www.ijarp.com ISSN 2456-9992 Page: 01-09
International Journal Advanced Research Publications
Research Article
Volume: 02
Issue: 05

LOCAL VENDOR SERVICE & PRODUCT RECOMMENDATION SYSTEM

[*]Rahul Kumar, Subham, Md Sajjad, Manish Kumar, Prof. Vishal Upmanu

Dept. of CSE, RD Engineering College, Duhai, Ghaziabad, India.

Article Received: 23 March 2026, Article Revised: 13 April 2026, Published on: 03 May 2026

***Corresponding Author: Rahul Kumar**
Dept. of CSE, RD Engineering College, Duhai, Ghaziabad, India.
DOI: https://doi-doi.org/101555/ijarp.5948

ABSTRACT

One of the most common problems people face when they visit a new place is that they cannot find basic nearby services like photocopy shops, stationery stores, or printing centres. Small local vendors are usually not listed on any digital platform, which creates problems for both the user and the vendor. In this paper, we have built a location-based system that helps users find nearby vendors and compare their service prices. The system uses the browser's GPS to get the user's location, a Node.js backend to find nearby vendors from a MongoDB database, and Leaflet.js to show them on an interactive map. The frontend is built using React.js and is designed to be simple and easy to use. The main features of the system include distance-based vendor filtering using the Haversine formula, display of per-service prices, and a price comparison view. During testing, the system was able to find vendors accurately and returned results in under 2 seconds.

KEYWORDS: *Location-Based Services, Vendor Discovery, GPS, Price Comparison, React.js, Node.js, MongoDB, Leaflet.js, Hyperlocal Commerce.*

1. INTRODUCTION

When we first started working on this project, the idea came from a very simple and real problem. Whenever a student joins a new college or moves to an unknown area, they struggle to find basic services like where to get a photocopy done, or where to find a stationery shop. Even if these shops are just 200 metres away, there is no easy way to find them.

Platforms like Google Maps do exist, but they mostly show large businesses. Small local

vendors — the ones running a photocopy stall near a college gate or a small cyber café in a lane — are almost never listed anywhere. And even if they are, there is usually no information about pricing. So users end up going to whichever shop they find first, often paying more than needed.

Our system tries to solve this. Users can open the app, share their location, and instantly see nearby vendors along with the prices they charge for different services. If multiple shops offer the same service, the user can compare prices side by side. For vendors, registering on the platform is simple and does not require any technical knowledge.

We chose React.js for the frontend, Node.js with Express for the backend APIs, MongoDB as the database, and Leaflet.js for the map. These technologies are widely used in the industry and were something our team was comfortable working with. Leaflet.js was preferred over Google Maps API mainly because it is free and open-source.

2. PROBLEM STATEMENT

The main problem this project addresses is quite straightforward. When someone visits a new place, they have no easy way to find small local service vendors nearby. Existing apps either do not list these vendors at all, or even if they do, there is no pricing information available. This forces users to either ask around or visit multiple shops manually just to find the best price.

This situation leads to two clear issues. First, users waste time searching for services that are available nearby but are not visible digitally. Second, because there is no price reference, vendors can
charge whatever they want, and users have no way to know if they are being overcharged. Our system directly addresses both of these problems.

3. OBJECTIVES

The main objectives of this project are listed below:

- To capture the user's real-time GPS location, with a manual pin option as a backup.
- To show vendors available within a certain distance from the user.
- To display the prices of services offered by each vendor clearly.
- To allow users to compare prices of the same service across different vendors.
- To let vendors register themselves and list their services easily.
- To keep the interface simple enough that any user can understand it without help.

4. RELATED WORK

4.1 Location-Based Recommendation Systems

Bao and Zheng [1] worked on recommending places to users based on their location history and social data. Their research showed that proximity is one of the strongest factors in deciding whether a user will visit a place. However, their approach requires the user to have prior location history, which is not suitable in our case since we are targeting first-time visitors who have no history on the platform.

4.2 Hyperlocal Commerce

Singla and Sachdeva [2] described how hyperlocal e-commerce systems work, where orders are routed based on how close the merchant is to the customer. We borrowed this proximity-first idea and applied it to service discovery rather than product delivery.

4.3 Haversine Distance Formula

The Haversine formula is a well-known method to calculate the straight-line distance between two GPS points on the Earth's
surface [3]. It is accurate enough for short distances like within a city or campus, and is not computationally expensive, which makes it a good fit for our use case.

4.4 Price Transparency

Jensen [4] studied how giving fishermen in Kerala access to market price data through mobile phones helped reduce unfair pricing. The core finding was that when buyers have access to price information, sellers cannot easily overcharge them. This is exactly the problem we are trying to solve in a local services context.

4.5 Leaflet.js

Leaflet.js is a free, open-source JavaScript library for interactive maps [5]. It works with OpenStreetMap tiles and does not need an API key. Many student projects and startups use it as an alternative to Google Maps. We found it easy to integrate and sufficient for our requirements.

5. METHODOLOGY

We divided the development into two phases. Phase 1, which is what this paper covers, includes the core features for both customers and vendors. Phase 2 is planned for the future and will include an admin panel and possibly an AI recommendation feature.

