Street Vendor Digital Platform: A Hyperlocal Ordering System

Mr. Maheswar Malik (2201298102)
Student, Dept. of CSE, GIFT Autonomous, Bhubaneswar

Mr. Karanjeet Debasish Beura (2201298096)
Student, Dept. of CSE, GIFT Autonomous, Bhubaneswar

Lecturer Subhalaxmi Nayak
Asst. Professor, Dept. of CSE, GIFT Autonomous, Bhubaneswar

***Abstract*—The rapid growth of digital technology has significantly transformed various sectors, including commerce, transportation, and food delivery. However, street vendors—especially in Tier-2 and Tier-3 cities—still rely on traditional methods of selling, limiting their reach and efficiency. This paper presents the development of a Street Vendor Digital Platform, a hyperlocal web application designed to connect nearby users with street vendors such as chai sellers, pani puri vendors, and tiffin service providers.**

The platform enables users to discover nearby vendors using location-based services, browse menus, and place orders efficiently. A key challenge in cash-on-delivery systems is the prevalence of fake orders, which negatively impacts vendors. To address this, the system introduces an OTP-based pickup verification mechanism and a trust scoring system that tracks user behavior.

The system is developed using React.js for the frontend, Tailwind CSS for styling, and FastAPI (Python) for backend development. SQLite serves as the database for storing user, vendor, and order data. OCR-based AI integration is used for extracting menu items from images. The proposed solution enhances vendor visibility, improves customer convenience, and ensures secure transactions. The system is scalable, user-friendly, and designed for real-world impact in local communities.

***Keywords*—Hyperlocal Platform, Street Vendor, OTP Verification, Trust Scoring, React.js, FastAPI, SQLite, OCR, Location-Based Services, Cash-on-Delivery**

I. INTRODUCTION

The rapid advancement of digital technology has profoundly transformed how people access services in everyday life. Web-based applications have become essential tools for delivering real-time information, improving accessibility, and enhancing user experience across various domains, including food delivery, tourism, and public services.

Street vendors—including chai sellers, pani puri vendors, and local tiffin service providers—form a significant part of the food ecosystem, especially in Tier-2 and Tier-3 cities. Despite providing affordable, culturally rich food options deeply embedded in daily life, these vendors face critical challenges: lack of digital presence, limited customer reach, and absence of structured order management. Meanwhile, modern consumers increasingly prefer digital solutions for convenience.

This paper introduces a Street Vendor Digital Platform, a hyperlocal web-based application designed to bridge the gap between offline vendors and digitally connected users. The system leverages location-based discovery, simple onboarding processes, and secure ordering mechanisms to create seamless interaction between users and vendors.

The primary motivation is to create an inclusive digital ecosystem that benefits both consumers and small-scale vendors. By digitizing street vendors, the platform contributes to financial inclusion, increases vendor income, and promotes local businesses. From a technical perspective, the project integrates modern web technologies, location-based services, and AI-based features into a practical real-world application.

The remainder of this paper is organized as follows: Section II presents a literature survey. Section III describes the proposed system architecture. Section IV details the technology stack. Section V covers implementation. Section VI presents testing and results. Section VII discusses findings and limitations. Section VIII concludes with future work directions.

II. LITERATURE SURVEY

A. Overview of Existing Systems

Online food delivery platforms such as Zomato and Swiggy have redefined food ordering by offering convenience, real-time tracking, and a wide variety of choices. However, their business models are primarily designed for registered restaurants with high commission fees, making it difficult for small-scale street vendors to participate [1][2].

Some hyperlocal service applications focus on connecting users with nearby services using GPS-based location tracking. While they support local businesses to some extent, they often lack features tailored specifically for street vendors, such as simple onboarding, offline-friendly operations, and minimal cost structures [3][4].

In many regions, street vendors rely on informal methods such as phone calls or WhatsApp for taking orders. Although easy to use, these methods lack structure, scalability, and reliability—with no centralized system for order management or secure transactions [5].

