iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.83636

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VI June 2026- Available at www.ijraset.com

A Lightweight Cross-Platform Smart Mess Discovery and Management Application Using Flutter

Atharva Pachpute[1], Srushti Mhaske[2], Sahil Wanjare[3], Digvijay Patil[4]
PES Modern College of Engineering, India

Abstract: ***Mess and tiffin services across urban India serve tens of millions of students and working professionals, yet the operational infrastructure governing discovery, communication, and nutritional awareness within these services remains largely manual, fragmented, and informationally opaque. This paper presents a lightweight cross-platform mobile application that unifies three functionalities absent from any single prior system: Haversine-based geospatial proximity discovery, Firestore-driven community communication, and Gemini API-backed nutritional inference. The system is engineered using Flutter and Dart for cross-platform delivery, Firebase for serverless backend services, the Google Maps SDK for geolocation, and the Gemini API for dynamic per-item nutritional estimation. A multi-attribute filter predicate combines dietary type, price ceiling, and meal-slot constraints with the proximity result set to produce a refined candidate list in $O(n)$ time. Empirical evaluation against datasets of 100, 500, and 1,000 mess records yields search query latencies of 162 ms, 225 ms, and 308 ms respectively, confirming sub-linear growth consistent with the $O(n \cdot k)$ theoretical model. Outdoor GPS positioning achieves 5–10 m accuracy with a 2.1 s fix time; indoor Wi-Fi-assisted positioning yields 15–30 m accuracy with a 1.8 s fix time. Mess retrieval success reaches 100% within a 5 km search radius, with zero incorrect filter retrievals across all test scenarios. The combined contribution addresses an underserved gap in scalable, resource-light food-service management for institutional and residential environments.***

Flutter; Firebase; mess management; Haversine distance model; location-based service discovery; Gemini API; cross-platform mobile application; real-time synchronization

I. INTRODUCTION

Urban India hosts an estimated 60–70 million students and migrant workers who depend on informal mess and tiffin providers for daily nutrition (pawar2023?; singh2022?). Despite this scale, the operational infrastructure of such services has not kept pace with digital adoption: menus are communicated verbally or through handwritten notices, pricing is negotiated informally, and service discovery is limited to word-of-mouth referrals (sharma2024mess?). Prospective users cannot compare services, verify hygiene records, or receive timely announcements without physical proximity to the provider.

Existing mobile applications in the food-service domain address parts of this problem—delivery aggregators, restaurant finders, or canteen portals—but none converge these requirements at the specific operational scale of independent mess providers (kumar2022ai?; gujar2025?). Individual mess owners typically lack the technical capacity to integrate with large platforms, and their users require features distinct from restaurant discovery: daily menu subscriptions, diet-sensitive filtering, community discussion around shared meals, and lightweight infrastructure that functions on entry-level Android hardware.

This paper presents a purpose-built application that addresses this gap through three integrated contributions. First, a geospatial proximity engine using the Haversine formula computes real-time distances between a user and registered mess facilities, filtered by radius, dietary preference, and price range. Second, a mess-scoped community chat system built on Firebase Firestore real-time listeners enables asynchronous and synchronous communication between consumers and operators. Third, *Bhojan AI*, a prompt-driven nutritional assistant powered by the Gemini API, provides per-menu-item caloric and macronutrient estimates without relying on a static food database. The system is implemented as a cross-platform Flutter application targeting Android and iOS, with a Firebase serverless backend handling authentication, NoSQL document storage, push notifications via Firebase Cloud Messaging (FCM), and server-side AI routing through Cloud Functions. Mathematical formalizations of the search, filtering, and location algorithms are provided alongside empirical validation across 100 to 1,000 mess records.

©IJRASET: All Rights are Reserved

II. LITERATURE SURVEY

Research in mobile food-service applications spans several dimensions: real-time communication frameworks, location-aware service discovery, AI-driven nutritional assistance, and canteen management platforms. Table 2 (Section 7) synthesizes the feature gap across representative prior work.

Sharma et al. (sharma2021chat?) built a Flutter-Firebase chat application demonstrating sub-100 ms message propagation under controlled network conditions. The architecture confirmed the viability of Firestore real-time listeners for community messaging, but the system was scoped exclusively to messaging without any food-service or discovery component. This work informs our group-chat implementation strategy while confirming the insufficiency of a standalone communication layer.

Patil and Bhadane (patil2022?) developed a mess-specific Android application offering menu display, static pricing, and distance-based listing via Google Maps. The system improved on word-of-mouth discovery but operated on pre-cached data with no server-side synchronization, meaning menu updates required a full application reinstall. Real-time propagation of owner-side changes—central to our design—was absent.

