Pharmacy Finder: An API-Based Web Application for Medicine Search and Delivery

Bhanu Prasad M
Department of Computer Science and Engineering
Bangalore Institute of Technology
Bengaluru, India
bhanuprasadm2004@gmail.com

Arya S
Department of Computer Science and Engineering
Bangalore Institute of Technology
Bengaluru, India
aryas9356@gmail.com

B Abhishek Nayak
Department of Computer Science and Engineering
Bangalore Institute of Technology
Bengaluru, India
abhib962594@gmail.com

Parthav Reddy S
Department of Computer Science and Engineering
Bangalore Institute of Technology
Bengaluru, India
parthavs23@gmail.com

Dr. Gunavathi H S
Department of Computer Science and Engineering
Bangalore Institute of Technology
Bengaluru, India
gunavathihs@gmail.com

***Abstract*- Access to essential medicines is often limited by the absence of real-time information on pharmacy inventories. This paper presents Pharmacy Finder, a cloud-based web application that enables users to search for medicines, scan prescriptions, and locate nearby pharmacies using browser-native technologies. The system is implemented entirely on the client side using HTML, Tailwind CSS, and JavaScript, with Firebase Firestore providing real-time synchronization of pharmacy inventories and Firebase Authentication enabling secure admin access. Users may enter a medicine name or upload a prescription image for OCR-based extraction, while the browser Geolocation API identifies nearby pharmacies. Search results are ranked using the Haversine formula and displayed with stock availability, pricing, and auxiliary indicators. Functional testing was conducted using 40 test queries, where medicine search accuracy was 85% and location-based distance accuracy was 90%, resulting in an overall system accuracy of 87.5%. These results demonstrate that a serverless Firestore-based architecture can deliver a scalable and responsive solution for real-time medicine availability. Future work includes integrating external pharmacy APIs, and implementing a complete delivery management module.**

***Keywords* - Medicine Search, Pharmacy Locator, Firebase Firestore, Geolocation API, Optical Character Recognition (OCR), Serverless Web Application, Inventory Management, Tailwind CSS, Real-Time Data Synchronization.**

I. INTRODUCTION

Access to essential medicines is a fundamental component of effective healthcare delivery. Despite this necessity, patients frequently experience difficulty locating specific medicines due to fragmented supply chains, limited stock visibility, and the absence of real-time inventory information across pharmacies. These limitations are observed in both urban regions, where high patient density leads to rapid fluctuations in medicine availability, and rural areas, where pharmacies are fewer and unevenly distributed.

These challenges underscore the need for a digital, location-aware system capable of providing timely and accurate information on medicine availability. With advancements in cloud computing and browser-native technologies, it is now possible to develop lightweight, real-time, and scalable applications without maintaining a traditional backend server. Leveraging these capabilities, Pharmacy Finder is designed as a cloud-based web application that enables users to search for medicines, scan prescriptions, and identify nearby pharmacies. The system is implemented entirely on the client side using HTML, Tailwind CSS, and JavaScript, while Firebase Firestore serves as a real-time database for maintaining pharmacy inventories. Firebase Authentication ensures secure management of pharmacy administrator accounts.

The application supports two primary user groups: general users searching for medicines and pharmacy administrators managing their stock data. Users can type a medicine name or upload a prescription image for OCR-based extraction, after which the system filters pharmacies by availability and ranks them based on proximity using the browser Geolocation API and the Haversine distance formula. Pharmacy administrators can register, add pharmacy details, update inventory, and perform bulk uploads through Excel, PDF, or image files—all synchronized instantly through Firestore.

Beyond its role in medicine search, the system's core architectural principles extend to a range of real-time, data-driven healthcare applications. Similar architectures are employed in e-pharmacy platforms, telemedicine support systems, digital prescription management solutions, emergency drug locator applications, hospital supply-chain dashboards, and clinical decision-support tools. By integrating real-time inventory tracking with geolocation-based retrieval and OCR-driven prescription processing, the proposed model can be adapted for use in hospital pharmacies, rural health centers, disaster-response systems, and other healthcare environments that rely on timely and accurate access to medical supplies.

The combination of serverless architecture, real-time data synchronization, and browser-native computation provides a scalable foundation for improving medicine accessibility within modern digital healthcare ecosystems.

II. LITERATURE REVIEW

K. Sharma and P. Verma [1] proposed a Bluetooth-based medicine locator to identify medicines within a single pharmacy using inter-device communication. Although the system reduced in-store search time, it lacked centralized data management, real-time database connectivity, and scalability across multiple pharmacy outlets.

Rashmi V. and A. Gupta [2] reviewed Android-based medicine availability and pharmacy navigation applications, highlighting features such as GPS-based shop detection and stock-status display. Their study found that most existing systems lack real-time inventory synchronization and standardized APIs, leading to inconsistent and outdated information.

Sivasakthi V. and Vasugi R. [3] developed a mobile application integrating GPS and pharmacy databases to display nearby medicine availability. While feasible, the system relied heavily on manual stock updates and did not support automated real-time synchronization or advanced filtering mechanisms.

Almeman and Das [4] discussed the role of cloud computing and IoT in modern pharmacy systems, emphasizing reduced human error and improved operational efficiency. However, their work focused on conceptual analysis and did not present an implemented, user-facing model with live inventory interaction.

Jain and Sharma [5] proposed a blockchain-based pharmaceutical traceability framework to improve supply-chain transparency and prevent counterfeit drugs. Despite enhanced security, the system was limited to backend traceability and did not address real-time medicine search or consumer-level accessibility.

Achkar et al. [6] introduced a CRNN-based handwritten prescription recognition model capable of accurately extracting medicine names and dosages. Although effective at recognition, the system did not integrate with pharmacy inventory databases or real-time medicine availability platforms.