B. AI in Hyperlocal Systems

Artificial Intelligence has emerged as a powerful tool in food delivery platforms for recommendation systems, demand prediction, route optimization, and customer behavior analysis [6]. However, implementing complex AI models in small-scale systems is often impractical. Lightweight techniques such as OCR (Optical Character Recognition) and rule-based trust systems can provide AI benefits without heavy infrastructure requirements [7].

OCR tools like Tesseract allow vendors to digitize menus from images, reducing manual effort. Rule-based trust systems analyze user actions and assign scores accordingly, effectively mimicking intelligent behavior without requiring machine learning models.

C. Trust-Based Systems

Trust is a critical factor in digital transactions, especially in Cash-on-Delivery (COD) environments. Many platforms face challenges related to fake orders and unreliable users. OTP verification ensures only genuine users complete transactions, while trust scoring systems track behavior and prevent misuse [8][9].

D. Feature Comparison and Research Gap

Table I summarizes a feature comparison of existing systems against the proposed solution. The review reveals key gaps: no platforms combining hyperlocal discovery with trust-based ordering; absence of OTP verification in small-scale platforms; limited lightweight AI solutions for vendor onboarding; and no dedicated systems focusing on Tier-2 and Tier-3 cities.

Feature	Zomato/Swiggy	Hyperlocal Apps	WhatsApp/Phone	Proposed System
Street Vendor Support	No	Partial	Informal	Yes
Location-Based Discovery	Yes	Yes	No	Yes
OTP Order Verification	No	No	No	Yes
Trust Scoring System	No	No	No	Yes
AI Menu Extraction	No	No	No	Yes (OCR)
Free / Low Cost	No (High Fee)	Moderate	Yes	Yes
COD Security	No	No	None	Yes
Vendor Dashboard	Limited	Partial	No	Yes

Table I: Feature Comparison of Existing Systems

III. PROPOSED SYSTEM

The Street Vendor Digital Platform is a web-based application designed to connect nearby users with street vendors in a simple, secure, and efficient manner. Unlike traditional food delivery platforms, this system is specifically designed for small-scale vendors such as chai sellers, pani puri vendors, and tiffin service providers.

The system allows users to discover vendors based on their current GPS location, view available menu items, and place orders using a COD model. To address the major issue of fake orders, the platform incorporates a Pickup OTP verification system. Additionally, a trust scoring mechanism tracks user behavior: completed orders award +1 point while missed orders deduct 2 points; users scoring below −3 are temporarily blocked.

A. System Architecture

The system follows a three-tier architecture ensuring modularity, scalability, and efficient data management:

- Presentation Layer (Frontend): Developed using React.js with Tailwind CSS for a modern, responsive user experience. Manages vendor listings, user inputs, cart state, and API communication.
- Application Layer (Backend): Implemented using FastAPI (Python), handling all business logic including JWT authentication, order workflows, OTP generation and verification, and trust scoring computation.
- Data Layer: SQLite with SQLAlchemy ORM for lightweight, efficient relational data storage of users, vendors, menus, orders, and reviews.

B. Database Design

The relational schema comprises five normalized entities as summarized in Table IV. Foreign key constraints maintain referential integrity throughout; cascade deletes ensure consistency when vendors or users are removed.

Entity	Key Fields	Relationships
Users	id, name, phone, role, trust_score	Places many Orders
Vendors	id, user_id, shop_name, location, is_online	Has many MenuItems
MenuItems	id, vendor_id, name, price	Belongs to Vendor
Orders	id, user_id, vendor_id, status, total, pickup_otp	Has many OrderItems
OrderItems	id, order_id, item_id, quantity	Belongs to Order

Table IV: Database Schema Summary

C. Key Features

- Location-Based Vendor Discovery: Users find nearby vendors via real-time GPS location.
- Real-Time Vendor Availability: Vendors toggle Online/Offline status dynamically.
- OTP-Based Order Verification: A unique pickup OTP is generated per order and verified by the vendor at collection.
- Trust Scoring System: Rule-based scoring deters fake orders and protects vendors.
- AI-Based Menu Extraction: Vendors upload menu images; Tesseract OCR extracts and digitizes items automatically.
- Light and Dark Mode UI: Theme switching for improved user experience across devices.