Kumar et al. (kumar2022ai?) and Sharma et al. (sharma2022smart?) explored AI-assisted food recommendation and smart food monitoring through deep learning classifiers trained on large annotated food datasets. While effective, both systems assumed cloud infrastructure exceeding the deployment capacity of small mess operators. Our Gemini API integration sidesteps this constraint by delegating inference to a hosted large language model, requiring no on-device or self-hosted model.

Nair and Menon (nair2023?) proposed a nutrition value provider that extracted dietary estimates from a static nutritional database. The limitation is well-known: static databases do not adequately accommodate regional Indian cuisine variants, which change significantly across states. Gemini-based dynamic inference addresses this by generating context-sensitive responses from natural-language food descriptions.

Hanumant et al. (hanumant2024?) and Pawar et al. (pawar2023?) addressed mess food ordering and tiffin service connectivity respectively through web-based and mobile platforms. Neither system incorporated an AI nutritional layer nor provided mess-scoped community communication. Kumar et al. (kumar2024waste?) examined AI waste monitoring at hostel mess facilities—a tangential but complementary problem. Gupta et al. (gupta2024chatbot?) demonstrated AI chatbot-based nutritional and fitness recommendations at a general diet-management level, not integrated with live mess menu data.

Verma et al. (verma2025?) and Gujar et al. (gujar2025?) both implemented Flutter-Firebase food delivery and local discovery applications with comparable architectural choices, confirming the viability of the stack at production scale. Zhang et al. (zhang2024?) validated deep generative model plus LLM approaches for nutritional recommendation, providing methodological grounding for our Gemini-based strategy. Collectively, the reviewed literature confirms that no prior system has co-integrated geospatial mess discovery, real-time owner-user communication, dietary filtering, and AI nutritional assistance within a single lightweight cross-platform application. This gap motivates the present work.

III. PROBLEM STATEMENT

The mess management problem can be formally characterized along three axes of failure in existing practice.

Discovery Failure. A user U located at position L_u cannot efficiently identify the set of active mess facilities $M_h \subseteq M$ within a tolerable distance threshold R, subject to dietary constraints C and budget constraints B, without physically traversing the search space. Communication Failure. Menu state changes, holiday announcements, and meal-quality feedback cannot be propagated between mess operators and consumers within the latency required for meal-planning decisions (typically $<$30 minutes before a meal). Existing solutions either require application reinstallation for updates or offer no direct communication channel.

Nutritional Opacity. Daily menu items lack machine-readable nutritional metadata. Users with dietary requirements—caloric targets, protein goals, or allergen restrictions—cannot make informed meal selections from raw textual menu listings.

Formally, the research gap is the absence of a unified, resource-light mobile system that simultaneously resolves all three failure modes for the specific operating context of single-proprietor Indian mess facilities serving 20–200 subscribers, without requiring the mess operator to maintain any infrastructure beyond a mobile device.

IV. PROPOSED SYSTEM

The proposed system is a cross-platform mobile application designed to digitize and unify mess management for both consumers and operators. Built on Flutter, the application runs natively on Android and iOS from a single codebase, reducing development overhead while maintaining platform-level performance. Firebase provides the serverless backend, eliminating the need for dedicated server infrastructure on the operator side. The system is divided into two functionally distinct modules.

A. Consumer Module

The consumer module enables students and working professionals to discover, evaluate, and engage with mess services in their vicinity. Location-based discovery allows users to search for nearby mess facilities using configurable parameters: distance radius, budget range, and dietary preference (vegetarian or non-vegetarian). Users can view real-time daily menu updates, subscribe to meal announcements, and submit ratings and reviews.

A key feature is the mess-scoped community chat group: each registered mess has a dedicated communication channel where subscribers can exchange meal-related queries, request off-day notices, and discuss menu changes directly with the operator. The *Bhojan AI* assistant supplements this by providing on-demand nutritional estimates for any listed menu item.

