

Grama-Vasathi: A Cloud-Connected Android Application for Rural Hospitality and Agri-Tourism Digitization

Harshita Krishnamurthy¹

¹Department of Computer Science & Engineering, Bangalore Institute of Technology, Bengaluru-560004

Abstract - Rural tourism in India holds enormous untapped potential, yet the sector remains fragmented due to lack of digital infrastructure, poor awareness of hospitality standards, and absence of organized booking systems. This paper presents Grama-Vasathi, an Android application built to bridge the gap between rural host families and urban travelers seeking authentic agri-tourism experiences. The system employs the Model-View-ViewModel (MVVM) architectural pattern and leverages Firebase Firestore, Firebase Authentication, and Firebase Storage as its cloud backend. The application delivers a Host Readiness Checklist with dynamic score calculation, a searchable Farm-Stay Catalog with activity-based filtering, a simulated booking system, and a Cultural Guide for guests. Built using Kotlin, Jetpack Compose, Kotlin Coroutines, and Dagger Hilt, the application achieves a responsive and offline-capable user experience. Results confirm successful fulfillment of all Must-Have requirements, with real-time score updates, reliable cloud synchronization, and accessible UI across Android API 28 and above.

Key Words: Android development, MVVM architecture, Firebase Firestore, Jetpack Compose, agri-tourism, rural hospitality, Kotlin Coroutines.

1. INTRODUCTION

India has a remarkably rich tapestry of agricultural heritage, village culture, and local traditions—yet most of this remains invisible to urban travelers simply because there is no organized way to discover or book authentic rural experiences. The agri-tourism sector, though growing, continues to suffer from a last-mile digital gap: the infrastructure that large hotel chains enjoy simply does not reach farming families in small villages.

The problem plays out on both sides. A village host like Smt. Kamala Devi, who has spare rooms and genuine warmth to offer, has no framework for evaluating her own hospitality preparedness. On the other side, a traveler like Arjun Mehta—an IT professional from Bengaluru who wants to experience organic farming or birdwatching—has no reliable, verified platform to find such stays.

This paper presents Grama-Vasathi (from the Kannada words for 'village stay'), an Android application developed during a structured internship at MindMatrix.io as part of the VTU Android Development Programme. The application was built using Kotlin, Jetpack Compose, Firebase cloud services, and the MVVM architectural pattern, covering the full development lifecycle from requirements analysis through implementation and testing.

2. BACKGROUND AND MOTIVATION

Agri-tourism platforms in European markets—like Italy's Agriturismo network—have demonstrated that rural experiences can be effectively organized and commercialized. India presents a different set of constraints: lower digital literacy among rural hosts, language diversity, infrastructure variability, and a wider gap between urban consumer expectations and rural service delivery norms.

Existing travel platforms like MakeMyTrip or Airbnb do list some rural properties, but they are not tailored for Indian

village hospitality. There is no concept of a Host Readiness Score, no checklist system to help a rural host understand whether their accommodation meets the hygiene and comfort standards that city guests expect. This specific gap is what Grama-Vasathi addresses.

Firebase Firestore was chosen for its real-time synchronization and flexible NoSQL document model. Jetpack Compose was selected for its declarative UI model that aligns naturally with MVVM, and Kotlin Coroutines were used to keep all cloud operations off the main thread.

3. SYSTEM ARCHITECTURE

The Grama-Vasathi application follows the Model-View-ViewModel (MVVM) architectural pattern, organized into three primary layers as shown in Fig. 3.1. This layered separation keeps the codebase modular, testable, and extensible for future enhancements like multilingual support or AI-powered recommendations.

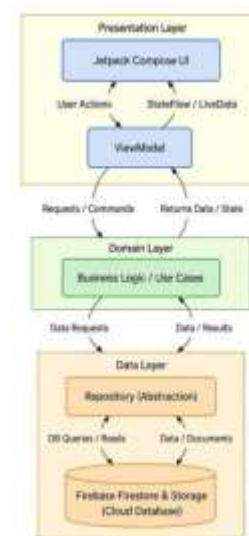


Figure 3.1: System Architecture

Fig - 3.1: System Architecture

3.1 Presentation Layer

All screens—Host Dashboard, Farm-Stay Catalog, Booking Screen, Checklist Wizard, and Cultural Guide—were built using Jetpack Compose. Each screen connects to a dedicated ViewModel managing UI state via Kotlin StateFlow, so changes like checklist updates or booking confirmations reflect instantly on screen without manual refreshes.