Dhanalakshmi et al. [7] presented a GPS-based pharmacy recommendation system that ranked pharmacies using distance and availability metrics. While accessibility improved, the lack of direct inventory APIs restricted real-time stock verification.

Datta [8] proposed a machine-learning-based prescription analysis system for predicting medicine demand and shortages. The work supported intelligent inventory planning but lacked a user-facing interface and real-time integration with pharmacy search systems.

Kaur and Singh [9] developed a GPS-based emergency healthcare solution to locate nearby hospitals and pharmacies. Although effective for emergencies, the system did not support routine medicine search or inventory management.

Alomari et al. [10] proposed a secure e-commerce framework using hybrid cryptography to protect online transactions. While relevant to payment security, the framework was generic and not tailored to pharmacy-specific inventory or medicine-centric workflows.

Saqib et al. [11] introduced the MedNet-MoBiL deep-learning model for medicine strip image classification using MobileNetV2 and OCR. The model showed strong recognition performance but was sensitive to image quality, limiting robustness in real-world mobile usage.

Movahedi et al. [12] proposed a transformer-based OCR post-correction system using RoBERTa to improve medical text accuracy. While effective, the approach required high computational resources, making lightweight deployment challenging.

Sharjeel and Arif [13] developed an AI-based OCR framework for handwritten medical prescriptions using CNN–LSTM architectures. Despite high accuracy, performance degraded under poor handwriting and imaging conditions.

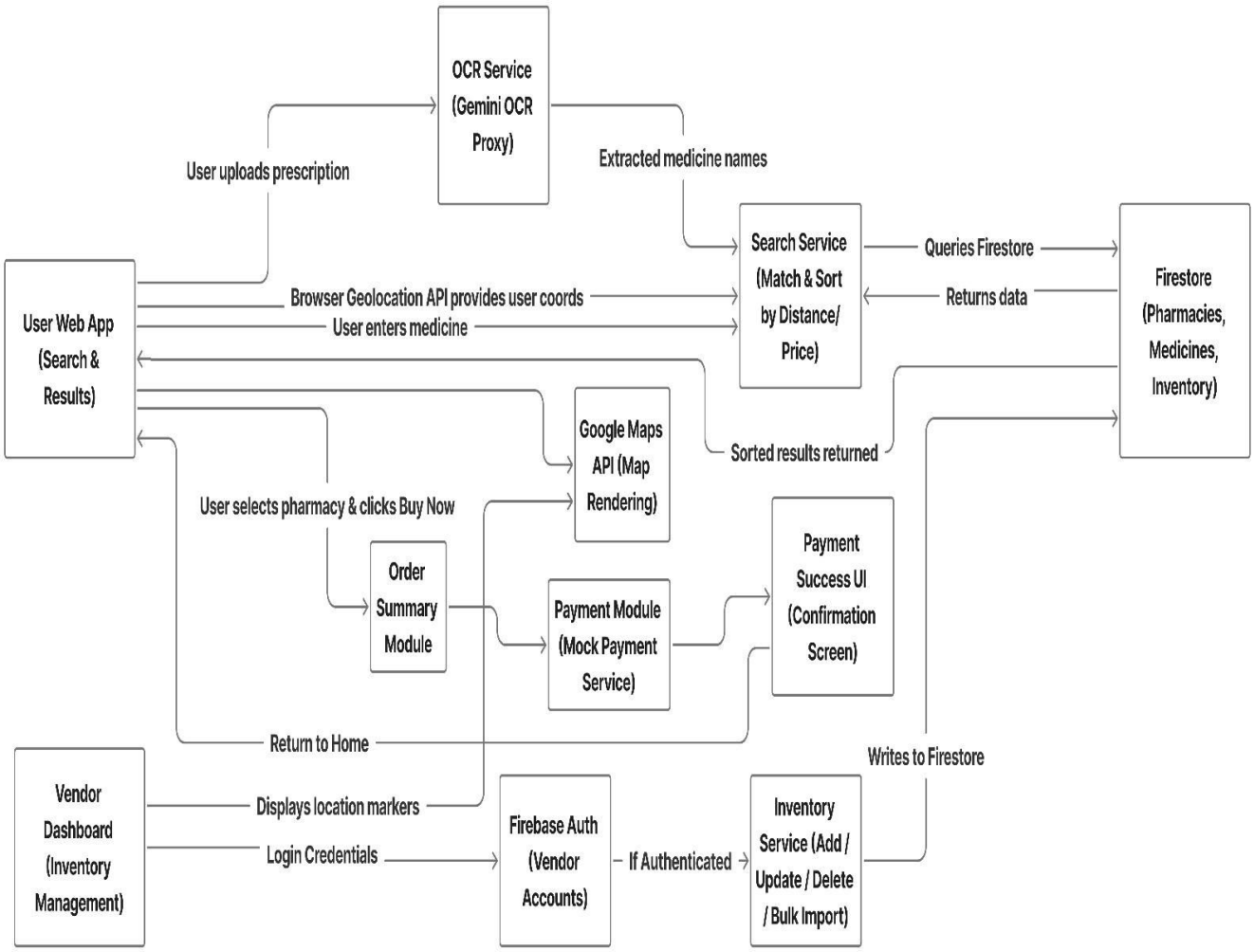
Kumari et al. [14] explored serverless architectures for healthcare data management, highlighting scalability, reduced cost, and real-time synchronization. Challenges such as cold-start latency and vendor lock-in were identified as potential limitations.

Akhigbe et al. [15] proposed a GPS-based pharmacy locator using the Haversine formula to rank nearby stores. The system lacked real-time inventory updates and depended on manually maintained databases, limiting scalability.