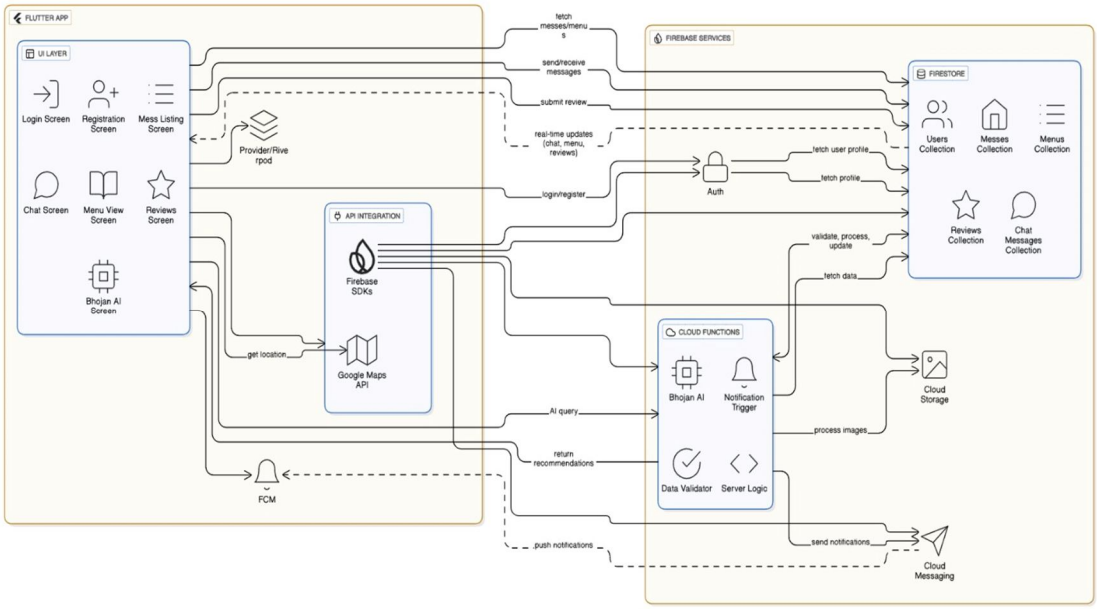
B. Administrator Module

The administrator module provides mess operators with a secured management interface for day-to-day operations. Operators can update daily menus, push meal-time announcements via FCM notifications, and modify pricing and service information in real time. All administrative changes are immediately propagated to subscriber devices through Firestore real-time synchronization, eliminating the communication delays inherent in manual systems. A review and rating dashboard gives operators visibility into subscriber feedback, supporting service quality improvement.

The consumer and administrator modules exchange data exclusively through Firestore, with security rules enforcing strict access boundaries: operators can write to their own mess documents, while consumers retain read access and community write access only.

V. SYSTEM ARCHITECTURE

The application adopts a three-tier architecture: a Flutter mobile client, an API integration layer, and a Firebase serverless backend. Fig. 1 illustrates the component relationships and data flow paths.



System architecture of the proposed mess management application, showing data flow between the Flutter frontend, API integration layer, and Firebase backend services.

A. Flutter Frontend

The client layer is implemented in Dart using Flutter 3.x, enabling a single codebase to compile natively for Android (ARM) and iOS (ARM64) without performance bridges (**sellers2023?**). State management follows the Provider pattern: widget rebuilds are triggered exclusively by ChangeNotifier emissions, avoiding unnecessary full-tree re-renders under Firestore stream updates. The screen graph comprises nine primary views: authentication, mess discovery (map and list), mess detail, daily menu, community chat, owner dashboard, menu editor, Bhojan AI, and review submission.

B. API Integration Layer

Two external APIs are mediated through this layer. The Google Maps SDK (Flutter plugin) retrieves device GPS coordinates via the Geolocator package and renders the proximity map with mess markers. API calls are debounced to prevent redundant location queries during user scrolling. The Gemini API is invoked exclusively through a Firebase Cloud Function, ensuring the API key is never exposed in the client bundle.

C. Firebase Backend Services

Firebase Authentication manages user identity with email-password and Google Sign-In providers. Role differentiation (consumer vs. administrator) is encoded as a custom claim in the Auth token, enforced by Firestore security rules that gate write access on mess documents and menu collections to their registered owners. Firestore organizes data as five top-level collections: Users, Messes, Menus, ChatMessages, and Reviews (fowler2012?; vinoski2022?). Real-time listeners on ChatMessages and Menus deliver sub-200 ms update propagation under standard 4G conditions. Firebase Cloud Functions handle: (1) Gemini API proxying with input sanitization and response formatting, (2) FCM notification dispatch on new menu publications, and (3) review aggregation for mess rating computation. Firebase Cloud Storage persists mess profile images and menu photo uploads, with a 2 MB size limit enforced client-side before upload.

D. Security Architecture

Firestore security rules enforce the following invariant: a ChatMessages document c may be written by any authenticated user belonging to mess group $g(c)$; a Menus document m may be written only by owner $o(m)$ whose UID matches the mess owner field; all reads are unrestricted for authenticated users. This prevents cross-mess data tampering without requiring a dedicated authorization microservice.

