IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154262

Dormant Asset Rental for Small Businesses: A Web-Based Digital Marketplace for SME Asset Monetisation

Lalit Sabale[1], Chaitanya Panmand[2], Somnath Mulik[3], Rishabh Pawar[4], Dr. A R Sonule[5], Dr. Manoj M Deshpande[6], Dr. A S Deore[7]

Department of Computer Engineering, A.C. Patil College of Engineering, Kharghar, Navi Mumbai[1-5]

Head of Department, Department of Computer Engineering, A.C. Patil College of Engineering, Kharghar, Navi Mumbai[6]

Principal, Department of Computer Engineering, A.C. Patil College of Engineering, Kharghar, Navi Mumbai[7]

Abstract: Small and medium-sized enterprises (SMEs) in India bear a disproportionate financial burden from dormant assets — physical and technological resources that are owned but not continuously utilised. These assets, which include vehicles, machinery, professional equipment, electronics, and event infrastructure, depreciate in value while incurring storage, insurance, and maintenance costs without generating any compensatory revenue. This paper presents Rentify, a full-stack, role-based digital rental marketplace architected to address this problem by enabling SMEs to monetise their underutilised assets through a structured, trusted, and feature-complete online platform. The system is implemented using HTML5, CSS3, and vanilla JavaScript for the frontend presentation layer, Python Flask for the backend REST API, and MySQL 8.0 as the relational database. The platform supports three user roles — renters, vendors, and administrators — each with dedicated dashboards and precisely defined access controls enforced through JSON Web Token authentication. Key features include a multi-category product catalogue, a flexible booking engine with hourly, daily, and weekly pricing tiers, coupon-based discount management, a simulated Razorpay-compatible payment workflow, and a comprehensive transaction history module. The database schema comprises thirteen normalised tables designed to the Third Normal Form. Systematic functional testing across all three user roles confirmed the correctness of all fifty-five tested workflows. API response times averaged 42 milliseconds on a local development server, well within the 200-millisecond threshold for a responsive user experience. The platform directly addresses four research gaps identified in the existing literature: SME neglect in current rental platforms, absence of multi-category solutions, poor identification-to-rental integration, and lack of transparent earning models.

Keywords: Dormant Assets, SME Marketplace, Rental Platform, Flask, MySQL, JWT Authentication, Sharing Economy, Asset Monetisation

1. INTRODUCTION

1.1 Background

Small and medium-sized enterprises constitute the foundational layer of the Indian economy. According to the Ministry of Micro, Small and Medium Enterprises, the SME sector accounts for approximately 30 percent of India's national Gross Domestic Product, contributes nearly 45 percent of the country's total exports, and provides direct and indirect employment to over 110 million individuals. Despite this outsized economic contribution, SMEs operate under persistent resource constraints and suffer from structural financial inefficiencies that are rarely addressed by technology platforms or policymakers.

One of the most financially consequential yet least studied of these inefficiencies is the phenomenon of dormant assets. A dormant asset, in the context of business operations, is defined as any physical or technological resource that a business has acquired for operational purposes but does not employ at full productive capacity on a continuous basis. The category is broad and includes delivery vehicles required only seasonally, professional cameras purchased for a single product launch, construction equipment deployed for one contract and then stored indefinitely, event sound systems used a few weekends per year, and laptop computers assigned to part-time or contract employees. Each such asset represents a committed capital expenditure that continues to depreciate, incur storage and insurance costs, and consume maintenance resources — all without generating any revenue to offset these ongoing obligations.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154262

The sharing economy, a term used to describe economic systems in which participants share access to goods and services rather than transferring permanent ownership, has demonstrated extraordinary capacity to unlock value from underutilised resources in the consumer domain. Platforms such as Airbnb in accommodation, Ola and Uber in transportation, and TaskRabbit in labour have collectively generated hundreds of billions of dollars in economic activity by applying digital intermediation to the challenge of idle capacity. However, the business-to-business dimension of the sharing economy — and specifically the SME dormant asset segment — remains almost entirely underserved by existing platforms.

