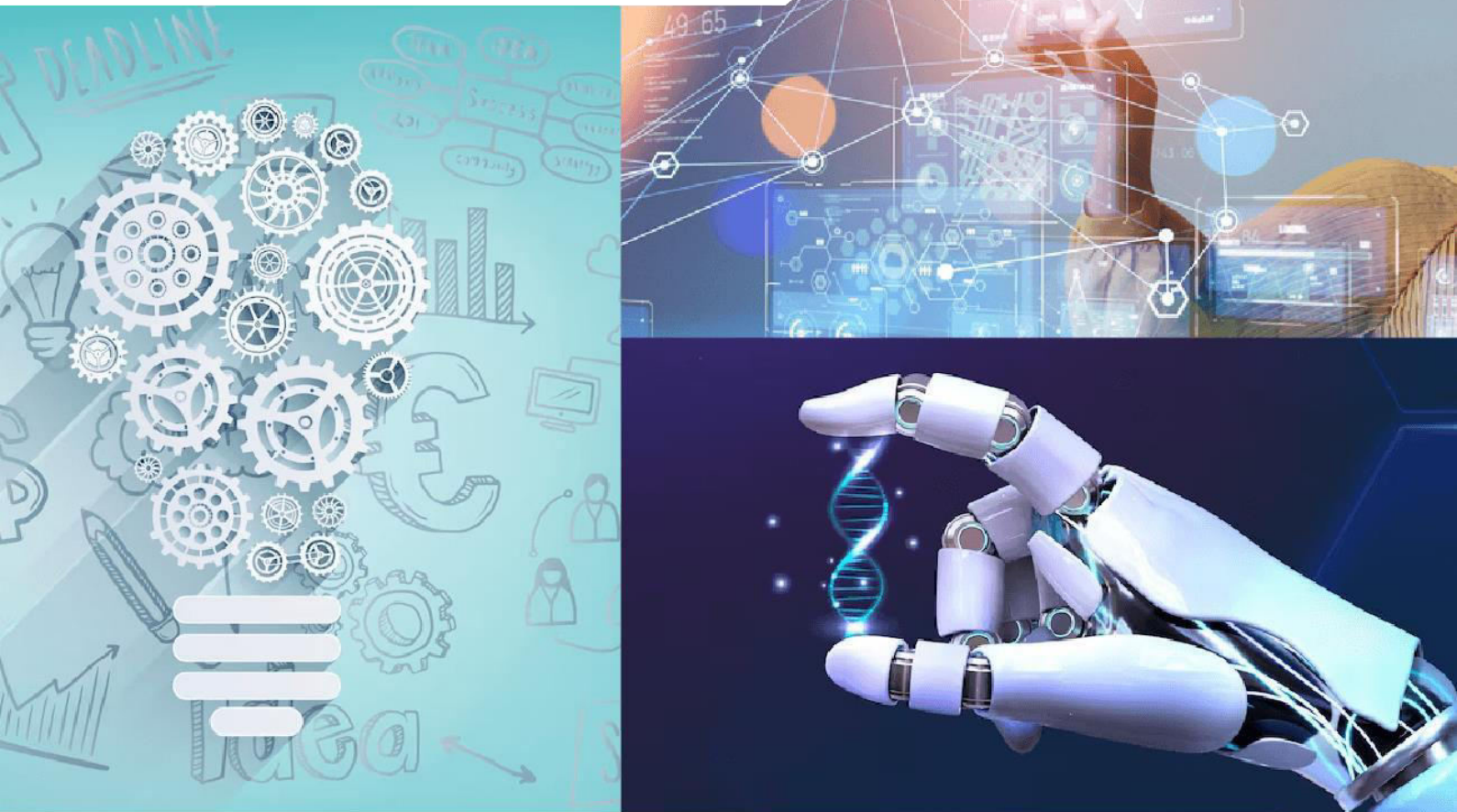




International Journal of Advanced Research in Education and Technology (IJARETY)