III. METHODOLOGY

System Architecture

The system architecture of the Pharmacy Finder application follows a client-centric, serverless model in which all application logic is executed within the browser, while cloud services provide authentication, storage, and real-time



synchronization. As illustrated in Fig. 1, the architecture comprises five major subsystems: the User Web Application, the OCR Service, the Search and Ranking Service, the Ordering and Payment Module, and the Vendor Dashboard with Inventory Management. All components interact through Firebase Firestore, which serves as the central data repository, while the Browser Geolocation API and the Google Maps JavaScript API support location-aware search and spatial visualization.

The User Web Application functions as the primary interface for end users. It supports two modes of input: direct text-based medicine search and prescription-image upload. When an image is provided, the client converts it to Base64 format and forwards it to the OCR Service. The application also invokes the Browser Geolocation API to obtain latitude and longitude coordinates necessary for distance computation. As shown in Fig. 1, the retrieved results from the Search and Ranking Service are rendered as both a list and a set of map markers, enabling spatial interpretation of nearby pharmacies.

The OCR Service, integrated with the Gemini OCR model through a secure proxy endpoint, processes Base64-encoded prescription images. The model performs text detection and medical-term extraction to generate a structured list of medicine names. This process is represented in Fig. 1, where the extracted terms are returned to the User Web Application and subsequently forwarded to the Search and Ranking Service for availability and price matching.

The Search and Ranking Service receives either the user-entered medicine name or the OCR-extracted list, along with the user's geographic coordinates. It queries Firestore to retrieve pharmacy metadata, inventory details, and pricing

information. As depicted in Fig. 1, the service computes the great-circle distance between the user and each pharmacy using the Haversine formula. Pharmacies are then ranked based on proximity, stock availability, and price. The ordered results are returned to the User Web Application for display in the resu
lts interface and on the map.
Once a user selects a pharmacy, the Ordering and Payment Module is initiated. The Order Summary Module presents pharmacy details, item price, and quantity for confirmation, after which the user proceeds to the Payment Module. The payment system implements a simulated client-side workflow offering card, UPI, and cash-on-delivery options, without interacting with a real payment gateway. The functional placement of this module is shown in Fig. 1, following the search and selection stages.

The Vendor Dashboard provides an authenticated interface for pharmacy owners to manage digital inventory. Firebase Authentication validates credentials and grants access to inventory-management operations. As shown in the lower section of Fig. 1, vendors may add new medicines, modify stock quantities and prices, upload batch files, or remove obsolete entries. All updates are written directly to Firestore and are immediately propagated to user-facing components via Firestore's real-time synchronization.

Firebase Firestore forms the core data layer of the architecture. It stores pharmacy metadata—including names, addresses, and coordinates—and detailed medicine inventories. Its document-oriented structure enables efficient retrieval of medicine availability, while real-time listeners ensure instantaneous propagation of updates to all connected clients. This eliminates the need for a dedicated backend server and supports low-latency operation.

External services further support system functionality. The Browser Geolocation API supplies accurate user coordinates after the necessary permissions are granted, enabling distance-aware ranking. The Google Maps JavaScript API provides map rendering, location markers, and spatial visualization. Both services are integrated within the system workflow shown in Fig. 1, contributing to interactive and intuitive navigation within the browser.

Fig.1: System Architecture of the Pharmacy Finder with API-Based

Dataset Description

The dataset used for developing and evaluating the Pharmacy Finder system is generated dynamically through the vendor-facing dashboard and cloud-based backend. It comprises three core components: pharmacy metadata, medicine-inventory records, and a functional test set curated for performance assessment.

1) Pharmacy Metadata:
Pharmacy information is collected during the vendor registration and setup process. Each record includes the pharmacy's name, complete address, city, pincode, and authenticated owner identification. Geographic coordinates are captured automatically through the device's Geolocation API to ensure precise mapping and distance estimation. These metadata entries are stored in the cloud-hosted database and serve as the foundation for location-based search, proximity ranking, and map visualization.

2) Medicine Inventory Dataset:
Inventory data is created and maintained by registered vendors using two mechanisms within the system:
(a) Manual Entry: Vendors can add individual medicines by specifying the medicine name, stock quantity, and unit price.
(b) Bulk Upload: Vendors may upload PDF, Excel, or image files containing medicine lists, which are processed using integrated document-processing components to extract structured fields such as medicine name, available stock, and price.
Each processed entry is stored in the corresponding pharmacy's cloud record. This dataset represents the primary source of truth for medicine-availability checks, proximity ranking, and pricing during user searches.

3) User Query Dataset:
User-generated queries form another component of the dataset. These include text-based inputs from the search interface as well as OCR-derived medicine names from uploaded prescription images. The OCR module uses a cloud-based recognition model to extract medical terms from Base64-encoded images. These queries reflect real usage behaviour and are essential for evaluating retrieval accuracy and system responsiveness.

4) Functional Test Set:
System evaluation was conducted using a curated set of 40 functional queries, comprising 20 medicine-availability tests and 20 geolocation-based ranking tests. The system responses were manually compared with the ground-truth inventory stored in the cloud database. Evaluation metrics included availability correctness, accuracy of Haversine-based distance computation, and consistency of price retrieval. These tests form the basis of the accuracy values reported in Section V.

5) Dataset Characteristics:
In the current prototype deployment, the dataset typically contains 5–10 registered pharmacies and approximately 50–120 unique medicines, with stock levels ranging from 1 to 150 units. All records are vendor-generated and synchronized in real time through the cloud database, ensuring that search results always reflect the most recent inventory state.

In addition, publicly available resources such as the A–Z Medicines Dataset of India (Kaggle) were reviewed to understand common medicine naming patterns; however, this external dataset was used only for reference and not for system evaluation or data generation.