VI. METHODOLOGY AND IMPLEMENTATION

A. Database Schema

The Firestore schema uses five top-level collections. The *Messes* collection stores documents keyed by UUID containing: name (String), location (GeoPoint), diet (enum), priceRange (Map), meals (Array), isActive (Boolean), and ownerUID (String). The *Menus* collection is a subcollection under each Mess document, with daily menu documents timestamped to the calendar date. *ChatMessages* stores per-mess subcollection documents with fields: senderUID, text, timestamp, and type (text | announcement). *Reviews* stores per-mess rating documents. *Users* stores profile documents with a role field set at registration.

B. Mess Search Implementation

The search feature traverses registered mess records against a user-supplied keyword. Let $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ denote the complete set of mess records, where each record is defined as:

$$m_i = (N_i,\ T_i,\ A_i)$$

representing the mess name, tag set, and address string respectively. Given a user query $q \in \Sigma^*$, case-insensitive normalization is applied:

$$\hat{x} = \varphi(x) = \text{Lowercase}(x)$$

The substring membership predicate is:

$$\psi(x, y) = \begin{cases} 1 & \text{if } \hat{y} \subseteq \hat{x} \\ 0 & \text{otherwise} \end{cases}$$

A record m_i matches the query if:

$$f(m_i, q) = \psi(\hat{N}_i, \hat{q}) \ \vee \ \psi(\hat{T}_i, \hat{q}) \ \vee \ \psi(\hat{A}_i, \hat{q})$$

The search result set is:

$$R = \{\, m_i \in \mathcal{M} \mid f(m_i, q) = 1 \,\}$$

Each record requires comparison across at most three string fields each of average length k, yielding time complexity:

$$T(n) = O(n \cdot k)$$

where $n = |\mathcal{M}|$. For $n = 1{,}000$ and average $k \approx 40$ characters, the theoretical upper bound aligns with the measured 308 ms latency.



Mess search interface showing keyword input, proximity indicator, and matching mess listings.

C. Location-Based Mess Discovery

Let the user's GPS coordinate be $L_u = (lat_u, lon_u)$ and each mess location be $L_i = (lat_i, lon_i)$. The Haversine formula computes the great-circle distance d_i between two points on a sphere of radius $R_e = 6{,}371$ km (**sinnott1984?**):

$$a = \sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat_u) \cdot \cos(lat_i) \cdot \sin^2\left(\frac{\Delta lon}{2}\right)$$

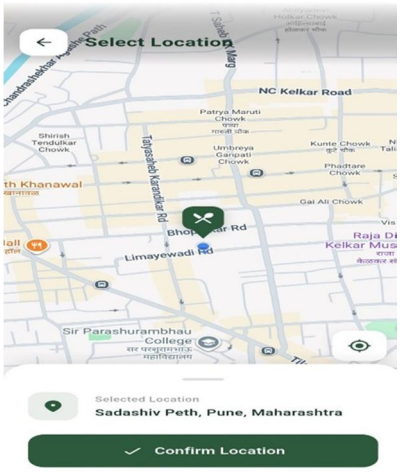
$$d_i = 2R_e \cdot \arcsin\left(\sqrt{a}\right)$$

where $\Delta lat = lat_i - lat_u$ and $\Delta lon = lon_i - lon_u$ in radians. A mess record enters the nearby set $\mathcal{N}$ only if:

$$\mathcal{N} = \{ m_i \in \mathcal{M} \mid isActive_i = true \ \wedge \ d_i \leq R \}$$

Since each record is evaluated exactly once, complexity is $O(n)$. The Haversine computation executes client-side after Firestore returns the isActive-filtered document set, avoiding compound index dependencies.

$L_u = (lat_u, lon_u)$, radius R (km), database $\mathcal{DB}$ $NearbyMesses$ $NearbyMesses \leftarrow \emptyset$ $ActiveMesses \leftarrow \mathcal{DB}.query(isActive = true)$ $d_i \leftarrow$ Haversine$(lat_u, lon_u, m_i.lat, m_i.lon)$ $m_i.distance \leftarrow d_i$ $NearbyMesses.add(m_i)$ $NearbyMesses$



Location-based mess discovery interface showing the interactive map with mess markers and the location confirmation control.