IV. TECHNOLOGY STACK

A. Frontend (React.js, Tailwind CSS)

React.js is a component-based JavaScript library for building dynamic and interactive user interfaces. Components (VendorCard, CartPanel, OrderTracker, OTPModal) are reusable and independently testable. State is managed through React Hooks (useState, useEffect), enabling real-time UI updates driven by API responses. Tailwind CSS provides a utility-first styling framework supporting responsive layouts, dark mode theming, and modern design patterns with minimal custom CSS overhead.

B. Backend (FastAPI, Python)

FastAPI is a high-performance, asynchronous Python web framework built on Starlette and Pydantic. Its non-blocking I/O model efficiently handles concurrent API requests—critical during simultaneous order placements. All business logic, authentication, OTP management, trust scoring, and OCR processing is handled in this layer. API responses are structured in JSON format for seamless frontend integration.

C. Database (SQLite + SQLAlchemy)

SQLite provides a lightweight, serverless relational database suitable for initial deployment without requiring infrastructure configuration. SQLAlchemy ORM abstracts database interactions into Python objects, simplifying CRUD operations, enforcing schema relationships, and enabling easy migration to PostgreSQL or MySQL in production. Indexes on foreign key columns (user_id, vendor_id, order_id) optimize query performance.

D. Security Stack

JWT (JSON Web Tokens) are issued on successful authentication, signed with a server-side secret (HS256), and required on all protected API endpoints—eliminating stateful session management. Passwords are hashed using bcrypt before storage.

Parameterized SQL queries via SQLAlchemy prevent SQL injection. Input validation and output encoding protect against XSS attacks. Role-based access control restricts vendor and admin endpoints from regular users.

E. AI Integration (Tesseract OCR)

Tesseract OCR processes uploaded menu images to extract text items. The extracted text is then parsed and structured into menu database records, dramatically reducing manual onboarding effort for vendors with printed or handwritten menus. This lightweight AI approach provides practical benefits without requiring cloud ML infrastructure.

V. IMPLEMENTATION

A. User and Vendor Registration/Login

The registration endpoint accepts POST requests with name, phone number, and role (user/vendor). Server-side validation enforces required fields, valid phone format, and uniqueness. Passwords are bcrypt-hashed before storage. On login, a JWT containing user_id and role is signed and returned. Subsequent requests include this token in the Authorization header for middleware verification.

B. Vendor Discovery and Menu Module

The location-based discovery endpoint accepts latitude and longitude parameters and returns vendors within a configurable radius, ordered by distance. Only vendors with is_online=True are returned to users. Vendors manage their menu through a dedicated dashboard, adding items manually or via OCR image upload. The backend processes uploaded images through Tesseract, returning extracted text as candidate menu items for vendor confirmation.

C. Order Placement and OTP Flow

Order placement accepts a vendor ID and list of item IDs with quantities. The backend computes the total, generates a unique pickup OTP (stored hashed in the Orders table), persists the order with status Pending, and notifies the vendor. The order progresses through states: Pending → Accepted → Ready → Completed. On physical pickup, the user shares the OTP with the vendor; the vendor submits it for backend verification. If valid, the order is marked Completed and the user's trust score incremented by 1.

D. Trust Scoring Module

The trust scoring system operates as a background process triggered on order status changes. A completed order increments the user's trust_score by 1. A missed order (marked by vendor after a timeout) decrements the score by 2. If the score drops below −3, the system sets a temporary block flag, preventing the user from placing new orders until the block is reviewed. This rule-based mechanism effectively deters fraudulent COD behavior without requiring ML models.

