iFarmAssist - Final Year Project Implementation Plan

1. Project Understanding Summary

iFarmAssist is an AI-powered integrated farm assistant designed to solves the problem of delayed and inaccessible agricultural advice for farmers in Kerala. The system serves **farmers** (who need instant, personalized guidance in Malayalam/English via text, voice, or image) and **agricultural officers** (who handle complex escalated cases). This system is needed to bridge the gap caused by language barriers and service delays, enhancing productivity by providing instant, context-aware advice using LLM + RAG technology, while continuously learning from expert feedback to reduce future workload.

2. Objective-to-Implementation Mapping Table

Objective from PDF	**Module/Feature**	**Technology Used**	**Implementation Steps**
Build multimodal input module (Malayalam queries via text, voice, images)	Input Capture Module (Mobile App)	React Native, `react-native-voice`, `react-native-camera`	1. Build UI with large icons. 2. Integrate STT (Whisper/Google). 3. Integrate Camera logic.
Implement context-aware AI engine using LLM + agriculture knowledge base	Core AI Engine (Backend)	Python (FastAPI/Flask), LangChain, Google Gemini API	1. Setup RAG pipeline. 2. Create Context Aggregator. 3. Integrate LLM.
Create real-time advisory system (Instant, reliable, simple Malayalam guidance)	Response Delivery	React Native, WebSocket/REST	1. Optimize API latency. 2. Format response card in UI.
Establish a learning loop (Farmer feedback + Expert validation)	Continuous Improvement Module	PostgreSQL, ChromaDB, Vector Embeddings	1. 'Helpful' buttons in App. 2. Pipeline to vectorise expert answers.
Develop a mobile application delivering advisory system	Mobile Application	**React Native** (Strictly NO Flutter)	1. Setup Expo/CLI. 2. Implement Screens. 3. Connect to Backend.

Objective from PDF	Module/Feature	Technology Used	Implementation Steps
Escalate complex issues to human experts	Escalation / Expert Dashboard	React (Web), PostgreSQL	1. Logic for Confidence Score check. 2. Web Dashboard queue for experts.

3. System Architecture Breakdown

Based on **Final Architecture Diagram** and **Flow Diagram**:

1. React Native Mobile App (Farmer Interface)

- **Purpose**: Primary interface for Farmers to submit queries and receive advice.
- **Inputs**: Voice (Mic), Image (Camera/Gallery), Text (Keyboard), Location (GPS).
- **Outputs**: Text Advice (Malayalam), Voice Output (TTS), Status Updates.
- **Technologies**: React Native, Axios, Expo AV (Audio).
- **Security**: JWT Authentication, Secure Storage for tokens.

2. Backend Services (Data Processing & AI)

- **Purpose**: Core logic, orchestrating data enrichment, AI reasoning, and API management.
- **Inputs**: Raw user query (Audio/Image/Text), User Context (Location, History).
- **Outputs**: JSON Response (Answer, Confidence Score).
- **Technologies**: Python (FastAPI/Django), LangChain, Pillow (Image).
- **Security**: Input validation, Rate limiting, API Key management.

3. APIs (Internal & External)

- **Internal**: `POST /query`, `GET /history`, `POST /feedback`, `GET /escalations`.
- **External**:
  - **STT**: Google Speech-to-Text / OpenAI Whisper.
  - **Weather**: OpenWeatherMap / IMD API.
  - **LLM**: Google Gemini API.
- **Inputs/Outputs**: JSON payloads.

4. Database

- **PostgreSQL**: Relational data (Users, Officers, Query Metadata, Escalation Cases).
- **Vector Database (ChromaDB/Pinecone)**: Embeddings of agricultural knowledge base & expert answers.
- **Purpose**: Persistent storage and RAG retrieval source.

5. ML/AI Components

- **RAG Engine**: Retrieves relevant docs based on query embedding.

- **Confidence Scorer**: Evaluates LLM answer quality.
- **Computer Vision**: Pre-trained model for plant disease identification (served via API).

6. Authentication

- **Service**: Custom Auth or Firebase Auth.
- **Role Management**: Farmers (Mobile), Officers (Web Dashboard).

7. Continuous Improvement (Learning Loop)

- **Purpose**: Retrain system with "Gold Standard" data from experts.
- **Mechanism**: Feedback -> Database -> Re-embedding -> Vector DB Update.

4. Module-Wise Implementation Plan

Module 1: Input Processing & Client (Mobile)

- **Objective**: Capture multimodal input.
- **Requirements**: Support Malayalam Audio, Images, Text.
- **Steps**:
  1. **Environment**: Install Node.js, React Native Layout.
  2. **Components**: `VoiceRecorder`, `ImagePicker`, `ChatInterface`.
  3. **API**: Send `FormData` (multipart) to Backend.
  4. **Validation**: Limit audio duration (30s), Image size (5MB).

Module 2: context Aggregator (Backend)

- **Objective**: Enrich query with User + Weather context.
- **Steps**:
  1. **Folder Structure**: `app/services/context.py`.
  2. **Logic**: Fetch User Profile (DB) + Weather (External API).
  3. **Prompt Engineering**: Construct "Enriched Prompt" string.