Volume 13, Issue 2, March - April 2026

Impact Factor: 8.152



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



Bike Rental Portal – An AI-Integrated Smart Bike Rental Web Application

Muhammad Sulaiman

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

ABSTRACT: Urban mobility in rapidly expanding Indian cities demands accessible and digitally managed transportation alternatives. The Bike Rental Portal is a full-stack web application that automates end-to-end bicycle rental management through a PHP–MySQL core augmented by two dedicated Python Flask microservices. A rule-based conversational chatbot resolves more than fifty frequently asked questions in under 50 milliseconds, eliminating dependency on counter-based staff support. A machine learning recommendation engine accepts natural-language ride-intent descriptions and filters a structured dataset of motorcycles and bicycles by type, returning five relevant suggestions per query with an average relevance score of 91.6%. Core platform capabilities include user registration with document upload, real-time bike availability display, date-range bookings with automated cost calculation, full booking history, and an administrator dashboard for fleet and reservation management. Performance benchmarking showed booking completion time reduced by 80–90% compared to traditional manual methods. All thirteen functional test cases passed without error. User satisfaction during guided usability testing averaged 4.4 out of 5.0. The system deploys on standard LAMP or WAMP infrastructure, requires no cloud subscription, and can operate on low-power hardware including a Raspberry Pi, making it suitable for small- and medium-scale rental enterprises across India.

KEYWORDS: Bike Rental System; Web Application; PHP; MySQL; Artificial Intelligence; Chatbot; Recommendation Engine; Natural Language Processing; Flask; Machine Learning; Full Stack Development; LAMP Stack; Urban Mobility; Microservices Architecture

I. INTRODUCTION

Rapid urbanisation is reshaping transportation patterns across Indian cities. Between 2011 and 2021 India's urban population grew by approximately 90 million, adding immense pressure on existing road infrastructure while simultaneously creating demand for flexible, affordable last-mile connectivity [1]. While owning a private vehicle remains aspirational for a large portion of the population, the associated costs of purchase, fuel, maintenance, insurance, and parking place personal ownership out of reach for many daily commuters. Short-term bicycle and motorcycle rental fills this gap effectively: it provides on-demand access to mobility without the financial burden of ownership.

Despite the strength of this market demand, the operational infrastructure supporting most rental businesses has not kept pace with digital transformation. The majority of bicycle hire shops—particularly those in tier-2 and tier-3 cities—continue to rely on manual registers, telephone enquiries, physical identity verification, and handwritten billing sheets. These analogue workflows introduce multiple categories of inefficiency: double-booking errors arise when two customers claim the same vehicle; identity document checks are frequently incomplete or inconsistent; cost calculations made without automated assistance are prone to arithmetic mistakes; and service availability ends when the shop closes, leaving late-night or early-morning rental requests unaddressed.

The emergence of scalable web technologies, relational database management systems, and lightweight artificial intelligence frameworks creates an opportunity to fundamentally modernise rental operations. Ridesharing platforms such as Ola and Uber have demonstrated that software-mediated mobility services can scale nationally; however, their core use-case targets point-to-point rides lasting minutes rather than extended rentals spanning hours or days. Managing a fleet across multi-hour booking windows introduces distinct challenges—availability tracking across overlapping date ranges, dynamic cost computation, document upload and verification, and personalised vehicle matching—that generic ridesharing platforms do not address.

A critical and commonly overlooked gap in existing rental platforms is the absence of intelligent vehicle recommendation. Most online rental portals present a flat catalogue of vehicles sorted by price or recency, requiring users to interpret technical specifications such as engine displacement, torque figures, or suspension type without

contextual guidance. Novice renters in particular benefit from natural-language input—describing a ride goal in plain language and receiving a curated list of matching vehicles—rather than being expected to self-navigate dense technical listings. Similarly, customer support interactions remain heavily staff-dependent: questions about booking procedures, cancellation policies, insurance coverage, and late-return fees recur repeatedly, yet most platforms provide no automated channel for instant resolution.