1.2 Problem Statement

The core problem addressed by this research is threefold. First, SMEs in India own substantial physical and technological assets that remain underutilised for significant periods, representing a direct financial loss that could be converted into recurring passive income through a structured rental mechanism. Second, no dedicated, trustworthy, and feature-complete digital platform currently exists in India to facilitate dormant asset rental specifically within the SME ecosystem; existing platforms such as OLX and Quikr are classifieds tools unsuitable for managed rental workflows, while specialised platforms such as Furlenco and Zoom Car are vertically constrained to single asset categories. Third, the informal rental arrangements that SMEs occasionally make through personal networks are legally unprotected, financially opaque, logistically unreliable, and impossible to scale.

The absence of a suitable platform imposes a double inefficiency on the economy: asset owners are unable to monetise idle resources, and asset seekers — startups, event organisers, contractors, and other SMEs — are forced to purchase equipment outright when temporary rental access would be both economically superior and operationally sufficient.

1.3 Objectives

The objectives of this project are as follows. The first objective is to design and implement a full-stack web-based rental marketplace that enables SME owners to list dormant physical assets for short-term rental with detailed descriptions, multi-tier pricing, and inventory management. The second objective is to implement a secure, stateless, role-based authentication system supporting three distinct user types — renters, vendors, and administrators — with appropriate access controls enforced at the API level. The third objective is to develop a booking engine supporting hourly, daily, and weekly pricing models with automatic cost calculation, coupon discount application, and security deposit management. The fourth objective is to design a normalised thirteen-table relational database schema with referential integrity, appropriate indexing, and efficient query performance. The fifth objective is to validate the platform through systematic functional, performance, and usability testing and document the outcomes as evidence of technical achievement.

2. LITERATURE REVIEW

The design of Rentify draws upon a substantial body of existing work in rental platform architecture, sharing economy economics, and digital marketplace trust design. This section reviews five representative systems and research contributions, identifies the gaps that remain in the current state of the art, and positions Rentify's contribution within that landscape.

2.1 Airbnb: Peer-to-Peer Accommodation Rental

Airbnb, founded in 2008, is the most extensively studied peer-to-peer rental platform in the academic literature. The platform pioneered several mechanisms that are now considered foundational to rental marketplace design: asymmetric identity verification through government-issued documents, a symmetric dual-review system in which both guests and hosts rate each other after each transaction, dynamic pricing algorithms responsive to seasonal demand signals and local event calendars, and a host guarantee insurance programme that mitigates asset damage risk. Zervas, Proserpio, and Byers (2017) provide a rigorous econometric analysis of Airbnb's impact on the hotel industry, establishing that the platform's core competence lies not in the accommodation itself but in the trust infrastructure that enables strangers to transact over high-value physical assets. However, Airbnb is architecturally exclusive to accommodation and provides no framework for the heterogeneous physical asset types owned by SMEs. Its trust mechanisms, pricing philosophy, and review design nonetheless serve as a direct blueprint for Rentify.

2.2 OLX India and Quikr: Classifieds Platforms

OLX India and Quikr are the dominant digital classifieds platforms in the Indian market. Both platforms host sections for equipment rentals alongside their primary focus on secondhand goods and real estate. An analytical examination of these platforms from the perspective of SME asset rental reveals critical structural deficiencies. Neither platform provides an integrated booking system, payment gateway, availability calendar, deposit management mechanism, or booking

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154262

status tracking. All rental arrangements are conducted entirely offline following an initial online introduction, exposing both parties to substantial trust, legal, and financial risks. The absence of identity verification, formal agreements, and dispute resolution mechanisms makes these platforms fundamentally incompatible with the requirements of systematic SME asset monetisation. Their primary relevance to this research is as negative reference points — they represent the current inadequate state of SME rental facilitation in India that Rentify is designed to supersede.