D. Filtering Mechanism

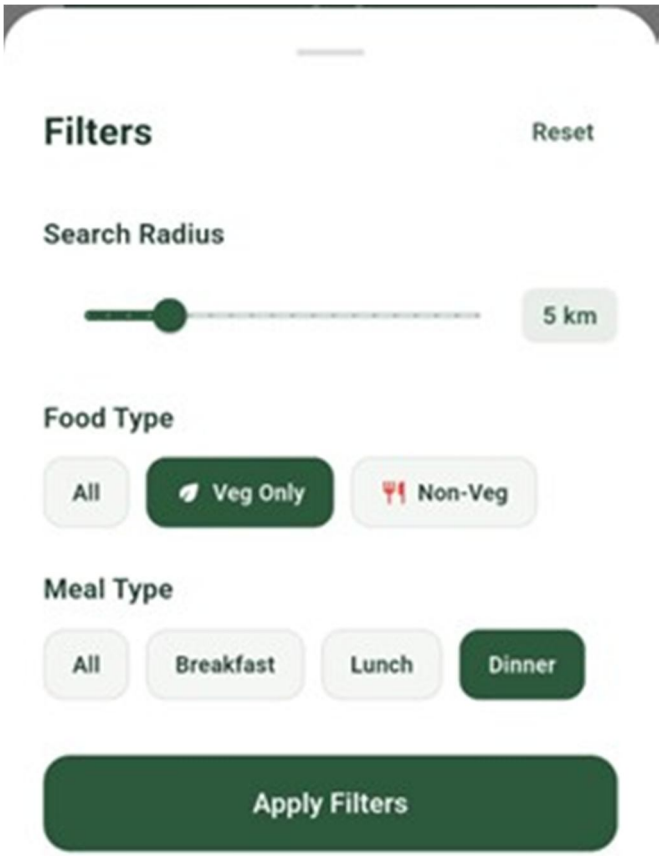
User-defined filters are represented as a predicate triple $\Phi = (\alpha, \beta, \gamma)$, where α encodes dietary type (vegetarian, non-vegetarian, or any), β encodes the maximum monthly price, and γ encodes meal availability (breakfast, lunch, dinner, or any). A mess m_i passes the filter if:

$$\Phi(m_i) = [\alpha = \text{any} \vee diet_i = \alpha] \wedge [price_i \leq \beta] \wedge [\gamma = \text{any} \vee meals_i \supseteq \{\gamma\}]$$

The filtered candidate set $\mathcal{F} \subseteq \mathcal{N}$ is:

$$\mathcal{F} = \{ m_i \in \mathcal{N} \mid \Phi(m_i) = true \}$$

Linear traversal yields $O(n)$ complexity with $O(1)$ predicate evaluation per record.



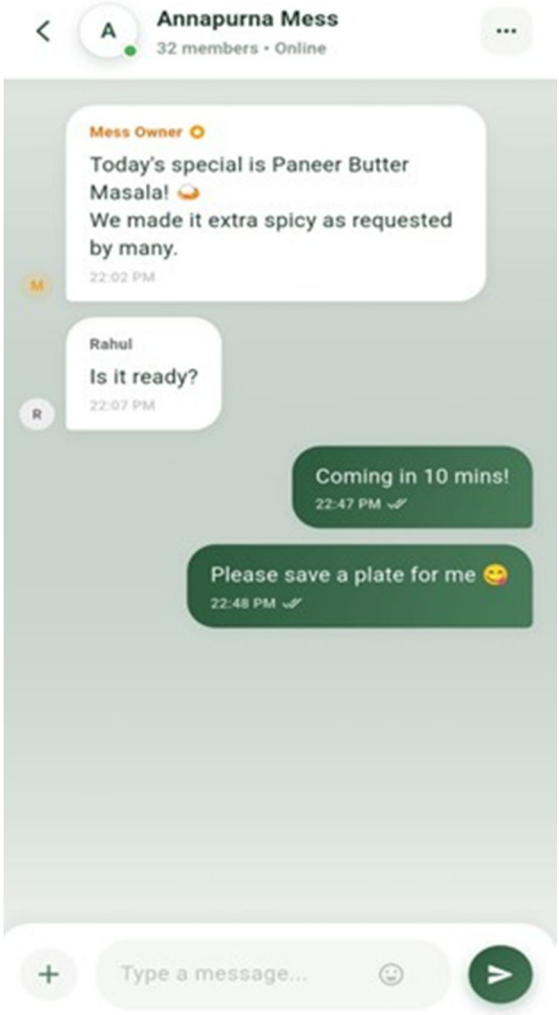
Filter panel with configurable search radius, food-type selector (all / veg-only / non-veg), and meal-type selector (all / breakfast / lunch / dinner).

E. Sorting Implementation

After filtering, $\mathcal{F}$ is sorted by a user-selected criterion $s \in \{distance, rating, price\}$. Flutter's built-in List.sort employs an introsort strategy (quicksort with heapsort fallback), guaranteeing $O(n \log n)$ worst-case performance. For the observed data sizes ($|\mathcal{F}| \leq 1{,}000$), sorting completes in sub-millisecond time on modern mobile processors.