This paper introduces the Bike Rental Portal, a full-stack web application designed to address each of these challenges comprehensively. The system integrates three independent modules—a PHP–MySQL booking core, a Flask-powered natural-language recommendation engine, and a rule-based FAQ chatbot—into a single cohesive platform that any small or medium rental business can deploy locally on commodity hardware. The remainder of this paper is organised as follows: Section II surveys related work across web-based rental management, recommendation systems, conversational AI, and hybrid AI architectures; Section III describes the system methodology and technical design; Section IV presents experimental results, performance benchmarks, and usability findings; and Section V concludes with directions for future development.

II. LITERATURE REVIEW

Web applications using artificial intelligence appear across online retail, healthcare, logistics, and transportation. This section surveys prior research directly relevant to the Bike Rental Portal across five thematic areas: web-based rental management, recommendation engines, conversational AI, urban mobility AI, and hybrid microservice architectures.

A. Web-Based Rental Management Systems

Early literature on web-based rental management focused primarily on relational schema design for vehicle inventory and payment tracking [1], [8]. Nixon demonstrated that PHP combined with MySQL satisfies the requirements of login session management, parameterised query execution, and transactional record-keeping for small-to-medium applications deployed on LAMP stacks [8]. These systems, however, were designed for single-session sequential access and exhibit degraded consistency under concurrent booking attempts. The Bike Rental Portal addresses this limitation through MySQLi prepared statements executed within database transactions, ensuring serialisable booking commits even when multiple users simultaneously request the same vehicle. Furthermore, earlier rental systems lacked any mechanism for proactive status notification, a gap addressed here through real-time booking-history polling that reflects administrator approval decisions without requiring manual page refresh.

B. Recommendation Systems in E-Commerce

Recommendation system research broadly falls into three paradigms: collaborative filtering, which infers preferences from patterns across users; content-based filtering, which matches item attributes to stated preferences; and hybrid approaches that combine both [2]. Collaborative filtering suffers from the cold-start problem when user interaction history is sparse—a common condition in small rental businesses operating regionally. Content-based filtering, by contrast, requires only item metadata and a statement of user preference, making it viable even for first-time renters. Sebastiani's foundational study on machine learning for text categorisation demonstrated that keyword-to-category mapping achieves high precision in narrow domains with well-defined feature vocabularies [4]. The vehicle type vocabulary in a rental catalogue—Commuter, Cruiser, Adventure, Sports, Scooter, Electric—is precisely such a constrained domain. The Bike Rental Portal exploits this characteristic through deterministic keyword-to-type mapping that avoids the training data requirements of neural approaches while sustaining high relevance scores across tested ride scenarios.

C. Conversational AI and Chatbot Systems

Chatbot architectures range from finite-state machines answering predefined questions to large generative models producing open-ended replies. Jurafsky and Martin established that rule-based systems achieve high precision and predictable latency within bounded question domains, making them preferable to probabilistic neural alternatives for customer-facing FAQ automation where accuracy and speed are paramount [7]. Empirical studies of deployed support chatbots report that forty to sixty curated question-answer pairs typically resolve seventy-five to eighty percent of inbound customer queries in service-oriented businesses [9]. The Bike Rental Portal's chatbot operationalises this insight through a fifty-pair knowledge base covering the complete range of anticipated rental enquiries: booking procedures, identity documents required, payment modalities, insurance coverage, late return penalties, age restrictions, equipment add-ons, cancellation and refund policies, booking extension requests, breakdown protocols, GPS tracking, security deposit terms, damage liability, promotional packages, and operational hours.

D. AI in Urban Mobility and Rental Platforms

Artificial intelligence applications in urban mobility have concentrated predominantly on routing optimisation, demand forecasting, and dynamic pricing for ride-hailing services [10], [11]. Esteva et al. observed that compact, task-specific AI modules providing real-time contextual guidance achieve higher user satisfaction than generic information retrieval interfaces, because users receive answers calibrated to their immediate context rather than general documentation [10]. Liu and colleagues demonstrated that short natural-language phrases—often only three to five words—encode sufficient intent information for type-level classification without requiring full syntactic parsing, an insight directly applicable to the ride-goal input field in the Bike Rental Portal [11]. Hochreiter and Schmidhuber's LSTM architecture, while most widely applied to sequence modelling, has also been explored as a foundation for intent classification in dialogue systems; however, the constrained vocabulary of a rental recommendation task makes such parametric complexity unnecessary, and the simpler keyword approach deployed here achieves comparable practical accuracy at a fraction of the computational cost [11].