2.3 Furlenco: Subscription-Based Furniture Rental

Furlenco, an Indian startup founded in 2012, represents a more sophisticated approach to product rental in the Indian market. The platform offers subscription-based furniture and appliance rental with professional delivery, assembly, maintenance, and replacement services. Furlenco demonstrates that Indian consumers are willing to rent rather than purchase even high-value domestic goods when the rental experience is frictionless, the quality is reliable, and the economics are clearly favourable over ownership. However, Furlenco operates exclusively as a vertically integrated business-to-consumer platform — Furlenco itself is the sole asset owner. There is no marketplace mechanism for third-party SMEs or individuals to list their own assets on the platform. Its design model is therefore architecturally irrelevant as a reference for SME dormant asset monetisation, though its user experience design and subscription pricing model provide useful interface reference points.

2.4 Zoom Car: Peer-to-Peer Vehicle Rental

Zoom Car represents India's most mature peer-to-peer rental marketplace and demonstrates that the structured digital rental model is both technically viable and commercially sustainable in the Indian market. The platform incorporates vehicle verification through Registration Certificate documents, insurance integration, GPS-based tracking and geofencing, dynamic availability calendars, in-app payment with multiple methods including UPI, and a rating and review system. However, Zoom Car's verification framework, insurance model, and logistics infrastructure are designed exclusively for automobiles and cannot be extended to the diverse asset types — cameras, drilling equipment, event furniture, gym machines, specialised office equipment — that characterise SME dormant asset inventories. Its significance for this research lies in its proof of concept that structured peer-to-peer rental works in the Indian regulatory and technological environment.

2.5 Fat Llama and KitSplit: International Multi-Category Rental

Fat Llama in the United Kingdom and KitSplit in the United States are the closest international analogues to the Rentify platform. Fat Llama allows users to rent any physical item — cameras, drones, projectors, musical instruments, camping gear, tools — from other users within a geographic proximity. KitSplit specialises in professional photography and film equipment. Both platforms demonstrate the viability of a horizontal, multi-category peer-to-peer rental marketplace. However, neither platform is designed for the specific context of Indian SMEs. They lack support for UPI and Razorpay (10)payment methods, do not address the informal business structures prevalent among Indian SMEs, provide no role-based multi-user access controls suitable for business operations, and are not optimised for the device and bandwidth constraints of tier-two and tier-three Indian markets.

2.6 Identified Research Gaps

The review of existing systems identifies four primary gaps that Rentify is designed to address. The first is SME neglect: no existing platform provides the formal transaction documentation, multi-user role-based access, multi-category asset management, and detailed financial reporting that SME business operations require. The second is the absence of multi-category solutions: all reviewed specialised platforms are vertically constrained to a single asset category, while SMEs own heterogeneous asset portfolios spanning twelve or more categories. The third is the disconnect between asset identification and rental facilitation: no existing platform guides SME owners through the process of identifying, valuing, and listing dormant assets with appropriate pricing, availability, and risk mitigation. The fourth is the absence of transparent earning models: existing platforms obscure commission structures and provide no tools for vendors to forecast their net earnings from a given rental transaction before listing.

3. METHODOLOGY

3.1 System Architecture

Rentify is implemented as a three-tier web application following the classic presentation-logic-data separation pattern. The presentation tier consists of the HTML5/CSS3/JavaScript frontend, which is delivered to and executed entirely within the user's web browser. The logic tier consists of the Python Flask backend, which runs as a Web Server Gateway Interface (WSGI) application on the server and is responsible for all business logic, authentication, data validation, and database interaction. The data tier consists of the MySQL 8.0 relational database server, which persists all platform data with referential integrity enforced through foreign key constraints. Communication between the presentation tier and the

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154262

logic tier is implemented exclusively through a RESTful API. The frontend uses the browser's native Fetch API to send asynchronous HTTP requests — using GET, POST, PUT, and DELETE methods as appropriate — to the backend endpoints. All request and response payloads are formatted as JSON objects with a standardised structure: a status field, a message field, and a data field.