We followed a bottom-up approach while building — first the database schema, then the backend APIs, and finally the frontend. We did it this way because if the database design is wrong, fixing it later requires changing a lot of other things too. Getting the schema

right first saved us quite a bit of time during later development.

Location handling turned out to be more tricky than we initially expected. Sometimes the browser's GPS gives a stale or inaccurate reading, especially indoors. We handled this by adding a manual map-pin option where users can drag a pin to their actual location if GPS is not working correctly.

6. SYSTEM ARCHITECTURE AND DESIGN

The overall system is divided into three layers — the frontend (React.js), the backend (Node.js + Express), and the database (MongoDB). The client sends requests to the server using REST APIs, the server processes them and queries MongoDB, and then sends back the results in JSON format.

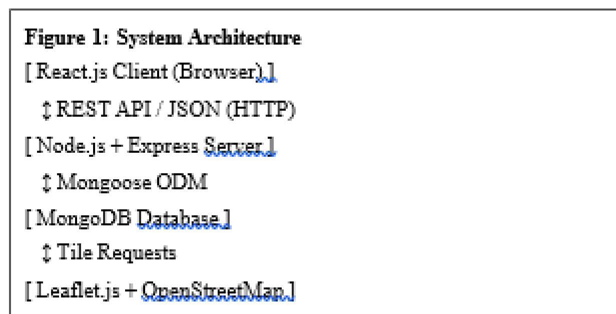

Figure 1: System Architecture Diagram.

6.1 Frontend — React.js

The React.js frontend has two main sections. The customer side shows a Leaflet.js map with vendor markers and a list of nearby vendors as cards, each showing the vendor's name, distance, and prices. The vendor side has a registration form where shop owners can enter their details and add the services they offer. We used React hooks like useState and useEffect to manage the component state, which kept things clean without needing a separate state management library.

6.2 Backend — Node.js + Express

The backend handles four main API endpoints. GET/api/vendors/nearby takes the user's latitude, longitude, and search radius, and returns a list of nearby vendors with their distances. GET /api/vendors/:id/services returns all services and prices for a specific vendor. POST /api/vendors/register is used when a new vendor signs up. POST /api/users/location stores the user's location on the server side.

6.3 Database — MongoDB

We used MongoDB instead of a relational database like MySQL because the service data for each vendor is not fixed in structure. One vendor might offer 3 services, another might offer 10, and the types of services are completely different. A document-based model handles this more naturally. We have three main collections: users, vendors, and services. Vendor GPS coordinates are stored as GeoJSON, which will allow us to use MongoDB's built-in geospatial queries in future versions.

6.4 Map — Leaflet.js

Leaflet.js renders the map using free OpenStreetMap tiles. Each vendor is shown as a marker on the map. When a user clicks a marker, a small popup appears showing the vendor's name, distance, and a button to view their full service list. The user's own location is shown with a different coloured marker. If GPS fails, the user sees a draggable pin they can move to their actual position manually.

7. IMPLEMENTATION DETAILS

7.1 Location and GPS Fallback

Location is captured using the browser's built-in navigator.geolocation API. One thing we learnt during testing is that passing maximumAge: 0 is important — without it, the browser sometimes returns a location from the previous session, which gives wrong distances. If the user denies location permission, we show a Leaflet.js map with a draggable pin. The user places the pin at their location and confirms it, and those coordinates are sent to the server.

7.2 Haversine Distance Calculation

We calculate distances between the user and each vendor on the server side using the Haversine formula:

$$d = 2R \times \arcsin\left(\sqrt{\sin^2(\Delta lat/2) + \cos(lat_1) \cdot \cos(lat_2) \cdot \sin^2(\Delta lon/2)}\right)$$

where R = 6,371 km (Earth's mean radius)

The result is returned in kilometres up to two decimal places. We tested the output by comparing it to actual distances on a campus. The values were about 5–10% shorter than real walking distances, which is expected since walking paths are not perfectly straight. But the ranking — which vendor is closest — was always correct.

7.3System Algorithm

1. Get user GPS coordinates (or manual pin).
2. Send coordinates + radius to backend.
3. Fetch all active vendors from MongoDB.
4. Calculate Haversine distance for each vendor.
5. Keep only vendors within the given radius.
6. Sort vendors by distance, nearest first.
7. Add service and price data to each vendor.
8. Send the sorted list back to React frontend.
9. Show vendors as map markers and list cards.
10. On service search: filter by type, show price comparison.

Figure 2: Vendor Discovery Algorithm.

7.4Price Comparison

This is probably the most useful feature of the whole system. When a user types in a service name like "photocopy" or "printout", the backend filters all vendors that offer that service and returns them sorted by distance. The frontend then shows them as cards with price and distance side by side. This way, even if one vendor is slightly farther, the user can see if it is worth going there for a cheaper price. This kind of comparison used to require visiting multiple shops physically, which our system eliminates.

7.5Vendor Registration

The vendor registration form collects the business name, owner name, phone number, and location. For location, vendors go through the same GPS flow as customers. Services are added through a dynamic form where each row has a service name, price, and unit (like "per page" or "per hour"). After submitting, the vendor immediately shows up on the customer map. We did not add any approval step in this phase, which makes onboarding faster, though we plan to add vendor verification in Phase 2.