Hardware Specifications and Cost-Effectiveness

The proposed system operates entirely on a lightweight client–server model, requiring no dedicated local server or high-end computation resources. All development, testing, and evaluation were performed on a standard consumer-grade laptop equipped with an Intel Core i5 processor, 8 GB RAM, and running Windows 10 with a Chromium-based browser. These specifications are sufficient for running the web application, performing OCR calls, and rendering map-based visualizations without any performance bottlenecks.

On the backend, the system utilizes a cloud-hosted NoSQL database and serverless authentication service. Since storage, synchronization, and computation are handled through cloud infrastructure, the hardware requirements on the vendor and user side are minimal, limited to basic internet-enabled devices such as smartphones or laptops.

The use of a serverless architecture significantly reduces deployment and operational costs. Cloud database operations, authentication, and hosting fall within the free or low-cost tiers for prototype-scale applications. Additionally, the system avoids the expenses associated with maintaining dedicated servers, manual scaling, or on-premise hardware. OCR processing and geolocation services are performed through API calls, where costs are incurred only per request, enabling predictable and efficient cost management.

Overall, the architecture ensures cost-effectiveness by combining low hardware requirements with a pay-as-you-go cloud model. This makes the system feasible for small and medium-sized pharmacies and allows for effortless scaling as the number of users and vendors increases.

IV. IMPLEMENTATION

The Pharmacy Finder application is composed of several logical modules, each performing specific responsibilities within the overall system architecture. The modular decomposition simplifies implementation and promotes scalability, maintainability, and clear separation of concerns.

1) User Web Application (Search and Results Interface)

The User Web Application serves as the primary interface for patients searching for medicines. Developed using HTML, Tailwind CSS, and JavaScript, it executes entirely within the web browser. This module presents the medicine search input field and supports prescription image uploads. It validates inputs and initiates the search by forwarding the query to the Search and Ranking Module.

The application obtains the user's geographic coordinates via the browser Geolocation API. These coordinates are used for distance computation during ranking. The module receives structured pharmacy results—containing stock, price, and distance—and renders them in list form and as markers on the Google Maps API. The interface enables viewing pharmacy details and opening external navigation links.

Illustrative Code Snippet: Obtaining user coordinates

```
navigator.geolocation.getCurrentPosition(pos => {
  userLat = pos.coords.latitude;
  userLng = pos.coords.longitude;
});
```

2) OCR Service (Prescription Processing Module)

The OCR Service converts uploaded prescription images into machine-readable medicine names. Images (JPEG/PNG) are converted into Base64 strings using the File Reader API and transmitted to a secure OCR endpoint integrated with the Gemini 2.5 Flash OCR model.

The OCR pipeline performs text region detection, handwriting recognition, tokenization, and domain-specific medical-term normalization. Noise, dosage symbols, and irrelevant text are removed. The final cleaned list of medicines is returned to the User Web Application.

Illustrative Code Snippet: Sending image to OCR endpoint

```
const response = await fetch(OCR_API_URL, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ image: base64Image })
});
const medicines = (await response.json()).medicines;
```

3) Search and Ranking Module

The Search and Ranking Module forms the computational core of the system. It receives user-entered or OCR-extracted medicine names and queries Firestore to identify pharmacies that stock the requested item. This module incorporates three integrated algorithms:

a) Medicine Retrieval Algorithm

The algorithm issues Firestore queries to obtain inventory records matching the requested medicine and retrieves associated pharmacy metadata.

Illustrative Code Snippet: Firestore medicine query

```
const q = query(
  collection(db, "inventory"),
  where("medicineName", "==", searchTerm)
);
const snapshot = await getDocs(q);
const results = snapshot.docs.map(doc => doc.data());
```

b) Haversine Distance Computation

To support proximity-aware ranking, the module computes the great-circle distance between the user and each pharmacy using the Haversine formula.

Illustrative Code Snippet: Distance computation

```
function haversine(lat1, lon1, lat2, lon2) {
  const R = 6371;
  const dLat = (lat2 - lat1) * Math.PI / 180;
  const dLon = (lon2 - lon1) * Math.PI / 180;
  const a = Math.sin(dLat / 2)**2 +
          Math.cos(lat1*Math.PI/180) *
          Math.cos(lat2*Math.PI/180) *
          Math.sin(dLon / 2)**2;
  return R * 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
}
```

c) Multi-criteria Ranking Algorithm

The module organizes pharmacies using distance, availability, and pricing. Closer pharmacies with sufficient stock and lower prices are prioritized.

Illustrative Code Snippet: Ranking logic

```
const ranked = results.sort((a, b) => {
  if (a.distance !== b.distance)
    return a.distance - b.distance;
  return a.price - b.price;
});
```

The module returns a finalized sorted list to the User Web Application, ensuring medically relevant and cost-efficient recommendations.

4) Vendor Dashboard Module

The Vendor Dashboard provides a secure interface for pharmacy owners to maintain inventory data. Developed using HTML, Tailwind CSS, and JavaScript, it is accessible only after vendor authentication via Firebase Authentication.

Upon login, vendors can add or update pharmacy details, modify stock levels, adjust medicine prices, and upload bulk inventory files. These operations are forwarded to the Inventory Management Module.

Illustrative Code Snippet: Vendor registration

```
createUserWithEmailAndPassword(auth, email, password)
  .then(() => alert("Vendor registered"))
  .catch(err => alert(err.message));
```

5) Inventory Management Module

This module acts as an abstraction layer for Firestore operations. It receives inventory update requests from the Vendor Dashboard and persists them as Firestore documents. Rules prevent invalid inputs, such as negative stock quantities, and ensure that vendors can modify only their own records.