3.2 Working Process Step by Step

The primary operational flow of the platform proceeds as follows. A new user visits the Rentify web application and registers by providing name, email, phone, password, and role selection (renter or vendor). The backend hashes the password using bcrypt with a salt factor of 12, inserts the user record into the users table, generates a JSON Web Token (JWT), and returns it to the frontend. The JWT is stored in the browser's localStorage and is attached as a Bearer token to the Authorization header of every subsequent API request.

A vendor user navigates to the vendor dashboard and creates a product listing by completing a structured form that captures the asset name, category, description, daily price, optional hourly and weekly prices, security deposit amount, stock quantity, location, and delivery option. The backend verifies the vendor's JWT and role, retrieves the vendor record from the vendors table using the JWT's user ID, and inserts a new product record with the vendor's ID as a foreign key. The product becomes immediately visible to renters in the browse catalogue.

A renter selects an asset and specifies a rental date range. The frontend's pricing engine calculates the total rental cost in real time based on the selected duration type, the number of days, and the applicable unit price. The renter may apply a coupon code, which triggers a validation call to the backend; the backend verifies the coupon's validity, usage limit, and minimum transaction threshold, then applies the discount. Upon confirming the booking, the frontend creates a booking record through the /api/bookings endpoint and then initiates the payment flow through /api/payments/create and /api/payments/verify. The backend updates the booking status, decrements the product stock, creates a transaction record in the audit trail, and dispatches an in-platform notification to the renter.

3.3 Algorithms and Logic

The booking price calculation algorithm implements the following logic. The duration in days is computed as the ceiling of the difference between the end date and start date in milliseconds divided by 86,400,000 (the number of milliseconds in one day), with a minimum value of one. If the selected duration type is 'week' and the product has a weekly price defined and the duration spans at least seven days, the base amount is calculated as the weekly price multiplied by the integer quotient of days divided by seven. Otherwise, the base amount is calculated as the daily price multiplied by the total number of days. The coupon discount is then applied — either as a percentage of the base amount or as a fixed deduction, depending on the coupon type — and the security deposit is added to arrive at the total payable amount.

The JWT authentication algorithm generates tokens using the HMAC SHA-256 signing algorithm with a server-side secret key. Each token encodes the user's numeric ID, email address, and role string as claims, along with an expiry timestamp set to 24 hours from the time of issuance. The auth_required decorator on protected Flask routes decodes the token from the Authorization header, verifies the signature against the server secret, checks that the expiry timestamp has not passed, and makes the decoded claims available to the route handler. If any verification step fails, the decorator returns a 401 Unauthorized response before the route handler is invoked.

3.4 DFD and ER Diagram Descriptions

The Data Flow Diagram (DFD) for the Rentify platform at Level 0 — the context diagram — shows three external entities: Renter, Vendor, and Administrator. The Renter interacts with the system by browsing products, submitting booking requests, making payments, and retrieving booking and transaction history. The Vendor interacts with the system by managing product listings, reviewing and approving booking requests, and retrieving earnings data. The Administrator interacts with the system by approving vendor accounts, moderating product listings, managing coupons, and accessing platform-wide analytics. All three entities communicate with the central Rentify system through the REST API boundary.

At Level 1, the DFD decomposes the central system into six primary processes: User Authentication (handling registration and login), Product Management (handling listing creation, editing, and catalogue browsing), Booking Management (handling booking creation, status transitions, and retrieval), Payment Processing (handling order creation, payment verification, and refund processing), Transaction Recording (creating and retrieving the financial audit trail), and Notification Dispatch (creating and delivering in-platform notifications to relevant users). Data stores include the Users table, Vendors table, Products table, Bookings table, Payments table, Transactions table, and supporting tables for reviews, wishlist, cart, coupons, and notifications.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154262