3.2 Domain Layer

The Domain Layer houses business rules in dedicated Use Case classes: the Host Readiness Score calculation, activity-based filtering logic for the Catalog, booking validation, and checklist evaluation. Isolating these operations keeps them independent from UI and data concerns, making them straightforwardly testable in isolation.

3.3 Data Layer

Repository classes (FarmStayRepository, HostRepository, BookingRepository) serve as the single source of truth,

abstracting all Firebase operations. Firebase Firestore stores host profiles, farm-stay listings, booking records, and checklist states; Firebase Storage handles image galleries. A local Room Database provides offline caching for rural areas with intermittent connectivity. A critical constraint enforced throughout: no Firestore method was ever called directly from a Composable function. All data access routes strictly through $\text{ViewModel} \rightarrow \text{Repository} \rightarrow \text{Firebase}$.

4. SYSTEM FEATURES AND FUNCTIONAL REQUIREMENTS

The functional scope was defined in a formal Software Requirements Document (SRD) with requirements categorized as Must-Have (M), Should-Have (S), and Could-Have (C). The following subsections describe the primary feature groups.

4.1 Host Registration and Profile Management

Village hosts register using name, phone/email, village name, and state. Firebase Authentication creates a secure session and a host profile document is created in the Firestore 'hosts' collection. Hosts can subsequently update their profile with spare room count, a farm description, and a multi-select amenities list—clean sheets, safe water, Western toilet, hot water, and Wi-Fi.

4.2 Host Training Checklist and Readiness Score

The Guest Readiness Checklist presents hosts with at least ten binary checkbox items covering room cleanliness, sanitation, guest comfort, and safety. The Host Readiness Score is calculated as $(\text{checked items} \div \text{total items}) \times 100$ and updates within one second of any toggle. Scores display with contextual labels: 0–49% shows 'Needs Improvement' in red, 50–79% shows 'Getting Ready' in amber, and 80–100% shows 'Guest Ready' in green.

The checklist is presented as a multi-step Wizard with a progress indicator, and all state is persisted in both Firestore and the local Room database so progress survives connectivity loss or unexpected app closure.

4.3 Farm-Stay Catalog and Activity Discovery

The Guest Catalog displays farm-stay listings from Firestore showing farm name, village, Host Readiness Score, activities, and thumbnail image. Guests filter listings using activity chips—Cow Milking, Field Plowing, Local Cooking, Birdwatching, Pottery, Organic Farming—with multiple simultaneous filters applying AND logic updating within 500 milliseconds. Tapping a listing opens a detail screen with full farm description, image gallery (minimum three photos), activities with icons, and a 'Book a Room' button.

4.4 Booking System

The simulated booking system accepts check-in and check-out dates via calendar picker and a guest count of 1–6. Validation rejects past check-in dates and check-out dates before check-in. On successful submission, a PENDING booking document is created in Firestore and a confirmation screen displays all details along with a note that the host will contact the guest to confirm.

4.5 Cultural Guide

The Cultural Guide is accessible from both the Catalog and Farm-Stay Detail screens. It covers five sections: how to greet locals, appropriate dress, food etiquette, cultural dos and don'ts, and common local phrases. This directly addresses the discomfort many urban travelers feel about rural customs, removing a real psychological barrier to booking.

5. IMPLEMENTATION

Development followed a five-phase SDLC: requirements engineering, architecture planning, core implementation, performance optimization, and version control management. The technology stack was selected to meet the offline-first, cloud-connected requirements of the application.

Technology	Role
Kotlin	Primary language; null safety, concise syntax
Jetpack Compose	Declarative UI for all screens
Firebase Firestore	Cloud NoSQL database
Firebase Auth	Secure user login/registration
Firebase Storage	Farm-stay image galleries
Kotlin Coroutines	Async Firebase operations
Dagger Hilt	Dependency injection
Room Database	Offline checklist cache
Git & GitHub	Version control & branching

Table - 1: Technology Stack

5.1 Data Layer Implementation

Firebase services were configured first, establishing Firestore collections for 'hosts', 'farmstays', 'bookings', and 'checklist_states' with structured document relationships. Firebase Authentication was set up with separate flows for hosts and guests. Repository classes encapsulate all Firebase operations, exposing reactive StateFlow streams to ViewModels. The Room database uses a write-local-then-sync-to-Firestore strategy for offline resilience.