7.6Payment

In the current version, payment is not actually processed. We added a dummy "Pay Now" button that shows a fake confirmation screen. The reason is that the main focus of this project is vendor discovery and price comparison, not payment handling. Full payment integration using Razorpay or PhonePe is planned for future work.

8. RESULTS AND DISCUSSION

We tested the system using a dataset of 15 vendors that we manually registered, spread across a simulated college campus area of about 1.2 km². The results were mostly as expected, with a few interesting observations.

8.1Proximity Filtering

The vendor filtering worked correctly in all test cases. When we

Shop C	480 m	₹ 2.50 / page

Figure 3: Price Comparison Test Output.

The comparison view showed all three vendors clearly. A user can see that Shop B is cheaper even though it is farther, and can decide accordingly. This is the core value our system provides.

8.4 Response Time

The average backend API response time was around 135 ms. The set a radius of 500 metres, only vendors within that range full page — map markers and vendor cards — loaded within appeared. When we increased it to 1 km, all 15 vendors showed about 1.8 seconds after the location was captured. This is good up. This confirmed that the Haversine filter was applying enough for a real-time application, though it may need correctly at the backend level.

8.2Distance Accuracy

As mentioned earlier, the Haversine distances were around 10–15% shorter than actual walking distances. This is expected and not a bug — the formula gives straight-line distance, not the actual path. What matters for our system is the ordering of vendors by distance, and that was accurate in every test we ran.

8.3Price Comparison

We set up three vendors offering photocopy services at different prices to test the comparison view:

Vendor	Distance	B&W Copy Price
Shop A	120 m	₹ 3.00 / page
Shop B	310 m	₹ 2.00 / page

Optimisation as the number of vendors grows.

8.5 Challenges We Faced

The biggest challenge was GPS accuracy indoors. When we tested inside a building, the

GPS position was off by 25–30 metres, which caused vendor markers to appear in slightly wrong locations on the map. We handled this with the manual pin option, but it is still something that needs a better solution in the future.

Another challenge was that our current approach fetches all vendors from MongoDB and filters them in the Node.js code. This works fine for a small dataset, but it will slow down once there are hundreds or thousands of vendors. We plan to switch to MongoDB's built-in geospatial queries ($nearSphere) to fix this scalability issue.

9. CONCLUSION

This project started from a problem we personally experienced — not being able to find basic services in a new place — and we tried to build something that genuinely solves it. The system allows users to discover nearby local vendors in real time, see their service prices, and compare options before making a decision. For vendors, it provides a simple way to become digitally visible without any complicated setup.

The price comparison feature was the most impactful part of the system from a usability perspective. Even during small-scale testing, it was clear that this kind of information can save users time and money. We are happy with how the core system turned out, and we believe it has good potential to grow into a more complete product with the future improvements we have planned.

10. FUTURE SCOPE

- **AI Recommendations:** Suggest vendors based on the user's past activity and preferences.
- **Ratings and Reviews:** Let users rate vendors and leave comments after visiting.
- **Online Orders:** Allow users to place advance service orders like document printing.
- **Real Payment Gateway:** Integrate Razorpay or PhonePe for actual transactions.
- **Admin Panel (Phase 2):** Dashboard for managing vendors, handling complaints, and viewing usage data.
- **Geospatial Optimisation:** Use MongoDB's $nearSphere to improve query performance at scale.
- **Navigation Support:** Add turn-by-turn directions to help users reach the selected vendor.

ACKNOWLEDGEMENT

We would like to sincerely thank everyone who helped us during the course of this project. We are especially grateful to **Prof. Vishal Upmanu**, Dept. of CSE, RD Engineering College, Duhai, Ghaziabad, for guiding us throughout the development process and helping us understand the concepts whenever we were stuck. This project would not have been possible without his support and encouragement.

REFERENCES

1. Bao, J., Zheng, Y., Wilkie, D., & Mokbel, M. (2015). Recommendations in location-based social networks: A survey. *GeoInformatica*, 19(3), 525–565.
2. Singla, M., & Sachdeva, R. (2018). Hyperlocal e-commerce: Architecture and challenges. *International Journal of Computer Applications*, 182(11), 35–40.
3. Sinnott, R. W. (1984). Virtues of the Haversine. *Sky and Telescope*, 68(2), 159.
4. Jensen, R. (2007). The digital provide: Information technology, market performance, and welfare in south Indian fisheries. *Quarterly Journal of Economics*, 122(3), 879–924.
5. Agafonkin, V. (2013). Leaflet.js — open-source maps for the web. Retrieved from https://leafletjs.com
6. Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14(6), 80–83.
7. Chodorow, K. (2013). *MongoDB: The Definitive Guide* (2nd ed.). O'Reilly Media.
8. Banks, A., & Porcello, E. (2017). *Learning React*. O'Reilly Media.
9. Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation). University of California, Irvine.
10. Longley, P. A., et al. (2015). *Geographic Information Science and Systems* (4th ed.). Wiley.