The Entity-Relationship (ER) diagram of the Rentify database defines the following primary entities and relationships. The User entity has a one-to-one optional relationship with the Vendor entity — every Vendor is a User, but not every User is a Vendor. The Vendor entity has a one-to-many relationship with the Product entity — a single Vendor may list multiple Products. The Product entity has a many-to-one relationship with the Category entity — each Product belongs to one Category. The User entity has a one-to-many relationship with the Booking entity — a User may create multiple Bookings. The Product entity has a one-to-many relationship with the Booking entity — a Product may be booked multiple times. The Booking entity has a one-to-many relationship with the Payment entity — a Booking may have multiple associated Payments (for partial payment scenarios). The Payment entity has a one-to-many relationship with the Transaction entity, providing the financial audit trail. Additional relationships include User-to-Wishlist (many-to-many through a junction table), User-to-Cart (one-to-many), Product-to-Review (one-to-many), and User-to-Notification (one-to-many).

4. IMPLEMENTATION

4.1 Authentication Module

The authentication module handles user registration, login, and token-based session management for all three user roles. On the registration endpoint, the backend receives the user's credentials, validates all required fields server-side, checks the database for a duplicate email address using a parameterised SELECT query, and, if the address is unique, hashes the password using the bcrypt library with an automatically generated random salt and a work factor of 12. The hashed password and all other user details are then inserted into the users table using a parameterised INSERT query. If the registering user selected the vendor role, an additional record is inserted into the vendors table with the is_approved field set to FALSE, requiring administrator approval before the vendor can list assets. A JWT is generated and returned to the frontend, which stores it in localStorage for use in subsequent requests.

```
# Flask login endpoint — parameterised queries prevent SQL injection
@app.route('/api/auth/login', methods=['POST'])
def login():
    d   = request.json
    db  = get_db(); cur = db.cursor()
    cur.execute(
        'SELECT id, name, email, password, role FROM users WHERE email = %s',
        (d.get('email'),))
    r = cur.fetchone()
    if not r: return err('Invalid credentials', 401)
    u = dict(zip(['id', 'name', 'email', 'password', 'role'], r))
    if not bcrypt.checkpw(d.get('password','').encode(), u['password'].encode()):
        return err('Invalid credentials', 401)
    u.pop('password')   # never return password hash to client
    token = jwt.encode({
        'id': u['id'], 'email': u['email'], 'role': u['role'],
        'exp': datetime.utcnow() + timedelta(hours=24)
    }, JWT_SECRET)
    return ok({'token': token, 'user': u})
```

4.2 Product Catalogue and Browse Module

The product catalogue module exposes a GET /api/products endpoint that supports a comprehensive set of optional filter parameters: a keyword search string matched against product names using a SQL LIKE clause, a category slug filter applied through a JOIN on the categories table, minimum and maximum daily price bounds, a sort order selector supporting newest-first, price ascending, price descending, and highest-rated options, and pagination parameters for page number and items per page. The SQL query dynamically assembles the WHERE clause and ORDER BY clause based on the supplied parameters, with all user-supplied values passed as parameterised query arguments to prevent SQL injection. The query joins four tables — products, categories, vendors, and users — to assemble the complete product card data required by the frontend in a single database round trip.

On the frontend, the browse page renders product cards dynamically by mapping the API response array through a JavaScript template function that generates the card HTML for each product. Filter changes trigger a re-invocation of the

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154262

API call with updated parameters, and the grid is re-rendered without a full page reload, providing a responsive single-page application experience.

4.3 Booking and Pricing Module

The booking module implements the complete rental transaction lifecycle from price calculation through booking confirmation. The frontend pricing engine performs real-time price calculation in the browser as the user adjusts rental dates and duration type, eliminating the need for a server round trip on every change. The calculation logic selects the appropriate pricing tier, applies coupon discounts after backend validation, adds the security deposit, and renders the complete itemised price breakdown before the user confirms the booking.

```
// Frontend booking price calculator
function calcPrice() {
  const p     = currentProduct;
  const start = document.getElementById('bkS').value;
  const end   = document.getElementById('bkE').value;
  const days  = Math.max(1,
    Math.ceil((new Date(end) - new Date(start)) / 86400000));
  let base;
  if (durType === 'week' && p.price_per_week && days >= 7)
    base = parseFloat(p.price_per_week) * Math.floor(days / 7);
  else
    base = parseFloat(p.price_per_day) * days;
  const discount = couponData
    ? (couponData.discount_type === 'percent'
      ? base * couponData.discount_value / 100
      : couponData.discount_value) : 0;
  const deposit = parseFloat(p.security_deposit || 0);
  const total   = base - discount + deposit;
  bookingData   = { product_id: p.id, start_date: start,
                end_date: end, base, discount, deposit, total };
  renderSummary();   // updates DOM with itemised breakdown
}
```