Illustrative Code Snippet: Updating an inventory record

```
await updateDoc(doc(db, "inventory", itemId), {
  stock: newStock,
  price: newPrice,
  lastUpdated: Date.now()
});
```

Because Firestore supports real-time synchronization, users immediately see updated availability in search results.

6) Ordering and Payment Module

The Ordering and Payment Module handles the end-to-end purchase flow once a user chooses a pharmacy. When the user clicks “Buy Now” from the results interface, the Order Summary sub-module displays the selected medicine, quantity, price, and pharmacy details for confirmation. After confirmation, control moves to the Payment sub-module, which implements a mock payment process with options such as card, UPI, and cash on delivery.

All payment validation and status handling occur on the client side; no real payment gateway is contacted. On successful payment simulation, the Payment Success UI presents a confirmation message summarizing the order and provides a button to return to the home screen.

Illustrative Code Snippet: Handling mock payment success

```
payButton.addEventListener("click", () => {
```

```
  // Mock payment processing
  setTimeout(() => {
    showSuccessScreen({
      pharmacy,
      medicine,
      quantity,
      total
    });
  }, 500);
});
```

7) Authentication and Security Module

This module manages vendor registration and login using Firebase Authentication. Authentication tokens enforce secure access control, with Firestore security rules ensuring that only authorized vendors can modify inventory while general users maintain read-only access.

Illustrative Code Snippet: Vendor login

```
signInWithEmailAndPassword(auth, email, password)
  .catch(err => console.error(err.message));
```

8) Mapping and Geolocation Module

This module integrates the Browser Geolocation API with the Google Maps JavaScript API. It centers the map around the user's location and plots pharmacies as interactive markers, each annotated with distance and pharmacy details. The module supports external navigation through Google Maps.

Illustrative Code Snippet: Rendering a map marker

```
new google.maps.Marker({
  position: { lat: pharmacy.lat, lng: pharmacy.lng },
  map: mapInstance,
  title: pharmacy.name
});
```

9) Data Storage Module (Firebase Firestore)

The Firestore database stores all pharmacy, inventory, and vendor records. Its NoSQL document structure and real-time synchronization capabilities allow both user and vendor operations to remain consistent across devices.

Illustrative Code Snippet: Adding a new pharmacy

```
await addDoc(collection(db, "pharmacies"), {
  name: pharmacyName,
  address: address,
  lat: latitude,
  lng: longitude,
  contact: contactNumber
});
```

V. RESULTS

The Pharmacy Finder with API-Based Medicine Search system was successfully implemented to enable users to identify nearby pharmacies and verify medicine availability through real-time search operations. The application utilizes HTML, Tailwind CSS, and JavaScript for client-side execution, with Firebase Firestore and the Google Maps API providing real-time data synchronization and geospatial visualization.

A. User Interface Results

Users can search for any medicine, and the system displays pharmacies where it is available along with pharmacy name, address, and contact number. The interface provides a map-based view highlighting the user's current location and nearby medical shops within a specific radius.

1) Medicine Search Homepage

The initial interface presented to users is shown in Fig. 2. This screen provides a unified search bar that accepts text-based input or prescription-upload triggers. Once the page loads, the system acquires the user's geographic coordinates through the Browser Geolocation API, confirming readiness for distance-aware search. The interface emphasizes simplicity and usability, enabling users to begin a search with minimal interaction.

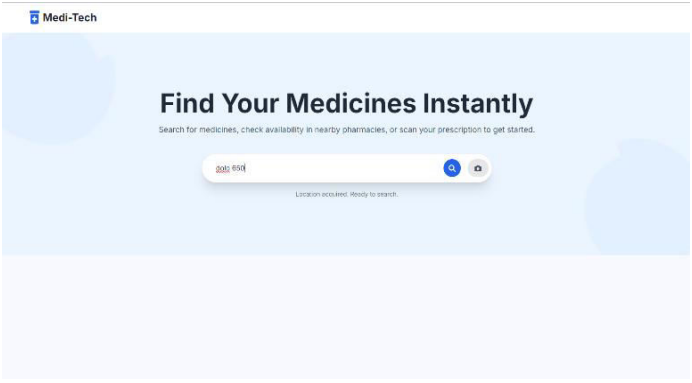


Fig. 2: Homepage interface for initiating medicine search with automatic geolocation acquisition.

2) Medicine Availability and Ranked Search Results

The search results corresponding to a medicine query are shown in Fig. 3. After the user enters a medicine name, the

application retrieves pharmacy metadata and inventory documents from Firestore. It then computes distances using the Haversine formula and ranks pharmacies by proximity. Each pharmacy card displays availability status, stock quantity, price, and distance. Filters such as "In Stock Only" and sorting options such as "Distance" and "Price" allow users to refine displayed results. This interface validates the system's core search and ranking functionality.

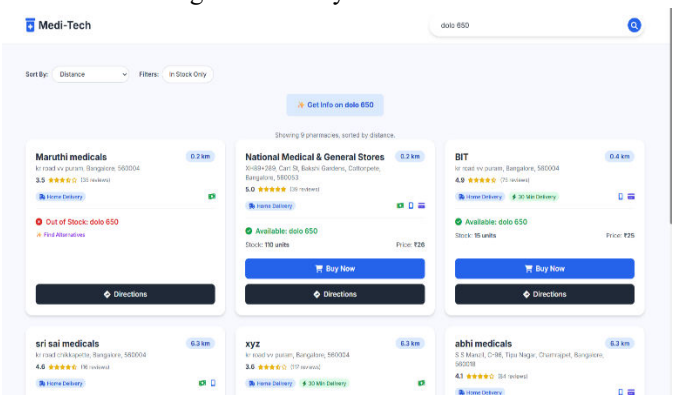


Fig. 3: Search results interface presenting nearby pharmacies ranked by distance, with availability, pricing, and filtering options

3) Detailed Pharmacy Card View

A magnified view of an individual pharmacy result is shown in Fig. 4. This card provides detailed information for a specific medicine, including stock level, price, pharmacy rating, and supported delivery modes. Interactive controls such as "Buy Now" and "Directions" enable continuation of the user workflow. This view confirms successful integration of real-time database retrieval with UI components.