5.2 Domain and Presentation Implementation

The Host Readiness Score calculation lives in ChecklistViewModel, updating score state on every checkbox toggle. Dagger Hilt was configured to provide all ViewModels, repositories, and Firebase service instances, eliminating manual dependency creation. All screens were built with Jetpack Compose using a warm, earthy color palette suited to the agricultural context. The multi-step Checklist Wizard, farm-stay activity filter chips, and reactive catalog updates were implemented using Compose's state-driven rendering model.

6. RESULTS AND EVALUATION

The development phase concluded with a functional Minimum Viable Product (MVP) satisfying all Must-Have requirements from the SRD. The Guest Farm-Stay Catalog loaded all listings within 2,000 milliseconds for up to 30 records (NFR-PERF-01 passed). The Host Readiness Score consistently updated within 1,000 milliseconds of any checkbox toggle (NFR-PERF-02 passed). Checklist state was fully preserved across force-closes and restarts (NFR-RELY-01 passed). The application ran without crashes on Android 9.0 API 28 with all UI elements rendering correctly (NFR-PORT-01 passed).

Activity filter chips updated the catalog within 500 milliseconds, with multiple simultaneous filters applying correct AND logic. Booking validation correctly rejected invalid date combinations, and Firestore booking documents were created with PENDING status on each successful submission. The color-coded Host Readiness Score labels were specifically designed to be interpretable without English literacy, satisfying the usability requirement NFR-USAB-01.

7. APPLICATIONS AND BROADER IMPACT

For rural host families, the application provides a dignified digital presence—a way to market spare rooms and agricultural experiences to a much wider audience. The Host Readiness Score doubles as an educational tool, guiding hosts toward the hygiene and hospitality standards urban guests expect.

For urban travelers, the catalog and cultural guide reduce the friction and uncertainty that deters city dwellers from rural stays. The Firestore schema and MVVM architecture are designed to accommodate multilingual support, additional activity categories, and regional customization with minimal structural changes. Future enhancements could include real payment integration, GPS-based discovery, AI-powered recommendations via Gemini API, and live booking confirmation.

8. CONCLUSIONS

Grama-Vasathi demonstrates that thoughtfully engineered mobile applications can meaningfully address real socio-economic challenges. By combining MVVM architecture, Firebase cloud services, and Jetpack Compose, the application delivers a platform useful to a village host evaluating her hospitality readiness and to an urban traveler searching for an authentic agri-tourism experience.

The MVVM pattern, when implemented with strict discipline—no business logic in Composables, all data access through `ViewModel` → `Repository` → `Firebase`—produces a codebase that is genuinely maintainable and testable. The successful MVP validates the project objectives and demonstrates that a student developer, given a well-structured environment and clear requirements, can build a functionally complete, architecturally sound cloud-connected Android application.

ACKNOWLEDGEMENT

The author thanks Dr. Savitha S.K., Professor, Dept. of CS&E, Bangalore Institute of Technology, for guidance as internal guide. Sincere appreciation is also due to Mr. Tirumal Mutalikdesai, Program Coordinator, MindMatrix.io (CL Infotech Pvt. Ltd.), for mentorship throughout the internship. Thanks are also extended to Dr. Suneetha K R, Prof. & Head, Dept. of CS&E, BIT, and Dr. Vijaya Prakash A M, Principal, BIT, for institutional support. This work was carried out under the VTU Internship Programme (BCS803), academic year 2025–26.

REFERENCES

- [1] Android Developers, "Guide to app architecture," Android Developers, 2024. [Online]. Available: <https://developer.android.com/topic/architecture>.
- [2] Android Developers, "Save data in a local database using Room," Android Developers, 2024. [Online]. Available: <https://developer.android.com/training/data-storage/room>.
- [3] Kotlin Foundation, "Coroutines basics," Kotlin Documentation, 2024. [Online]. Available: <https://kotlinlang.org/docs/coroutines-basics.html>.
- [4] Android Developers, "Dependency injection with Hilt," Android Developers, 2024. [Online]. Available: <https://developer.android.com/training/dependency-injection/hilt-android>.
- [5] Android Developers, "State and Jetpack Compose," Android Developers, 2024. [Online]. Available: <https://developer.android.com/jetpack/compose/state>.
- [6] Google Firebase, "Firestore Documentation," Google, 2024. [Online]. Available: <https://firebase.google.com/docs/firestore>.