4.4 Vendor Dashboard Module

The vendor dashboard provides authenticated vendor users with a comprehensive interface for listing management, booking request review, and earnings analytics. The dashboard's primary statistics panel displays total earnings accumulated across all completed bookings, the count of active product listings, the total number of bookings received, and the number of pending booking requests awaiting the vendor's approval or rejection. These figures are computed by the backend through aggregation queries on the bookings, payments, and products tables, joined on the vendor's ID. The top-performing listings table ranks the vendor's products by total bookings received and cumulative revenue generated, enabling data-driven decisions about pricing and availability management.

4.5 Administrator Panel Module

The administrator panel provides the platform's highest privilege level with global read and write access across all entities. Key administrative functions include: approving or revoking vendor accounts (updating the is_approved field in the vendors table), browsing and moderating product listings, viewing all bookings and transactions across the entire platform, creating and deactivating discount coupons, and accessing aggregated analytics including total registered users, active vendors, listed products, and cumulative platform revenue. The admin panel's analytics are computed through a set of aggregate queries that run at request time and are not cached in the current prototype implementation.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154262

4.6 Database Schema: Key Tables

Table	Primary Key	Key Foreign Keys / Constraints
users	id (AUTO_INCREMENT)	UNIQUE: email; ENUM: role
vendors	id (AUTO_INCREMENT)	UNIQUE FK: user_id → users.id ON DELETE CASCADE
categories	id (AUTO_INCREMENT)	UNIQUE: slug
products	id (AUTO_INCREMENT)	FK: vendor_id → vendors.id; FK: category_id → categories.id
bookings	id (AUTO_INCREMENT)	FK: user_id → users.id; FK: product_id → products.id; ENUM: status
payments	id (AUTO_INCREMENT)	FK: booking_id → bookings.id; FK: user_id → users.id
transactions	transaction_id	FK: user_id, booking_id, payment_id (all ON DELETE SET NULL)
reviews	id (AUTO_INCREMENT)	UNIQUE KEY (product_id, user_id); CHECK: rating BETWEEN 1 AND 5
coupons	id (AUTO_INCREMENT)	UNIQUE: code; ENUM: discount_type

Table 4.1: Rentify Database Schema — Key Tables, Primary Keys, and Referential Constraints

5. RESULTS AND DISCUSSION

5.1 Outputs Achieved

The Rentify platform was subjected to systematic testing across four dimensions: functional correctness, API response performance, database query performance, and cross-browser compatibility. Functional testing was conducted across all three user roles, covering twenty-eight user workflows, fifteen vendor workflows, and twelve administrator workflows — a total of fifty-five test cases. All fifty-five test cases passed without error on the local development environment running against a MySQL database seeded with 500 products across twelve categories, 342 simulated users, and 200 transaction records.

API response performance was measured by issuing one hundred successive HTTP GET requests to the /api/products endpoint and recording the response time for each request using a lightweight Python benchmarking script. The mean response time was 42 milliseconds, with a maximum observed response time of 118 milliseconds occurring during the first request after a cold start due to MySQL connection pool initialisation. All subsequent requests fell within the 20-to-80-millisecond range, well within the 200-millisecond threshold generally accepted as imperceptible to users in interactive web applications.