Module 3: Core AI Engine

- **Objective**: RAG + Decision.
- **Steps**:
  1. **Ingestion Script**: Load PDF/Text docs -> Chunk -> Embed -> ChromaDB.
  2. **Retrieval**: Query Vector DB with prompt embedding.
  3. **Generation**: Call Google Gemini with Context + Prompt.
  4. **Scoring**: Implement heuristic or LLM-based confidence check (0-100%).

Module 4: Escalation & Expert Dashboard

- **Objective**: Handle low confidence queries.
- **Steps**:
  1. **Web App**: React.js Admin Panel.
  2. **Backend Logic**: If `score < Threshold`, save to `EscalationQueue` table.
  3. **Feature**: Expert views `EscalationQueue`, types answer, submits.
  4. **Notification**: Push notification to Farmer.

5. React Native Application Implementation

- **Project Setup**: `npx react-native init iFarmAssist` (or Expo).
- **Screen Flow**:
  - **Splash Screen**: Branding.
  - **Login/Signup**: Phone number/OTP preferred for farmers.
  - **Home Screen**: Big Buttons [Voice] [Camera] [Text].
  - **Chat/Result Screen**: Thread view of query & response.
  - **Profile**: Manage crops/location.
- **Navigation**: `react-navigation` (Stack Navigator).
- **State Management**: `Zustand` or `Context API` (Simple & effective).
- **API Calls**: Centralized `apiClient.js` with Interceptors.
- **UI-UX**: High contrast, Localized text (Malayalam labels).

6. Backend Implementation Plan

- **Framework**: **FastAPI** (Python) - High performance, easy async for AI.
- **API Design**: RESTful.
  - `POST /api/v1/auth/login`
  - `POST /api/v1/query/submit` (Multipart)
  - `GET /api/v1/expert/pending`
- **Auth**: OAuth2 with Password Bearer (JWT).
- **Roles**: `role` field in User table ('FARMER', 'EXPERT').
- **Logging**: `logger` middleware to track every query for analytics.

7. Database Design & Implementation

PostgreSQL Schema

- **Users**: `id, name, phone, role, location, crops_grown`
- **Queries**: `id, user_id, input_type, input_content_url, enriched_prompt, response_text, confidence_score, status (SOLVED/SCALATED), created_at`
- **Escalations**: `id, query_id, expert_id, expert_response, resolved_at`
- **Feedback**: `id, query_id, rating (HELPFUL/NOT), comments`

Vector DB (ChromaDB)

- **Collection**: `knowledge_base`
  - Metadata: `source_doc`, `date`, `category`
- **Collection**: `expert_knowledge` (For learning loop)

8. Security Implementation

- **Auth Flow**: JWT Tokens (Access/Refresh).
- **Data Encryption**: HTTPS (TLS) for all API calls. BCRYPT for passwords (if used).
- **Secure API Access**: API Keys for External Services stored in `.env`.
- **Vulnerability Prevention**: SQL Injection (use ORM like SQLAlchemy), XSS (Sanitize inputs).

9. Testing Plan

- **Unit Testing**: `pytest` for Backend logic (Prompt generation, Score calc). `Jest` for React Native components.
- **Integration Testing**: Test API endpoints with DB mock. Test RAG retrieval accuracy.
- **System Testing**: Full flow: App -> Backend -> AI -> App.
- **UAT**: Give app to 5 real users (farmers/students), check if Malayalam advice is accurate.
- **Tools**: Postman (API), Jest, PyTest.

10. Deployment & Hosting Plan

- **Mobile**: Build `.apk` using `eas build` or `gradlew assembleRelease`. Distribute via direct APK sharing for project demo.
- **Backend**: Render / Railway (Free tier for students) or AWS EC2 (Free tier).
- **Database**: Supabase (PostgreSQL), ChromaDB (Self-hosted on backend container).
- **CI/CD**: GitHub Actions to run tests on push.

11. Project Timeline (Final Year Ready)

- **Week 1**: **Setup & Design**. Repo init, DB Schema Design, UI Wireframes in Figma.
- **Week 2**: **Backend Core**. FastAPI setup, Auth, Database Models.
- **Week 3**: **AI Engine Basic**. RAG Setup, Pinecone/Chroma init, Ingestion script.
- **Week 4**: **Mobile App Core**. React Native init, UI Layouts (Home, Chat).
- **Week 5**: **Integration**. Connect App to Backend (Text Query).
- **Week 6**: **Multimodal**. Add Voice (STT) and Image handling.
- **Week 7**: **Escalation Module**. Expert Dashboard (Web), Logic setup.
- **Week 8**: **Learning Loop**. Feedback API, Re-training pipeline.
- **Week 9**: **Testing & Validating**. Unit tests, UAT, Bug fixing.
- **Week 10**: **Final Polish**. Deployment, Documentation, Viva Prep.

12. Final Deliverables

- **Internal Review**: System Architecture Document, Database Schema, Figma Prototypes.
- **Final Evaluation**: Working React Native APK, Live Web Dashboard, Source Code (GitHub), Project Report (PDF).
- **Viva/Presentation**: PPT Slides, Live Demo of "Voice Query -> Malayalam Response", Code Walkthrough.