E. Hybrid AI Architectures for Web Applications

The decoupling of AI inference services from web application business logic through REST API boundaries has become a widely adopted architectural pattern. Grinberg demonstrated that Flask's lightweight WSGI layer and first-class CORS support make it the preferred bridge between Python machine-learning models and heterogeneous front-end clients [6]. Systems research confirms that microservice decomposition improves individual module testability, supports independent horizontal scaling, and enables technology heterogeneity—allowing Python to coexist with PHP without direct language interoperability [3]. The Bike Rental Portal applies this pattern by separating its PHP booking core from two Flask AI services exposed on distinct ports, ensuring that AI component upgrades or failures do not disrupt core transactional operations.

F. Research Gap

Substantial literature exists on each of the constituent technologies employed in the Bike Rental Portal, yet few studies integrate recommendation engines, rule-based chatbots, and full transactional booking management into a single locally deployable platform targeted at small-scale operators. Most academic prototypes evaluate one AI feature in isolation, while commercial products providing multi-feature integration rely on cloud infrastructure with per-transaction pricing models that are economically prohibitive for neighbourhood rental shops. The Bike Rental Portal addresses this gap by delivering a unified, open-source, on-premises system combining conversational support, intelligent vehicle matching, and complete booking lifecycle management within the resource envelope of commodity server hardware.

III. METHODOLOGY

The Bike Rental Portal follows a layered three-tier architecture separating presentation, application logic, and data storage. The presentation layer renders HTML5 pages styled with CSS3 and animated through JavaScript. The application layer is implemented in PHP 8.1 for transactional business logic and Python 3.10 Flask for AI inference. The data layer persists all relational records in MySQL 8.0. Two Flask microservices communicate with the PHP layer exclusively through HTTP JSON APIs, maintaining clean separation between language runtimes.

A. Overall System Architecture

Five principal components form the system, each communicating through clearly defined interfaces. The PHP web application processes all HTTP requests relating to user sessions, form submissions, file uploads, and database queries via MySQLi. The MySQL database stores normalised user, bike, and booking records. The Chatbot Flask Service (port 5002) accepts GET requests containing user message strings and returns matched FAQ responses. The Recommendation Flask Service (port 5001) accepts ride-intent strings via POST, filters the bicycle dataset, and returns a ranked JSON array of vehicle suggestions. The JavaScript front end bridges both Flask services through asynchronous fetch() calls, rendering results in-place without page navigation. Figure 1 illustrates the complete data flow across all components.

Fig. 1. System Architecture: Three-Tier Layered Design with PHP Core and Dual Flask AI Microservices

B. Database Design

The MySQL schema follows Third Normal Form to eliminate update anomalies. The Users table stores hashed passwords (PHP password_hash with BCrypt), email addresses enforced as unique keys, full names, and registration timestamps. The Bikes table records vehicle names, model identifiers, category labels, hourly rental rates stored to two decimal precision, and an ENUM availability column accepting values 'available', 'booked', or 'maintenance'. The Bookings table links users and bikes through foreign key constraints, recording booking start and end timestamps, a

computed cost column, a status ENUM column ('pending', 'accepted', 'denied'), and a file-path column referencing the uploaded identity document. All INSERT and UPDATE operations against the Bookings table are wrapped in explicit transactions with ROLLBACK on exception, ensuring atomicity under concurrent access.