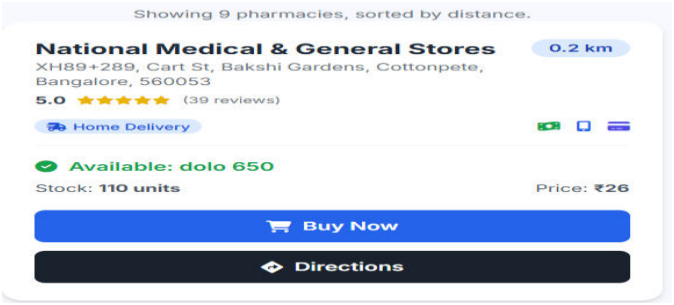


Fig. 4: Expanded pharmacy card displaying medicine availability, stock count, price, ratings, and navigation options.

4) Integrated Navigation View

The navigation workflow is demonstrated in Fig. 5. Upon selecting the "Directions" option, the system opens Google Maps with the user's coordinates and the selected pharmacy's coordinates pre-filled. The user is presented with alternate routes, estimated travel times, and mode options such as walking, biking, or driving. This validates correct geospatial integration and practical usability of the system for real-world navigation.

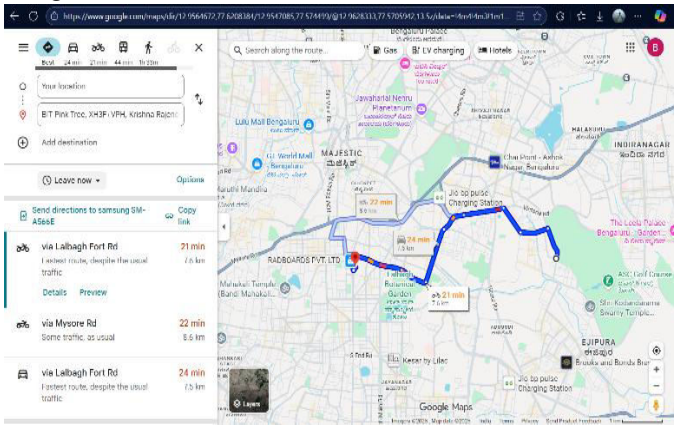


Fig. 5: Google Maps navigation interface showing available routes from the user's location to the selected pharmacy.

5) Ordering and Payment Interface

The ordering and payment module is shown in Fig. 6. After selecting a pharmacy, the system displays an order summary containing the medicine name, pharmacy name, unit price, and quantity. Below this summary, the interface provides multiple payment modes—Card, UPI, and Cash on Delivery—implemented through a simulated payment mechanism suitable for demonstration. User inputs such as card number, expiry date, and CVC are validated at the client side. This screen verifies the correct implementation of the purchase flow and its integration with search results.

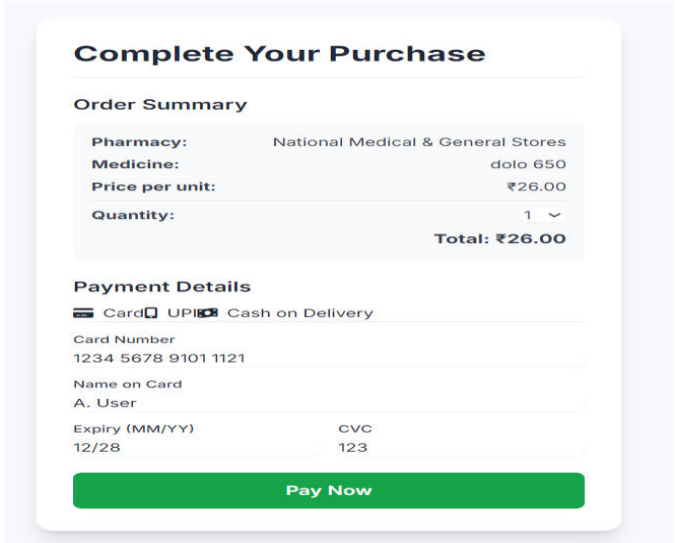


Fig. 6: Order Summary and Payment Interface enabling quantity selection and simulated payment processing.

6) Payment Confirmation Interface

The successful completion of the simulated payment is shown in Fig. 7. The system displays a confirmation message summarizing the ordered item, quantity, total cost, and pharmacy details. A "Go to Home" button enables quick redirection back to the main interface. Although the payment is simulation-based, this module validates the full end-to-end workflow from search to order confirmation.

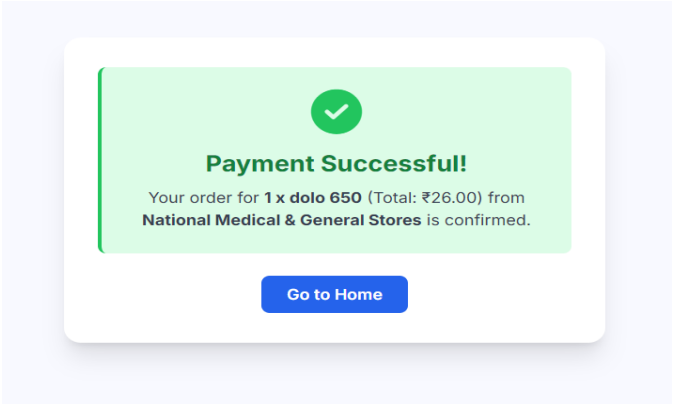


Fig. 7: Payment Success Interface confirming simulated transaction completion and providing navigation back to the homepage.