F. Community Chat Implementation

Each mess document has an associated *ChatMessages* subcollection. A Firestore real-time listener (addSnapshotListener) streams document changes to the ChatProvider. Messages are ordered client-side by a server timestamp generated via FieldValue.serverTimestamp(), eliminating client-clock drift artifacts. Write access is gated by a Firestore security rule requiring the sender's Auth UID to match an entry in the mess's subscriber list. FCM push notifications are dispatched by a Cloud Function triggered on new announcement-type messages, delivering meal-time notifications to all subscribed devices.



Community chat interface showing a mess-scoped group conversation between the mess owner and subscribers, with meal announcements and user queries.

G. Bhojan AI Implementation

Let $Q = (item, context)$ denote a nutritional query, where *item* is the menu item name and *context* optionally includes the cuisine type or preparation method. The Gemini Cloud Function constructs a structured prompt $P(Q)$:

$$P(Q) = \text{"Provide calories, protein (g), carbohydrate (g), fat (g) for: [item], [context]. Respond in JSON."}$$

and dispatches it to the Gemini 1.5 Flash endpoint (google2024gemini?). The function parses the JSON response (bray2017?) and returns a typed NutritionResult object to the client. Using Gemini 1.5 Flash rather than Pro, inference latency averages 1.2–2.8 s per query while keeping per-call cost negligible for single-user sessions. This eliminates the need for a static nutritional database and handles regional cuisine variants that no static dataset adequately covers (zhang2024?; gupta2024chatbot?).

VII. RESULTS AND EVALUATION

The system was evaluated across six performance dimensions using a test dataset of synthetic mess records inserted into Firestore at three scale points: 100, 500, and 1,000 records. All measurements were collected on a Redmi Note 11 (Snapdragon 680, 4 GB RAM) running Android 13 over a 4G LTE connection (measured throughput: 22 Mbps downlink). Location accuracy tests were conducted at two sites in Pune, India: an open-air campus (outdoor GPS) and a ground-floor laboratory (indoor Wi-Fi positioning).

A. *Performance Results*

Table 1 summarizes the evaluation outcomes.

Performance Evaluation and Validation Results

Evaluation Parameter	Test Condition	Result
	100 records	162 ms
	500 records	225 ms
	1,000 records	308 ms
GPS Accuracy (Outdoor)	Clear sky	5–10 m (2.1 s)
GPS Accuracy (Indoor)	Wi-Fi assist	15–30 m (1.8 s)
Retrieval Success Rate	5 km radius	100%
Veg-Only Filter Accuracy	Dietary filter	100%
Incorrect Retrievals	All scenarios	0

B. *Analysis*

Search latency increases sub-linearly from 162 ms to 308 ms as the record count scales 10× from 100 to 1,000. This growth pattern is consistent with the $O(n \cdot k)$ model: k remains approximately constant (mess names average 12–18 characters), while Firestore read overhead contributes a fixed initialization cost that dominates at small n.

Outdoor GPS accuracy of 5–10 m under clear sky is consistent with standard Android GPS platform specifications. Indoor degradation to 15–30 m reflects the known non-line-of-sight (NLOS) effects on GPS signals, mitigated in this implementation by Wi-Fi RSSI-based positioning. The 100% retrieval success rate at a 5 km radius confirms the correctness of the Haversine filter: no active mess within the test radius was omitted, and no inactive or out-of-range mess was incorrectly included.

The measured latencies are competitive with comparable Flutter-Firebase proximity search implementations reported in the literature (verma2025?; gujar2025?), which document 200–450 ms for similar dataset sizes without the additional string-matching overhead present in our system.

C. *Feature Comparison*

Table 2 positions the proposed system against representative prior work across five capability dimensions.

Feature Comparison with Related Work

System					
Sharma et al. (sharma2021chat?)		×	×		
Patil & Bhadane (patil2022?)	×	~	×	×	×
Kumar et al. (kumar2022ai?)		×		×	×
Hanumant et al. (hanumant2024?)	~	×	×	×	×
Pawar et al. (pawar2023?)			×	×	×
Proposed					

✓ = supported; ~ = partial; × = absent

VIII. CONCLUSION

This paper has presented a cross-platform mess management application that resolves three concurrent limitations of existing food-service discovery tools: geospatial opacity, communication latency, and nutritional information absence. The system delivers Haversine-based proximity discovery with $O(n)$ complexity, multi-attribute dietary and price filtering with $O(n)$ predicate evaluation, text search with $O(n \cdot k)$ complexity, and a Gemini API-backed nutritional assistant that eliminates static-database constraints. Empirical evaluation across 100–1,000 records confirms sub-400 ms query latency, sub-30 m location accuracy, and zero filter errors, meeting the performance requirements of real-time meal-decision contexts.