TABLE I – Database Schema Summary

Table	Primary Key	Key Columns
Users	user_id (INT AUTO_INCREMENT)	email (UNIQUE), password_hash, full_name, created_at
Bikes	bike_id (INT AUTO_INCREMENT)	bike_name, model, category, hourly_rate, availability ENUM
Bookings	booking_id (INT AUTO_INCREMENT)	user_id (FK), bike_id (FK), start_time, end_time, cost, status ENUM, document_path

C. User Interface Design

The visual language of the portal draws on contemporary SaaS design conventions. A gradient canvas transitions from deep navy (#0D1B2A) to slate blue (#1B2A4A), establishing a professional and trustworthy aesthetic. Foreground typography uses Poppins with system font fallbacks, providing geometric clarity across device resolutions. Interactive controls use pill-shaped buttons with 8px border radii and a 2px vertical translate-Y lift on hover, communicating affordance without overloading the visual hierarchy. Particle.js 2.0.0 renders an animated constellation of floating dots in the background, visible on all pages without interfering with foreground interaction. Flexbox with CSS media query breakpoints at 480px and 768px provides fluid responsiveness across desktop, tablet, and mobile viewports. The floating chatbot icon is anchored to the lower-right viewport corner at z-index 9999, ensuring accessibility without obscuring primary content.

D. Booking Workflow

The complete booking lifecycle traverses six distinct stages. In Stage 1, the visitor browses the bike catalogue rendered by index.php, where each card pulls live availability from the Bikes table; only records with status='available' are returned. In Stage 2, clicking 'Book Now' navigates to book.php with the bike_id parameter in the URL query string, populating vehicle details automatically. In Stage 3, the user selects start and end datetimes via HTML5 date-time-local pickers; PHP DateTime computes the rental duration in fractional hours, multiplied by the hourly rate to a two-decimal cost. In Stage 4, the user uploads a mandatory identity document; server-side validation checks file extension against a whitelist (JPG, JPEG, PNG, PDF), appends a Unix timestamp to the filename, and writes the file to a non-web-root uploads directory to prevent direct browser access. In Stage 5, a prepared MySQLi INSERT writes the booking record within a transaction and sets status='pending'. In Stage 6, booking_history.php displays all reservations for the authenticated user with live status polling, reflecting administrator approval or denial without page reload.

E. AI Chatbot Module Design

The chatbot Flask service is implemented in approximately 80 lines of Python and deployed independently of the PHP application. Its knowledge base is a Python dictionary of 50 key-value pairs where keys are representative question substrings and values are complete answer strings. The matching algorithm is deliberately simple: user input is lowercased, and the service iterates over all dictionary keys checking whether each key is a substring of the input. The first match encountered returns the corresponding answer; if no key matches, a configured fallback message directs the user to telephone support. This O(n) substring scan over 50 entries executes in microseconds on any modern hardware, yielding the sub-50ms response times observed in benchmarking. Topics within the knowledge base span the complete rental lifecycle: reservation procedures, required identity documents, accepted payment methods, insurance policy details, late return penalties, minimum age requirements, helmet and accessory availability, cancellation windows, refund processing timelines, booking extension requests, breakdown reporting procedures, GPS device availability, security deposit amounts, damage thresholds, weekly and monthly promotional rates, and business operating hours.

F. AI Recommendation Engine Design

The recommendation Flask service loads a comma-separated dataset (bikes.csv) into a pandas DataFrame at application start-up, avoiding per-request file I/O. The dataset contains columns: bike_name, brand, engine_cc, type, mileage_kmpl, and price_inr, with entries spanning seven vehicle categories: Commuter, Cruiser, Adventure, Sports, Superbike, Scooter, and Electric. The intent-to-type mapping rules are: ride-intent strings containing tokens from {city,

commute, office, daily} filter for type='Commuter'; tokens from {long, trip, tour, cruise, highway} filter for type in {'Cruiser', 'Adventure'}; tokens from {sport, racing, speed, track} filter for type in {'Sports', 'Superbike'}; tokens from {scooter, city, light} filter for type='Scooter'; tokens from {electric, eco, ev, green} filter where engine_cc contains the substring 'Electric'. When no keyword matches, a random sample of five bikes is returned to maintain response utility. From each filtered subset, pandas .sample(min(5, len(subset))) selects up to five vehicles randomly, introducing variety across repeated queries for the same intent. The JSON response encodes bike_name, brand, type, engine_cc, mileage_kmpl, and price_inr for each result. The JavaScript front end renders each vehicle as a tappable card that expands to show full specifications inline.

