iFarmAssist

Project Progress Report (Phase 1 - 4)

Status: Completed & Verified

Date: December 26, 2025

1. Executive Summary

The core foundation of the **iFarmAssist** system has been successfully implemented and verified. The system now features a fully functional **Hybrid Architecture** connecting a React Native mobile application to a Python FastAPI backend, integrated with Google's Gemini AI and a RAG (Retrieval-Augmented Generation) pipeline.

2. Phase 1: Environment & Backend Setup

Objective: Establish a scalable project structure and backend server.

- Established folder structure separating `mobile/` and `backend/`.
- Initialized **FastAPI** server with virtual environment.
- Configured `uvicorn` server binding to `0.0.0.0` for external access.
- Implemented `.env` security for API keys.

3. Phase 2: Core Services & Database

Objective: Enable data persistence and external integrations.

- **Database:** Configured **SQLAlchemy** ORM with SQLite (Dev) / PostgreSQL (Prod) compatibility.
- **Schema:** Designed models for `Users`, `Queries`, `Escalations`, and `Feedback`.
- **Authentication:** Implemented JWT (JSON Web Token) infrastructure.
- **Vector DB:** integrated **ChromaDB** for storing agricultural knowledge.

4. Phase 3: AI Engine Implementation

Objective: Build the 'Brain' of the system using RAG.

- **RAG Pipeline:** Created `vector_db.py` to ingest PDFs and retrieve relevant context based on user queries.
- **Gemini Integration:** Integrated `gemini-2.0-flash-exp` model via Google GenAI SDK.

- **Context Aggregator:** Implemented logic to inject **User Profile** (Location/Crops) and **Live Weather** (OpenWeatherMap API) into the AI prompt.
- **Resilience:** Implemented 'Smart Retry' logic to handle API Rate Limits (429 Errors).

5. Phase 4: Mobile App Foundation

Objective: Create the farmer-facing interface.

- **Framework:** Initialized **React Native (Expo)** project.
- **UI/UX:** Developed `LoginScreen` and `HomeScreen` with responsive layout.
- **Networking:** Configured `axios` client with dynamic IP handling (auto-detects backend IP).
- **Error Handling:** Implemented user-friendly error messages (e.g., 'Server Busy' instead of crash dumps).
- **Result:** App successfully sends text queries and displays AI responses.

6. Technical Stack Summary

Component	Technology Used
Frontend	React Native, Expo, Axios
Backend	FastAPI, Python, Uvicorn
AI / LLM	Google Gemini 2.0 Flash
Vector Store	ChromaDB, Sentence-Transformers
Database	SQLite (Dev) / PostgreSQL

Conclusion

The project is currently ahead of schedule with a stable, working prototype. The core infrastructure is robust, handling real-world scenarios like network latency and API limits. **Next Steps:** Phase 5 will introduce Multimodal features (Voice Input and Image Analysis) to complete the functional deliverables.