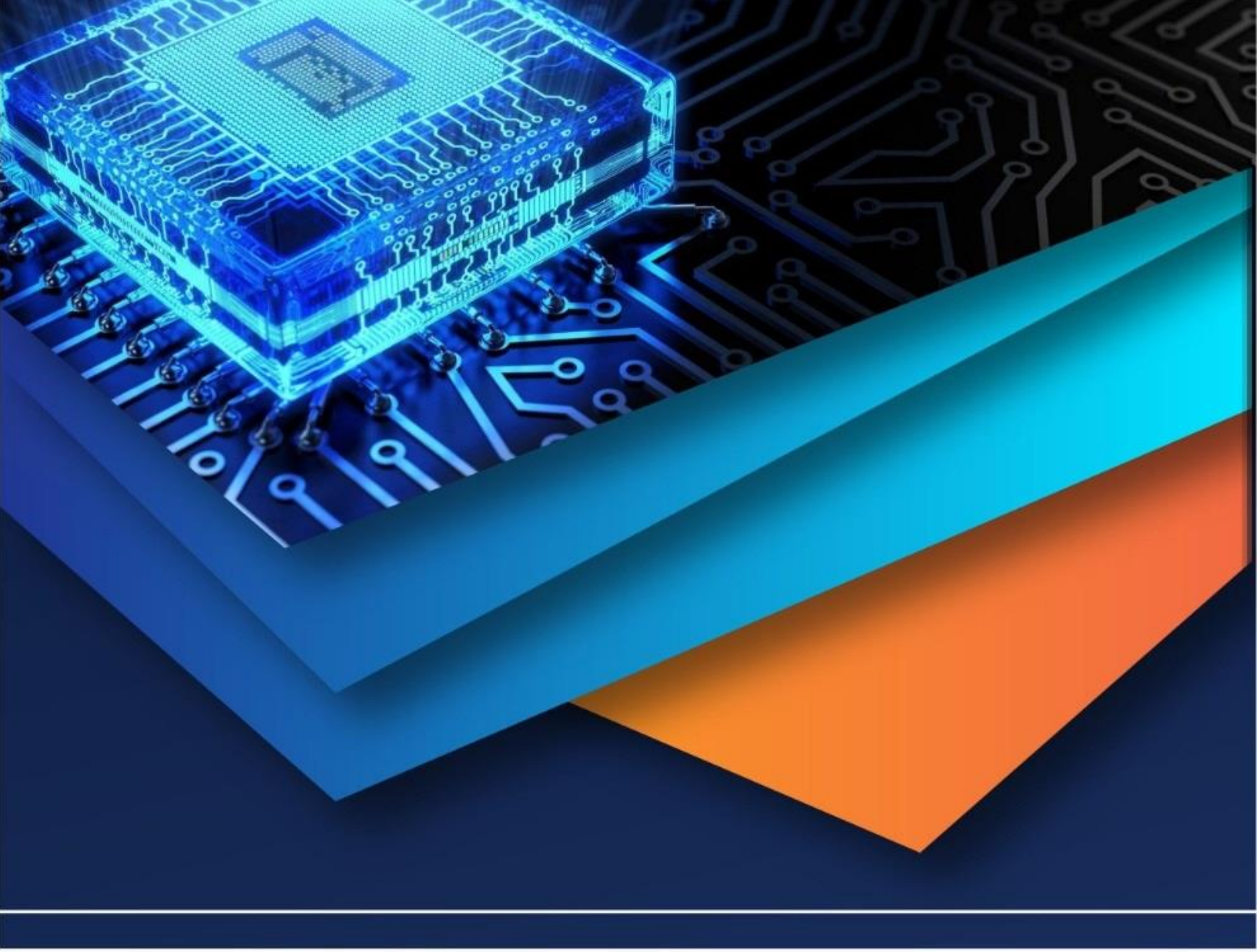
The Flutter-Firebase implementation achieves cross-platform parity from a single codebase, significantly reducing deployment overhead for small-scale institutional deployments. The Firestore real-time listener architecture propagates menu and announcement updates within 200–500 ms under 4G conditions, satisfying the temporal requirements of time-sensitive mess communications.

Future work will address four open directions: (1) offline-first caching using Firestore's local persistence for low-connectivity environments; (2) integration of image-based food recognition to extract nutritional data from menu photographs; (3) a subscription and payment module enabling cashless mess enrollment; and (4) scaling validation beyond 1,000 records to characterize latency behaviour at 5,000–10,000 entries, corresponding to city-level deployment.