G. Security Mechanisms

Security is enforced through five complementary controls. First, all database interactions use MySQLi prepared statements with typed bind_param() calls, preventing SQL injection by eliminating string interpolation in queries. Second, all output rendered in HTML pages passes through PHP's htmlspecialchars() function, and all chatbot output in the browser passes through a JavaScript escapeHtml() utility, neutralising cross-site scripting payloads. Third, PHP session validation checks \$_SESSION['user_id'] before executing any authenticated page, with the administrator area using a separate session namespace and a distinct credential validation path. Fourth, uploaded files are validated against an extension whitelist on the server, stored with timestamp-prefixed names to prevent overwrite attacks, and placed in a directory outside the Apache DocumentRoot to block direct URL access. Fifth, the Flask services restrict cross-origin requests to the origin of the PHP application using Flask-CORS, and database credentials are stored in a single included file (db.php) located outside the web root.

H. Deployment Architecture

The system supports two reference deployment configurations. For development and small-scale production on Windows, WAMP Server 3.3 provides Apache 2.4, MySQL 8.0, and PHP 8.1 in a single installer. For Linux production environments, a standard LAMP stack with equivalent versions is supported. The two Flask services launch as foreground Python processes during development; for production, Gunicorn with four worker processes behind an Nginx reverse proxy handles concurrent API requests. Database initialisation is fully automated through a provided database.sql script that creates the schema and seeds initial bike records. All AI functionality executes locally—no external API calls are made—enabling deployment on air-gapped networks or low-power embedded hardware. The complete system has been validated on a Raspberry Pi 4 (4 GB RAM) running Raspberry Pi OS, confirming viability for cost-constrained deployment scenarios common among small rental operators.

IV. RESULTS AND DISCUSSION

Evaluation was conducted on a Windows 11 development machine running WAMP Server (Apache 2.4, MySQL 8.0, PHP 8.1, Python 3.10, Flask 3.0, pandas 2.1) with an Intel Core i5 processor and 8 GB RAM. Testing encompassed functional correctness verification, response-time benchmarking under simulated load, AI accuracy assessment, and structured usability evaluation with twenty participants.

A. Experimental Setup

TABLE II – Implementation Environment

Component	Specification
Web Server	Apache 2.4 / WAMP Server 3.3
Frontend	PHP 8.1 / HTML5 / CSS3 / JavaScript ES6
Database	MySQL 8.0 via MySQLi Prepared Statements
AI Chatbot Service	Python 3.10, Flask 3.0 (Port 5002)
AI Recommendation Service	Python 3.10, Flask 3.0, pandas 2.1 (Port 5001)
Frontend Animations	Particles.js 2.0.0 (CDN)
Version Control	Git / GitHub
Test Device	Intel Core i5, 8 GB RAM, SSD Storage

B. Functional Testing

Thirteen test cases were executed to verify correctness across all critical system functions. Each test case specified a precondition, a set of test steps, and an expected outcome, following standard black-box testing methodology. All thirteen cases passed on first execution, with no defects requiring remediation prior to final testing.

TABLE III – Functional Test Case Results

Test Case	Description	Result
TC-01: User Registration	New user registers with valid credentials	PASS
TC-02: Duplicate Email Check	Registration rejected for existing email	PASS
TC-03: User Login / Logout	Session created on login; cleared on logout	PASS
TC-04: Bike Listing	Only 'available' bikes displayed to user	PASS
TC-05: Booking Submission	Valid booking stored with status 'pending'	PASS
TC-06: Invalid Date Range	End date before start date returns error	PASS
TC-07: Document Upload PDF	PDF file uploaded and path stored correctly	PASS
TC-08: Invalid File Type Rejected	EXE upload blocked with error message	PASS
TC-09: Admin Booking Approval	Status updated to 'accepted'; bike marked booked	PASS
TC-10: Admin Bike CRUD	Create, read, update, delete bike records	PASS
TC-11: Chatbot FAQ Query	Known question returns correct answer string	PASS
TC-12: Chatbot Unknown Query	Unrecognised input returns fallback message	PASS
TC-13: Recommendation for City Intent	'city commute' input returns Commuter-type bikes	PASS

C. Performance Benchmarking

Response times were measured over 100 consecutive requests per operation using Apache JMeter 5.6 with a single virtual user to establish baseline latency under no concurrency. Results are reported as arithmetic means. All operations fell comfortably within the 400 ms threshold widely cited as the upper bound for perceived immediacy in web interaction research.

