Overall Backend Architecture Plan

The backend for the "Digital Krishi Officer" app will be a hybrid system combining **n8n** (for no-code workflow automation and agentic tasks) and **Python FastAPI** (for core API handling, database interactions, and custom logic). This fusion leverages n8n's strengths in orchestrating integrations, scheduling, and event-driven workflows (e.g., pulling data from external APIs like weather or government services) without writing code, while FastAPI handles high-performance, scalable API endpoints, authentication, real-time features, and complex computations like AI inference or image processing.

Key Principles:

- **Separation of Concerns**: FastAPI manages synchronous, user-facing requests (e.g., chat

queries, data storage). n8n handles asynchronous, agentic workflows (e.g., background monitoring, notifications) that can be triggered via webhooks or schedules.

- **Integration Between Components**:
  - FastAPI exposes APIs that can trigger n8n workflows (e.g., via HTTP requests to n8n's webhook nodes).
  - n8n can call FastAPI endpoints (e.g., to store data or send push notifications through FastAPI's notification system).
  - Shared Database: Both can connect to the same PostgreSQL + Vector DB (e.g., Qdrant/Weaviate) for data consistency. Use environment variables or API keys for secure access.
  - Deployment: Host FastAPI on AWS/Azure with Gunicorn/UVicorn for scalability. Run n8n self-hosted on the same cloud with Docker for workflows.
- **Scalability & Security**: Use JWT authentication in FastAPI. n8n workflows should be secured with API keys. Edge computing (mentioned in PDF) can be handled via AWS Lambda or Azure Functions triggered by n8n for heavy tasks like AR analysis.
- **Tech Stack Alignment**: Align with PDF (FastAPI backend, PostgreSQL/Vector DB, GPT4o-mini, Whisper ASR, CV models). Add libraries like SQLAlchemy/ORM for DB, PyTorch/OpenCV for vision, and WebSockets (via FastAPI) for real-time chat.

Core Features Mapping

Based on the provided core features and PDF details (e.g., multilingual AI advice, AR analysis, integrations with govt/weather/retailers, community features), here's how features are divided:

Feature	Description (from PDF/User)	Handled By	Rationale

Chat with AI	Instant advice via voice/text/image in Malayalam, context-aware (location, crop, season, history).	FastAPI (core) + n8n (agentic enhancements)	FastAPI for real-time query handling; n8n for multi-step agentic flows if complex (e.g., chaining AI calls).
Scheduler	Alerts for farming tasks (e.g., planting reminders).	n8n	No-code scheduling workflows.
Tips on Crop	Personalized crop tips based on history/context.	FastAPI	API endpoint to query DB/AI.
Solution Trust %	Confidence scoring for AI recommendations.	FastAPI	Custom logic in AI response processing.
Decentralized Group Chat	Peer-to-peer farmer community integration.	FastAPI	Real-time WebSockets; "decentralized" interpreted as community forums (not blockchain-based unless specified).
Q/A Repository	Store and retrieve community Q&As.	FastAPI	DB-backed API for CRUD operations.
Disaster Alert	Real-time warnings (e.g., from weather data).	n8n	Agentic workflow to poll APIs and notify.
Govt Updates	Integration with schemes, subsidies, updates.	n8n	Scheduled polling of govt APIs.
Live Crop Details (AR Analysis)	Real-time AR field analysis (soil, health, pests) via image/scan.	FastAPI (core) + n8n (if multi-step)	FastAPI for image upload/processing; n8n for integrating external CV services if agentic.
Nearest Authorized Retailers	Mapping and recommendations based on location.	FastAPI	API with geolocation queries (e.g., using Google Maps API or DB).

Parts Handled by Python FastAPI

FastAPI will serve as the primary backend server, exposing RESTful APIs and WebSockets for the mobile app (React Native/Flutter). It handles user sessions, data persistence, and compute-intensive tasks. Use Pydantic for models, SQLAlchemy for PostgreSQL, and Weaviate/Qdrant client for vector search (e.g., for semantic crop queries).

Key Components in FastAPI:

1. **Authentication & User Management**:
    - Endpoints: `/auth/register`, `/auth/login` (JWT-based).
    - Store farmer profiles (location, crop history) in PostgreSQL.
    - Integration: Use farmer history for context-aware AI.
2. **AI Chat Endpoint**:
    - Endpoint: `/chat` (POST for text/voice/image queries).
    - Logic:
        - Process inputs (e.g., Whisper for ASR on voice, multimodal LLM like GPT4o-mini for text/image).
        - Fetch context from DB (location via GPS header, history via user ID).
        - Call OpenAI API or local model for response.
        - Compute trust % (e.g., based on model confidence or custom scoring: if similarity search in vector DB > 0.8, trust = 90%).
        - Support Malayalam via IndicBERT or fine-tuned models.
    - Outputs: JSON with advice, trust %, tips.