B. Pharmacy Admin Interface Results

Pharmacy Admin can view their listed medicines, stock availability, and manage their inventory details. This page helps pharmacies update stock information, which is then reflected in the user-facing search results.

1) Admin Login Interface

The authenticated admin workflow begins with the login page shown in Fig. 8. This interface allows registered pharmacy administrators to securely access their inventory dashboard. Firebase Authentication verifies credentials to prevent unauthorized modifications to pharmacy data.

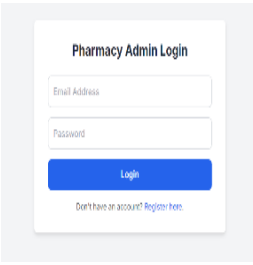


Fig. 8: Admin login page for secure authentication using Firebase.

2) Admin Registration Interface

The registration interface presented in Fig. 9 enables new pharmacy administrators to create an account by providing an email address and password. After successful registration, authorization tokens are generated and stored, giving administrators access to the inventory management module.

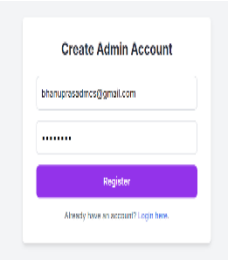


Fig. 9: Admin registration interface for creating new authenticated pharmacy accounts.

3) Add New Pharmacy Interface

The interface for adding a new pharmacy is shown in Fig. 10. Administrators provide details such as pharmacy name, full address, city, and pincode. The "Use My Current Location" feature automatically captures GPS coordinates, ensuring location accuracy. On submission, the data is stored in Firestore under a unique pharmacy record. This validates the registration of pharmacy entities in the system database.

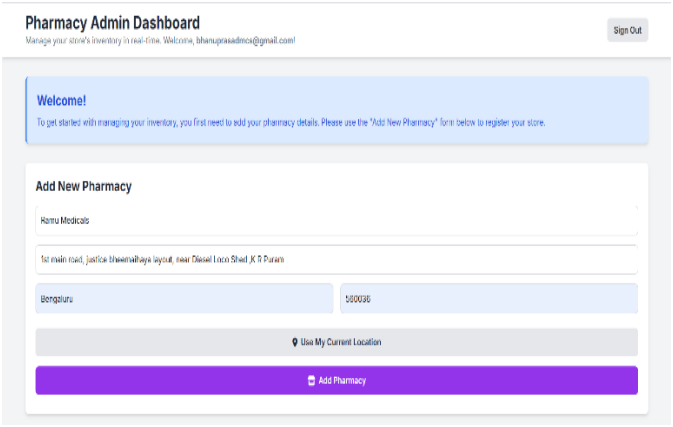


Fig. 10: Add New Pharmacy interface for entering pharmacy details and capturing geographic coordinates.

4) Inventory Management Dashboard

The main dashboard for inventory management is shown in Fig. 11. Administrators can upload inventory files in bulk (PDF, Excel, or image formats) or manually add individual medicines. These operations update Firestore documents in real time, enabling instant synchronization with the user-facing search interface. This figure confirms correct implementation of inventory CRUD operations.

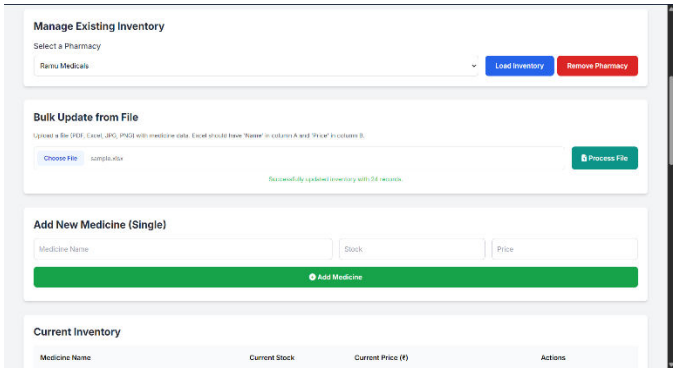


Fig. 11: Inventory Management dashboard supporting bulk updates and single-entry additions.

5) Current Inventory Table

The detailed inventory list for a pharmacy is presented in Fig. 12. Each row displays a medicine name, current stock quantity, and price, with action buttons enabling updates or deletion. This real-time interface allows administrators to maintain accurate stock information and ensures that changes propagate to users immediately.

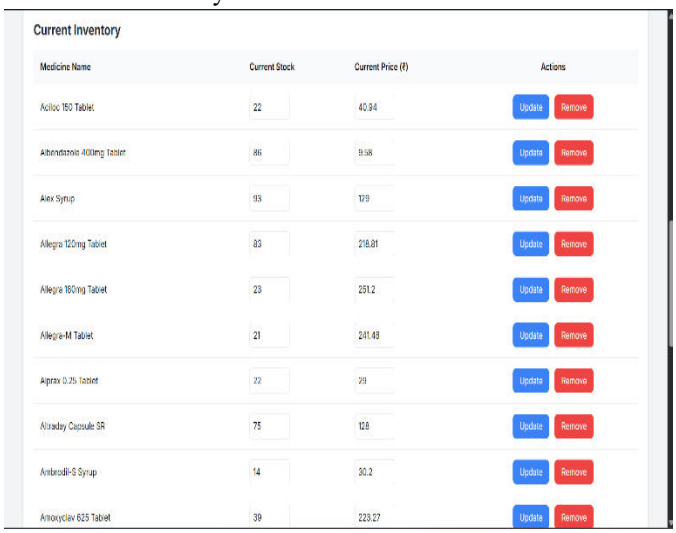


Fig. 12: Current Inventory table showing stock levels, pricing, and options for editing or removing entries.