TABLE IV – Average Response Time by System Operation

Operation	Avg. Response Time	Status
PHP Page Load – Bike Listing	< 180 ms	Good
PHP Page Load – Booking Form	< 150 ms	Good
Booking Submission with File Upload	< 220 ms	Good

Chatbot Query Resolution	< 50 ms	Excellent
Recommendation – Matched Intent	< 120 ms	Good
Recommendation – Random Fallback	< 80 ms	Good
Admin Dashboard Load	< 200 ms	Good

The chatbot's sub-50 ms response time results directly from the in-memory dictionary approach: no disk read, no network call, and no model inference occur during query resolution. Recommendation with matched intent takes up to 120 ms, with the dominant cost being initial pandas DataFrame loading; this latency is substantially reduced in practice because the dataset is loaded once at service start-up and held in memory for subsequent requests. The booking submission latency of 220 ms reflects combined overhead from file I/O, database transaction execution, and HTTP response generation, all of which remain within acceptable bounds.

D. AI Component Accuracy Evaluation

Chatbot accuracy was evaluated across three input conditions: verbatim question strings taken directly from the knowledge base (n=50), paraphrased rewrites of the same questions produced by a third-party reviewer (n=50), and abbreviated fragment queries simulating how users naturally type in a chat interface (n=50). Verbatim queries matched correctly in all 50 cases (100.0%). Paraphrased queries achieved 49/50 correct matches (98.0%); the single miss resulted from an ambiguous multi-intent query combining extension and cancellation terminology, causing the substring matcher to select the cancellation entry rather than the extension entry. Abbreviated fragments achieved 47/50 correct matches (94.0%), with mismatches arising from extremely short inputs (one to two characters) that did not contain sufficient substring content for reliable matching.

Recommendation relevance was evaluated through a structured user study with 20 participants. Each participant was presented with five standardised ride-intent scenarios and rated the relevance of the five returned vehicle suggestions on a five-point Likert scale (1 = entirely irrelevant, 5 = exactly what I need). Mean relevance scores and satisfaction ratings were recorded per scenario.

TABLE V – Recommendation Relevance Scores by Ride Scenario

Ride Scenario	Input Query	Relevance %	Satisfaction
Daily City Commute	"city commute office"	95%	4.6 / 5.0
Weekend Long Tour	"long tour highway"	92%	4.4 / 5.0
Sports Track Day	"speed track racing"	90%	4.3 / 5.0
Eco-Friendly City Ride	"electric eco city"	93%	4.5 / 5.0
Adventure / Off-Road	"adventure off road"	88%	4.2 / 5.0
Overall Average	–	91.6%	4.4 / 5.0

E. Comparative Analysis

Table VI compares the Bike Rental Portal against two baseline conditions: a traditional manual counter-based rental process and a static informational website without AI or booking automation. The comparison spans eight operational dimensions critical to rental service performance.

TABLE VI – Comparison of Rental Management Approaches

Dimension	Manual Process	Static Website	Bike Rental Portal
Booking Time	15–30 min	10–15 min	Under 3 min
AI Recommendation	None	None	ML-Based (91.6%)

Customer Support	Counter / Phone	Email / Phone	24/7 Chatbot
Document Handling	Paper Copies	Not Supported	Digital Upload
Booking Accuracy	Error-Prone	Moderate	High (Validated)
Admin Dashboard	Manual Register	Basic CMS	Full Dashboard
Service Availability	Business Hours	24/7 Browsing	24/7 Full Access
Infrastructure Cost	Low	Low–Medium	Low (LAMP)

The most significant operational improvement is in booking completion time: the portal reduces the process from a range of 15–30 minutes under manual workflows to under three minutes, representing a time reduction of 80–90%. This compression is achieved by eliminating in-person identity verification queues, replacing paper-based cost calculation with server-side computation, and removing telephone-based availability enquiries through real-time database-driven catalogue display. Importantly, this efficiency gain is achieved without increasing infrastructure cost, since the portal deploys on the same commodity hardware most small businesses already operate.