E. Frontend Component Architecture

The React frontend is structured into reusable components: Navbar (navigation and auth state), HomeMap (GPS vendor listing), VendorDetail (menu and cart), OrderTracker (real-time status), and OTPModal (pickup verification). State is lifted to parent components where shared across siblings. The useEffect hook manages API data fetching on component mount, with loading and error states handled gracefully for user feedback.

VI. SYSTEM TESTING AND RESULTS

A. Testing Strategy

A multi-level testing strategy was employed throughout development. Unit testing validated individual functions in isolation—bcrypt hashing, JWT issuance, trust score logic, and OTP generation. Integration testing verified correct data flow across boundaries: the order placement pipeline (user input → API → database → OTP → vendor notification), the OCR pipeline (image upload → Tesseract processing → menu creation), and the authentication pipeline (registration → login → protected route access). System testing evaluated the complete application under realistic usage scenarios.

B. Functional Test Results

Table II presents the functional test cases and outcomes. All twelve test cases passed, confirming correct behavior across authentication, vendor discovery, order management, OTP verification, trust scoring, and access control modules.

Test #	Test Case	Expected Result	Status
TC-01	Valid user registration	Account created in DB	Pass
TC-02	Duplicate phone/email	Error: already exists	Pass
TC-03	Valid login	JWT issued, redirect to dashboard	Pass

TC-04	Invalid login credentials	Access denied	Pass
TC-05	Vendor discovery by location	Nearby vendors displayed	Pass
TC-06	Place order (COD)	Order stored, OTP generated	Pass
TC-07	OTP verification – valid	Order marked Completed	Pass
TC-08	OTP verification – invalid	Order unchanged, error shown	Pass
TC-09	Trust score update	Score updated correctly	Pass
TC-10	User blocked (score < -3)	Access restricted	Pass
TC-11	OCR menu image upload	Menu items extracted correctly	Pass
TC-12	Unauthorized admin access	403 Forbidden returned	Pass

Table II: Functional Test Cases and Results

C. Performance Metrics

Performance testing was conducted using browser developer tools and manual timing under simulated concurrent loads of 10–50 users on a local FastAPI server. Table III summarizes observed performance metrics.

Metric	Observed Value	Remarks
API Response Time	< 200 ms	FastAPI async handling
Vendor List Load	< 0.8 sec	Indexed DB query
Order Placement	< 0.5 sec	Single transaction
OTP Verification	< 0.3 sec	Hash comparison + update
OCR Extraction	2–4 sec	Tesseract processing time
UI Load Time	< 1 sec	React component rendering
Concurrent Users	Up to 50	No degradation (local server)

Table III: System Performance Metrics

D. Security Testing

Security testing confirmed: SQL injection attempts on login and search inputs returned sanitized errors with no data leakage, as SQLAlchemy parameterized queries were in effect. XSS payloads submitted in vendor names and menu item fields were escaped on output. Direct API calls to vendor-only endpoints using a regular user JWT returned 403 Forbidden. Expired JWTs were correctly rejected with 401 Unauthorized. All critical security requirements were satisfied.

E. Usability Feedback

User acceptance testing involved a group of 15 participants (12 student volunteers and 3 faculty members) who used the system for mock vendor discovery and ordering. Average ratings on a 5-point Likert scale: Ease of Registration (4.5/5), Vendor Discovery Clarity (4.4/5), OTP Flow Usability (4.6/5), Order Tracking (4.3/5), Overall Satisfaction (4.5/5). Qualitative feedback highlighted location-based discovery and instant OTP verification as standout features.

VII. DISCUSSION AND LIMITATIONS

A. Analysis of Results

The system achieved all primary objectives: location-based vendor digitization, secure OTP-verified COD ordering, trust-based fraud prevention, and AI-assisted onboarding. Functional testing demonstrated a 100% pass rate across all twelve test cases. Performance metrics confirmed sub-second response times for all core operations. Security testing validated protection against common web attack vectors. UAT results confirmed strong usability with average satisfaction scores above 4.3/5 across all categories.

