cohen, "Carsharing and personal vehicle services: Worldwide market developments and emerging trends," *Int. J. Sustain. Transp.*, vol. 7, no. 1, pp. 5–34, 2013.
- [2] B. Cohen and J. Kietzmann, "Ride on! Mobility business models for the sharing economy," *Organ. Environ.*, vol. 27, no. 3, pp. 279–296, Sep. 2014.
- [3] R. Botsman and R. Rogers, *What's Mine Is Yours: The Rise of Collaborative Consumption*. New York, NY, USA: Harper Business, 2010.
- [4] J. Hamari, M. Sjöklint, and A. Ukkonen, "The sharing economy: Why people participate in collaborative consumption," *J. Assoc. Inf. Sci. Technol.*, vol. 67, no. 9, pp. 2047–2059, Sep. 2016.
- [5] R. Belk, "You are what you can access: Sharing and collaborative consumption online," *J. Bus. Res.*, vol. 67, no. 8, pp. 1595–1600, Aug. 2014.
- [6] E. Ert, A. Fleischer, and N. Magen, "Trust and reputation in the sharing economy: The role of personal photos in Airbnb," *Tourism Manage.*, vol. 55, pp. 62–73, Aug. 2016.
- [7] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, "HOTP: An HMAC-based one-time password algorithm," IETF, RFC 4226, Dec. 2005.
- [8] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-based one-time password algorithm," IETF, RFC 6238, May 2011.
- [9] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Digital identity guidelines: Authentication and lifecycle management," NIST, Gaithersburg, MD, USA, Special Publication 800-63B, Jun. 2017.
- [10] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, USA, 2000.
- [11] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to build high-performance network programs," *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, Nov./Dec. 2010.
- [12] M. Haklay and P. Weber, "OpenStreetMap: User-generated street maps," *IEEE Pervasive Comput.*, vol. 7, no. 4, pp. 12–18, Oct./Dec. 2008.
- [13] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in *Proc. SIGCHI Conf. Human Factors Comput. Syst. (CHI '94)*, Boston, MA, USA, 1994, pp. 152–158.
- [14] OWASP Foundation, "OWASP Top Ten," 2021. [Online]. Available: <https://owasp.org/www-project-top-ten/>