3. **Image/AR Analysis Endpoint**:

- Endpoint: `/analyze/image` (POST with image upload).
- Logic:
  - Use CV libraries (e.g., OpenCV/PyTorch) for pest/disease detection (integrate Plantix-like models).
  - Real-time: Process on edge (e.g., via AWS SageMaker endpoint) if heavy.
  - Overlay details (soil type, health) – return JSON for client-side AR rendering (ARCore/ARKit).
  - Store analysis in DB for history.

4. **Crop Tips & Q/A Repository**:
   - Endpoints: `/tips/{crop_type}` (GET personalized tips), `/qa` (POST to add, GET to search).
   - Logic: Vector DB for semantic search on Q&As/tips. Use embeddings from GPT4o-mini.
5. **Decentralized Group Chat**:
   - Use FastAPI WebSockets: `/ws/chat/{group_id}`.
   - Logic: Room-based chat (e.g., using Redis for pub/sub). Store messages in PostgreSQL for persistence.
   - "Decentralized" via peer-moderated groups, not full P2P (unless integrating WebRTC, but keep simple).
6. **Nearest Retailers**:
   - Endpoint: `/retailers/nearby` (GET with lat/long).
   - Logic: Query authorized retailer DB (populated via admin) or integrate Google Places API. Return mapped list.
7. **Other APIs**:

- `/user/history` (GET/POST farmer data).
- Notification hub: `/notifications` (for pushing alerts from n8n).

Implementation Notes:

- Dependencies: `fastapi`, `sqlalchemy`, `weaviate-client`, `openai`, `torch` (for CV), `pydub` (for audio).
- Rate Limiting: Use SlowAPI to prevent abuse.
- Error Handling: Standard HTTP errors with Malayalam support in responses.

Agentic Tasks in n8n (No-Code Workflows)

n8n excels at no-code agents: Use its AI nodes (e.g., OpenAI integration), schedulers, webhooks, and API nodes for autonomous, event-driven tasks. Each "agent" is a workflow that runs independently, triggered by time, webhooks, or events.

Key Workflows (Agents) in n8n:

1. **Scheduler Agent**:
   - Trigger: Cron node (e.g., daily at 6 AM).
   - Flow:
     - Fetch user data from FastAPI API (HTTP node).
     - Use AI node (GPT4o-mini) to generate personalized schedules/tips based on season/crop.
     - Send push notifications via FastAPI webhook or Firebase integration.
   - Outputs: Alerts like "Time to fertilize based on your rice crop history."

2. **Disaster Alert Agent**:

- Trigger: Schedule (every hour) or webhook (from weather service).
- Flow:
  - Poll IMD Weather API (HTTP node) for disasters (e.g., floods in Kerala).
  - Filter for user locations (from DB via FastAPI call).
  - AI node to contextualize (e.g., "Heavy rain alert: Protect crops with X method").
  - Push to users via email/SMS (Twilio node) or FastAPI notification endpoint.
- Integration: Govt APIs (Kerala Ag Dept, IMD).

3. **Govt Updates Agent**:
   - Trigger: Schedule (daily).
   - Flow:
     - Poll govt APIs/databases (HTTP node for Kerala AIMS, subsidy schemes).
     - AI node to summarize updates in Malayalam.
     - Store in vector DB via FastAPI call.
     - Notify relevant users (e.g., subsidy eligibility based on crop type).
4. **Community Integration Agent (for Peer-to-Peer Learning)**:
   - Trigger: Webhook (from FastAPI when new Q/A posted).
   - Flow:
     - Analyze new Q/A with AI node for categorization.
     - Match with similar farmer profiles (vector search via API call).
     - Facilitate "decentralized" sharing: Email/SMS peers or post to group chat via FastAPI webhook.

5. **AR/Crop Analysis Enhancement Agent** (if multi-step):

- Trigger: Webhook from FastAPI (post-image upload).
- Flow:
  - Chain CV services (e.g., HTTP to Plantix API).
  - AI node for multimodal analysis (e.g., combine image with weather data).
  - Return enriched data to FastAPI.

Implementation Notes:

- Nodes: Use n8n's built-in HTTP, Cron, OpenAI, Database (for direct PostgreSQL access if needed), and Notification nodes.
- Agentic Nature: Workflows act as "agents" by looping/branching based on conditions (e.g., if disaster threshold met, notify).
- Monitoring: n8n dashboard for logs; scale with workers.
- Security: API keys for external integrations; restrict webhook access.

Integration & Deployment Flow

1. **User Flow Example**: Farmer sends chat query → FastAPI processes AI → If scheduler needed, triggers n8n webhook → n8n runs agent and pushes back.
2. **Data Flow**: Shared DB ensures sync (e.g., n8n writes alerts to a queue table, FastAPI reads).
3. **Testing**: Start with local n8n + FastAPI dev server. Use Postman for APIs, n8n editor for workflows.
4. **Potential Extensions**: If AR needs heavy edge computing, add AWS Lambda triggers in n8n.

This plan ensures a robust, maintainable backend. If you provide more details (e.g., specific APIs), I can refine workflows or generate sample code.