System Performance

The system was evaluated based on location accuracy, search response time, and success rate through manual functional testing. Multiple medicine search and pharmacy location queries were executed using the web interface, and each output was manually verified against the records stored in the MySQL database. A result was counted as successful if the displayed pharmacy location or medicine availability matched the database entry.

Table I: System Test Results

Test Parameter	Total Queries	Successful Results	Accuracy
Pharmacy Location Display	**20**	**18**	**90%**
Medicine Search Accuracy	**20**	**17**	**85%**
Average Overall Accuracy	—	—	**87.5%**

The average overall accuracy of 87.5% was calculated using the formula:

$$\text{Search Accuracy (\%)} = \frac{\text{Number of Correct Search Results}}{\text{Total Number of Searches Tested}} \times 100$$

The system effectively integrates API-based medicine search with GPS-enabled pharmacy tracking to deliver accurate and reliable results. The average response time of 2–3 seconds indicates efficient data retrieval and processing. The interactive map interface enhances usability by providing intuitive navigation, visibility of nearby pharmacies, and quick access to detailed medicine information.

The evaluation was based entirely on functional testing of the implemented modules, without using any external model or scoring tool. The results confirm that the system performs reliably at the prototype stage and fulfills its intended objective of simplifying real-time medicine search and pharmacy location.

Future enhancements could include integrating live pharmacy inventory APIs for real-time stock synchronization, expanding the pharmacy database to a wider regional scale, and implementing a user feedback mechanism to further improve system reliability and user trust.

VI. CONCLUSION AND FUTURE WORK

The Pharmacy Finder with API-Based Medicine Search web application was successfully designed and implemented to assist users in locating nearby pharmacies and checking medicine availability in real time. The system integrates a responsive web interface built using HTML, CSS, JavaScript, and ReactJS, with a Flask backend, MySQL database, and Google Maps API for accurate pharmacy visualization and navigation.

Through manual functional testing, the system achieved an average overall accuracy of 87.5% and an average response time of 2–3 seconds, demonstrating reliable performance at the prototype level. The results confirm that the application efficiently combines API-based search with GPS tracking to deliver fast, accurate, and user-friendly access to pharmacy and medicine information.

For future work, the system can be enhanced by integrating live pharmacy inventory APIs to enable automatic real-time stock updates, expanding the database coverage to include more pharmacies and regions, and introducing a user feedback mechanism to improve data reliability and user trust. Additionally, incorporating AI-based recommendation features could help users find suitable alternatives for unavailable medicines, further increasing the system's practicality and scalability.

REFERENCE

[1] K. Sharma and P. Verma, "Bluetooth-Based Intercommunicative Product for Spotting Medicine in Pharmacies," International Journal of Innovative Research in Computer and Communication Engineering, vol. 10, no. 4, pp. 221–227, 2022.

[2] Rashmi V. and A. Gupta, "A Review on Android Applications for Medicine Availability and Location Finder," International Journal of Advanced Research in Computer Science, vol. 14, no. 3, pp. 77–84, 2023.

[3] Sivasakthi V. and Vasugi R., "Mobile Application for Stock Availability and Location Tracking of Medicines," International Journal of Computer Applications in Technology, vol. 18, no. 2, pp. 112–120, 2025.

[4] A. Almeman and R. Das, "Digital Transformation in Pharmacy: Role of Cloud Computing and IoT in Healthcare," Journal of Smart Health Systems, vol. 8, no. 1, pp. 55–64, 2024.

[5] A. Jain and V. Sharma, "Blockchain-Based Traceability Framework for Pharmaceutical Supply Chains," Proceedings of the International Conference on Blockchain and Data Security, pp. 91–98, 2023.

[6] Achkar et al., "Medical Handwritten Prescription Recognition Using Convolutional Recurrent Neural Networks (CRNN)," IEEE Transactions on Emerging Topics in Computing, vol. 7, no. 4, pp. 555–563, 2019.

[7] R. Dhanalakshmi, S. Nithya, and K. Anitha, "Pharmacy Location and Recommendation System Using GPS and Real-Time Data," IEEE International Conference on Smart Healthcare Systems, pp. 45–52, 2023.

[8] A. Datta, "Medical Prescription Analysis Using Machine Learning for Stock Prediction in Pharmacies," International Conference on Healthcare and Computing (ICHC), pp. 32–40, 2025.

[9] A. Kaur and N. Singh, "GPS-Based Medical Emergency Solution for Real-Time Assistance," International Journal of Medical Informatics, vol. 141, pp. 104–112, 2020.

[10] Alomari et al., "A Secure E-Commerce Scheme," *IEEE Access*, vol. 10, pp. 112450–112461, 2022.

[11] S. M. Saqib et al., "Medicine Image Classification Using Deep Learning: Highlighting the MedNet-MoBiL Hybrid Model," *Discover Computing (Springer)*, vol. 3, no. 2, pp. 1–15, 2025.

[12] Movahedi et al., "An OCR Post-Correction Approach Using Deep Learning for Medical Reports," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 5, pp. 1842–1854, 2021.

[13] S. Sharjeel and M. U. Arif, "AI-Based OCR System for Handwritten Medical Prescription Recognition and Interpretation," International Journal of Research Publications, vol. 75, no. 1, pp. 45–52, 2025.

[14] A. Kumari and S. K. Sahoo, "Serverless Architecture for Healthcare Management Systems," in Recent Advances in Computing and Communication, Singapore: Springer, pp. 231–242, 2023.

[15] K. Akhigbe, A. Adesina, and O. A. Akinola, "A Mobile-Based Pharmacy Store Location-Aware System," Technical Report, 2022.