Metric	Mean	Maximum	Pass Threshold
GET /api/products (no filter)	42 ms	118 ms	< 200 ms
GET /api/products (filtered, sorted)	38 ms	95 ms	< 200 ms
POST /api/auth/login	28 ms	67 ms	< 200 ms
POST /api/bookings (with coupon)	55 ms	132 ms	< 300 ms
GET /api/transactions (200 records)	22 ms	58 ms	< 200 ms
Database: product listing query (JOIN x4)	< 15 ms	—	< 100 ms
Database: transaction history query (JOIN x5)	< 22 ms	—	< 100 ms
Usability: user task completion time	3 min 47 s	5 min	< 10 min

Table 5.1: Performance Benchmark Results — API Response Times, Database Query Times, and Usability

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154262

Usability testing was conducted with five student volunteers from the same academic programme. Each participant completed a standardised task set — registration, product browsing with category filter, coupon application, booking creation, and transaction history review — without assistance from the development team. All five participants completed the full task set successfully. The mean completion time was 3 minutes and 47 seconds, well within the 10-minute ceiling established as the usability acceptance criterion. The most frequently reported observation was that the booking page date picker required more explicit labelling, which was addressed in the final implementation iteration.

5.2 Advantages of the System

Rentify offers several distinct advantages over existing solutions. First, it is the only platform reviewed that simultaneously addresses the full combination of SME requirements: multi-category asset support, role-based multi-user access, structured booking and payment workflows, transparent earnings reporting, and a comprehensive administrative control panel. Second, the platform's technology stack — Python Flask, MySQL, and vanilla JavaScript — is entirely open-source, requiring no licensed software components and keeping infrastructure costs minimal for both the platform operator and the SME vendors who adopt it. Third, the JWT-based stateless authentication architecture scales horizontally without requiring shared session storage, making the platform ready for load-balanced multi-instance production deployments. Fourth, the thirteen-table normalised database schema ensures complete data integrity throughout the transaction lifecycle, providing the audit trail that SME businesses require for financial management and tax compliance. Fifth, the simulated Razorpay (10)payment flow is structurally identical to a production Razorpay(10) integration, requiring only the substitution of the simulated verification call with actual Razorpay(10) signature verification to be production-ready — significantly reducing the effort required for commercial deployment.

5.3 Limitations

The current implementation carries three principal limitations. The first and most significant is the use of a simulated payment gateway. No real financial transactions are processed in the prototype; the Razorpay (10) order IDs and payment IDs are generated as placeholder strings. While the code architecture is explicitly designed to minimise the effort required for live Razorpay(10) integration, the absence of real payment settlement means the platform cannot be commercially deployed in its current state without this integration. The second limitation is the use of emoji characters as product images rather than actual uploaded image files. This approach is technically convenient for a prototype and provides visual differentiation between listings, but a production platform requires an image upload and storage capability backed by a cloud object storage service such as AWS S3, Cloudflare R2, or Google Cloud Storage, along with server-side image resizing and content delivery network distribution. The third limitation is the absence of a mobile native application. Research confirms that over 67 percent of Indian SME owners access digital services primarily through mobile devices; a responsive web application, while functional on mobile browsers, does not provide the same user experience as a native Android or iOS application in terms of notification delivery, offline capability, and camera integration for listing photographs.

6. CONCLUSION

6.1 Summary of Work

This paper has presented Rentify, a full-stack web-based rental marketplace designed specifically to address the dormant asset problem faced by small and medium-sized enterprises in India. The platform implements a three-tier architecture comprising a vanilla JavaScript single-page application frontend, a Python Flask RESTful API backend, and a MySQL relational database with a thirteen-table normalised schema. The system supports three distinct user roles — renters, vendors, and administrators — each with dedicated dashboards, access controls enforced through JSON Web Token authentication, and precisely defined operational workflows.

The platform's booking engine supports hourly, daily, and weekly pricing models with automatic price calculation, coupon discount application, and security deposit management. The vendor analytics dashboard provides SME owners with the earnings visibility, booking management tools, and listing performance data that are absent from all existing rental platforms reviewed in the literature survey. Systematic testing across fifty-five functional test cases, complemented by API performance benchmarking and cross-browser compatibility verification, confirms that the platform delivers correct, performant, and usable functionality across its complete feature set.