The use of JWT-based stateless authentication proved advantageous for scalability. The normalized database schema effectively prevented data redundancy. The React component architecture enabled rapid UI development with consistent state management throughout the ordering workflow.

B. Challenges Encountered

Several technical challenges were addressed during development. Accurate distance-based vendor filtering required handling edge cases in GeoJSON coordinate comparisons. Managing React component state during order tracking (ensuring real-time status updates without full page reload) required implementing polling with useEffect cleanup. Integrating Tesseract OCR with FastAPI required careful subprocess management and timeout handling for large menu images.

C. Limitations

- Supports only Cash-on-Delivery; no UPI, card, or digital wallet payment integration in current version.
- Trust scoring is rule-based; no machine learning models for behavioral prediction or anomaly detection.
- Testing conducted on local server; production-scale performance under hundreds of concurrent users requires cloud infrastructure and load balancing.
- No real-time GPS delivery tracking; order status is updated manually by the vendor.
- No dedicated mobile application; web interface is responsive but not touch-optimized.

VIII. CONCLUSION & FUTURE WORK

A. Conclusion

The Street Vendor Digital Platform was successfully designed, implemented, and validated as a reliable, secure, and user-friendly web-based platform for hyperlocal street vendor discovery and ordering. The system fulfills all defined objectives: it digitizes small-scale vendors with minimal technical requirements; enables location-based vendor discovery; provides OTP-secured COD ordering; implements trust scoring to deter fake orders; and uses OCR-based AI for simplified menu onboarding.

Testing across functional, security, performance, and usability dimensions confirmed the system's readiness for real-world deployment in Tier-2 and Tier-3 city environments. It represents a practical and effective solution that empowers small-scale vendors, promotes digital inclusion, and improves accessibility for everyday consumers.

B. Future Enhancements

- Online Payment Integration: UPI, debit/credit cards, and mobile wallets for cashless transactions.
- Advanced AI and ML: Recommendation systems, demand prediction for vendors, and ML-based fraud detection using behavioral patterns.
- Real-Time Delivery Tracking: GPS-based order tracking with estimated pickup time and navigation support.
- Mobile Application: Cross-platform app using React Native or Flutter for improved mobile accessibility and push notifications.
- Cloud Deployment: AWS/GCP/Azure deployment with auto-scaling, containerization (Docker/Kubernetes), and distributed database support.
- Admin Dashboard: Super-admin module for managing users, vendors, complaints, and generating analytics reports.
- Advanced Proctoring and Smart Features: Voice-based ordering, AI chatbots for customer support, and IoT-based vendor notifications.

REFERENCES

[1] Chanchala Jain et al., "Online Food Delivery Apps Zomato and Swiggy: A Comparative Study," ResearchGate, 2024.
[2] Mitali Gupta, "Impact of Online Food Delivery Apps on Restaurant Business," International Journal of Research and Analytical Reviews, 2019.
[3] IJRAR Journal, "Impact of Food Delivery Apps like Zomato and Swiggy," IJRAR, 2020.
[4] IJSREM, "Comparative Study of Online Food Delivery Trends in India," IJSREM, 2025.
[5] A. Shankar et al., "Online Food Delivery: A Systematic Review of Literature," ScienceDirect, 2022.
[6] Amity University Research, "Impact of AI on Food Delivery Systems – Case Study on Swiggy," ResearchGate, 2023.
[7] Dev.to, "System Design of Food Delivery Platforms like Swiggy and Zomato," DEV Community, 2026.

[8] ResearchGate, "Algorithmic Management in Food Delivery Platforms," ResearchGate, 2025.
[9] Global Scientific Journal, "Consumer Behaviour Towards Food Delivery Apps," Global Scientific Journal, 2023.