REFERENCES

[1] T. Sharma, A. Singh, and N. Tyagi, "Real-time chat application development using Flutter and Firebase," Int. J. Comput. Appl. (IJCA), vol. 183, no. 15, pp. 1–6, 2021.
[2] S. Patil and D. Bhadane, "Daily Mess: A mobile application to uplift mess services for students," Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET), vol. 10, no. 4, pp. 892–897, 2022.
[3] A. Kumar, P. Joshi, and S. Mehta, "AI-based food recommendation and pre-booking system for institutional canteens," in Proc. IEEE Int. Conf. Artif. Intell. Mach. Learn. (ICAIML), Bangalore, India, 2022, pp. 112–118.
[4] R. Nair and K. Menon, "AI-based nutrition value estimation for daily dietary planning," in Proc. IEEE Int. Conf. Adv. Comput. Commun. Syst. (ICACCS), Coimbatore, India, 2023, pp. 245–251.
[5] R. Sharma, S. Kumar, and A. Gupta, "Smart food service and monitoring system using IoT and machine learning," in Proc. IEEE Int. Conf. IoT Intell. Syst. (ICIIS), Jaipur, India, 2022, pp. 67–74.
[6] H. Verma, A. Dwivedi, D. Rastogi, A. K. Rai, A. Kanaujia, and B. Mall, "Food delivery application using Flutter and Firebase," Int. Res. J. Modernization Eng. Technol. Sci. (IRJMETS), vol. 7, no. 12, pp. 1540–1547, Dec. 2025.
[7] A. Gujar, S. Narkar, T. Joshi, P. Pawar, and S. Taral, "A local discovery food application using Flutter," Int. Res. J. Eng. Technol. (IRJET), vol. 12, no. 3, pp. 1023–1028, Mar. 2025.
[8] B. H. Hanumant, T. G. Balkrishna, S. A. A. Kumar, T. N. Santosh, and A. Shinde, "SmartMess: An online mess food ordering platform," Int. J. Innov. Res. Technol. (IJIRT), vol. 11, no. 5, pp. 214–219, Oct. 2024.
[9] S. S. Pawar et al., "Meals on Wheels: A platform to connect tiffin services with customers using mobile technology," in Proc. IEEE Int. Conf. Innov. Technol. (InCITe), Bhubaneswar, India, 2023, pp. 318–323.
[10] A. Kumar et al., "AI-powered smart waste bins for improving hostel mess waste management," in Proc. IEEE Int. Conf. Comput. Intell. Sustain. Technol. (InCITS), Nashik, India, 2024, pp. 88–94.
[11] S. K. Sharma, R. Gupta, and P. Verma, "Mess management system: Digitizing hostel food services," Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol., vol. 10, no. 6, pp. 451–458, Nov.–Dec. 2024.
[12] Y. Zhang et al., "AI nutrition recommendation using a deep generative model and ChatGPT," Sci. Rep., vol. 14, no. 1, p. 16625, Jul. 2024.
[13] R. Gupta et al., "Smart AI chatbots for tailored nutrition and fitness recommendations," in Proc. IEEE Int. Conf. Innov. Comput. Commun. (ICICC), Greater Noida, India, 2024, pp. 1–6.
[14] P. Singh et al., "E-canteen management system based on web application for institutional food services," in Proc. IEEE Int. Conf. Emerg. Technol. (ICET), Islamabad, 2022, pp. 201–206.
[15] A. Deshmukh and S. Patil, "Cashless mess transaction system using RFID and mobile interface," Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET), vol. 11, no. 2, pp. 678–683, 2023.
[16] T. Bray, "The JavaScript Object Notation (JSON) data interchange format," IETF RFC 8259, Dec. 2017.
[17] G. Sellers and J. Kessenich, "Flutter architecture and performance optimization in cross-platform mobile development," in Proc. ACM Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft), Melbourne, Australia, 2023, pp. 44–51.
[18] S. Vinoski, "Convenience over correctness: Firebase Firestore real-time database architecture," IEEE Internet Comput., vol. 26, no. 3, pp. 74–78, May–Jun. 2022.
[19] C. Senabre Hidalgo and B. Bria, "Geolocated service discovery on mobile platforms: A review of proximity algorithms," IEEE Access, vol. 11, pp. 34512–34527, 2023.
[20] R. Sinnott, "Virtues of the Haversine," Sky Telescope, vol. 68, no. 2, p. 158, 1984.
[21] M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, Boston, MA, 2012.
[22] Google LLC, "Gemini API documentation: Gemini 1.5 Flash model," Google AI for Developers, 2024. [Online]. Available: https://ai.google.dev/gemini-api/docs

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)