The four research gaps identified in the literature review — SME neglect, absence of multi-category solutions, poor identification-to-rental integration, and lack of transparent earning models — are directly addressed by Rentify's design and confirmed to be resolved by the functional testing outcomes. The platform thus represents a meaningful technical contribution to the field of SME digital commerce and the Indian sharing economy.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154262

6.2 Future Scope

Several high-priority enhancements are identified for future development. The development of native Android and iOS mobile applications using the React Native framework is the most impactful next step, given the mobile-primary digital access patterns of Indian SME owners. The existing Flask REST API requires no modification to serve a React Native frontend, making this a relatively contained implementation effort with substantial impact on platform adoption.

The integration of a machine learning-based dynamic pricing recommendation engine represents the most technically ambitious enhancement. A model trained on the platform's transaction history, incorporating seasonal demand patterns, local event calendars, and comparable listing prices, would enable Rentify to recommend optimal rental rates to vendors — systematically eliminating the underpricing and overpricing that result from information asymmetry in the current manual pricing model. The thirteen-table database schema is specifically designed to capture the transaction and behavioural data required to train and refine such a model.

Live Razorpay(10) payment gateway integration, blockchain-based asset ownership verification for high-value rental transactions, integration with the Government of India's Udyam SME registration portal, and predictive demand forecasting analytics represent additional directions for future development that would progressively transform the prototype into a commercially deployable platform capable of meaningful impact on SME asset utilisation and passive income generation across India.

REFERENCES

[1]. G. Zervas, D. Proserpio, and J. W. Byers, "The Rise of the Sharing Economy: Estimating the Impact of Airbnb on the Hotel Industry," Journal of Marketing Research, vol. 54, no. 5, pp. 687–705, Oct. 2017. doi: 10.1509/jmr.15.0204.
[2]. R. Belk, "You Are What You Can Access: Sharing and Collaborative Consumption Online," Journal of Business Research, vol. 67, no. 8, pp. 1595–1600, Aug. 2014. doi: 10.1016/j.jbusres.2013.10.001.
[3]. J. Hamari, M. Sjöklint, and A. Ukkonen, "The Sharing Economy: Why People Participate in Collaborative Consumption," Journal of the Association for Information Science and Technology, vol. 67, no. 9, pp. 2047–2059, Sep. 2016. doi: 10.1002/asi.23552.
[4]. A. Sundararajan, The Sharing Economy: The End of Employment and the Rise of Crowd-Based Capitalism. Cambridge, MA: MIT Press, 2016. ISBN: 9780262034579.
[5]. Ministry of Micro, Small and Medium Enterprises, Government of India, "Annual Report 2022–23," New Delhi, India, 2023. [Online]. Available: https://msme.gov.in.
[6]. Internet and Mobile Associatio Dr. S R Deoren of India (IAMAI), "India Internet Report 2023: Mobile Internet Usage Trends Among SMEs," Mumbai, India, 2023. [Online]. Available: https://www.iamai.in.
[7]. Pallets Projects, "Flask — A Lightweight WSGI Web Application Framework," Official Documentation, version 3.0, 2024. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/.
[8]. Oracle Corporation, "MySQL 8.0 Reference Manual," 2024. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/.
[9]. M. B. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force (IETF) RFC 7519, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519.
[10]. Razorpay Software Private Limited, "Razorpay Payment Gateway API Integration Documentation," Bengaluru, India, 2024. [Online]. Available: https://razorpay.com/docs/.
[11]. [11] N. Provos and D. Mazieres, "A Future-Adaptable Password Scheme," in Proceedings of the USENIX Annual Technical Conference, Monterey, CA, 1999, pp. 81–91.
[12]. T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," Internet Engineering Task Force (IETF) RFC 3986, Jan. 2005. [Online]. Available: https://tools.ietf.org/html/rfc3986.