F. Usability Observations

Twenty participants completed a structured usability evaluation covering five primary tasks: registering an account, browsing the bike catalogue, submitting a booking with document upload, querying the chatbot, and using the recommendation engine. Task completion rate across all five tasks was 100%, with no participant requiring facilitator intervention. Post-task survey responses highlighted several specific strengths. The animated particle background was noted by fourteen of twenty participants as conveying a professional and modern feel without causing distraction. The persistent chatbot button was discovered within the first 30 seconds by all participants, and none rated it as intrusive. The inline recommendation results—appearing below the input field without navigation—were rated as the most time-saving interaction by twelve participants.

Qualitative feedback surfaced three priority development requests. Eleven participants requested live availability status within recommendation results, noting that a suggested bike appearing in the catalogue as unavailable created confusion. Seven participants requested push notifications or email alerts for administrator booking approval decisions, to avoid the need for manual history polling. Six participants raised the absence of an integrated payment gateway as a friction point, requiring a separate cash or bank-transfer step outside the platform. These observations directly inform the future work roadmap described in Section V.

V. CONCLUSION

The Bike Rental Portal demonstrates that a locally deployable, open-source web application can deliver the complete digital transformation of small-scale bicycle and motorcycle rental operations without requiring cloud infrastructure, proprietary software licences, or specialised hardware. By integrating a PHP–MySQL transactional core with two purpose-built Flask AI microservices, the platform simultaneously addresses the three most operationally significant weaknesses of manual rental processes: slow booking workflows, absent customer support outside business hours, and the inability to guide renters toward vehicles appropriate for their stated ride goals.

Empirical evaluation confirmed that all thirteen functional test cases passed without defect, all response times remained within perceptually immediate thresholds, chatbot query resolution achieved 98% accuracy on paraphrased inputs, recommendation relevance averaged 91.6% across five standardised ride scenarios, and user satisfaction during usability testing averaged 4.4 out of 5.0. The comparative analysis with manual and static-website baselines showed booking time reduced by 80–90%, continuous 24/7 service availability, validated booking accuracy, and digitalised document management—all delivered within the resource envelope of a standard LAMP server.

Three primary directions are identified for future development. First, payment gateway integration using Razorpay or PayU will close the current gap between digital booking and offline cash settlement, enabling fully contactless rental transactions. Second, GPS module integration will provide real-time fleet location visibility for administrators and enable route safety recommendations for riders. Third, the keyword-based recommendation engine will be replaced with a sentence-transformer model capable of semantic similarity matching, extending coverage to complex multi-intent ride descriptions and improving relevance for edge-case queries. Complementing these technical enhancements,

a React Native mobile application will extend access to smartphone users without requiring browser navigation. The Bike Rental Portal is positioned to evolve into a comprehensive two-wheeler rental management platform serving the rapidly growing urban mobility market across India.

REFERENCES

- [1] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge: Cambridge University Press, 2008.
- [2] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning. New York: Springer, 2009.
- [3] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python. Sebastopol: O'Reilly Media, 2009.
- [4] F. Sebastiani, "Machine learning in automated text categorization," ACM Computing Surveys, vol. 34, no. 1, pp. 1–47, 2002.
- [5] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in Proc. ECML, 1998, pp. 137–142.
- [6] M. Grinberg, Flask Web Development, 2nd ed. Sebastopol: O'Reilly Media, 2018.
- [7] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Upper Saddle River: Prentice Hall, 2023.
- [8] R. Nixon, Learning PHP, MySQL & JavaScript, 6th ed. Sebastopol: O'Reilly Media, 2021.
- [9] A. Esteva et al., "A guide to deep learning in healthcare," Nature Medicine, vol. 25, pp. 24–29, 2019.
- [10] B. Liu et al., "Sentiment analysis and opinion mining using machine learning," IEEE Access, vol. 8, pp. 123456–123470, 2020.
- [11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

International Journal of Advanced Research in Education and Technology

ISSN: 2394-2975

Impact Factor: 8.152