1. Product Overview (SaaS Edition)

- Product Name: KNOWEB EstateIQ
- Target Market: Tea, Rubber, and Cinnamon Plantations (Scalable from small estates to large regional plantation companies).
- Deployment Model: Cloud-based SaaS (Software as a Service).
- Core Value Proposition: A unified platform to digitize muster, harvest, and inventory operations, capable of handling multiple distinct estates under a single subscription.

2. Enhanced User Hierarchy (Multi-Tenant Structure)

To sell this, we need a hierarchy that separates *your* management from the *client's* management.

Level 1: System Owner (You/Vendor)

- **Super Admin (SaaS Admin):**
    - Client Management: Create and suspend client accounts (e.g., Estate A, Estate B).
    - Subscription Management: Set limits on the number of users or divisions per client.
    - Global Configuration: Push updates to all clients simultaneously.

Level 2: Client Admin (The Estate Owner/Manager)

- **Estate Admin:**
    - White-Labeling: Upload their own estate logo and name (replacing "Dunwatta Estate" dynamically).
    - Master Setup: Define their specific Divisions (e.g., "Upper Division", "Lower Division" instead of just GC, FD).
    - User Provisioning: Create accounts for their own Managers, Field Officers, and Storekeepers.

Level 3: Operational Users (Standard Roles)

- **Manager**: Approves musters, stocks, and views financial reports *only for their assigned estate*.
- **Field Officer**: Enters daily data for their specific division.

- **Storekeeper**: Manages inventory for the specific estate's warehouse.

3. Key Feature Adjustments for Sales

A. Configurable "Master Data" (Crucial for selling)

Instead of hardcoding "Tea, Rubber, Cinnamon," allow the client to toggle crops on/off.

- Crop Selector: "Does this estate grow Cinnamon? Yes/No." (Hides Cinnamon modules if No).
- Division Builder: Allow clients to add/rename divisions dynamically (e.g., Client A has "North Division," Client B has "Valley Section").
- Work Item Library: Pre-load standard tasks (Plucking, Tapping) but allow clients to add custom tasks (e.g., "Vanilla Harvesting" or "Coconut Picking").

B. Dynamic Dashboard

- Widget-Based View: Allow Managers to choose what they see. A tea estate manager needs "Green Leaf Weight" prominent, while a rubber estate manager needs "Latex Liters."
- Rainfall/Weather: Integrate a public weather API so the "Rainfall" widget auto-updates based on the estate's GPS location.

C. Billing & Audit Ready

- Activity Logs: A "System Audit Trail" is a strong selling point for large companies to prevent fraud.
- Exportable Reports: Ensure all reports (Paysheets, Crop Books) can be exported to Excel/PDF with the *client's* logo, not yours.

4. Revised Module List (Market-Ready)

Module	Sales Pitch Feature
1. Digital Muster	"Eliminate ghost workers." GPS-tagged attendance to prove where and when the Field Officer took attendance.
2. Smart Harvest	"Real-time Yield Tracking." Compare today's harvest against the 10-year historical average for that specific field.

Module	Sales Pitch Feature
3. Inventory Control	"Stop Pilferage." Alerts when fertilizer stock drops below a 'Buffer Level' defined by the manager.
4. Checkroll Link	"Payroll Ready." Auto-calculate "Work Offered Days" and "Over Kilos" to export directly to their payroll software.
5. AI Insights (Premium)	"Predictive Staffing." Use historical data to predict how many pluckers are needed next week based on weather forecasts.

5. Technical Strategy for Selling

- Multi-Tenancy: The database must have a Tenant_ID column in every table. This ensures Estate A never sees Estate B's data, which is the #1 security concern for buyers.
- Mobile-First Design: Field Officers often use cheap smartphones. The app must be lightweight and work with spotty 3G signals (Offline Mode recommended).
- Onboarding Wizard: When a new client signs up, a "Setup Wizard" should guide them: *1. Name your Estate -> 2. Upload Logo -> 3. Add Divisions -> 4. Select Crops.*

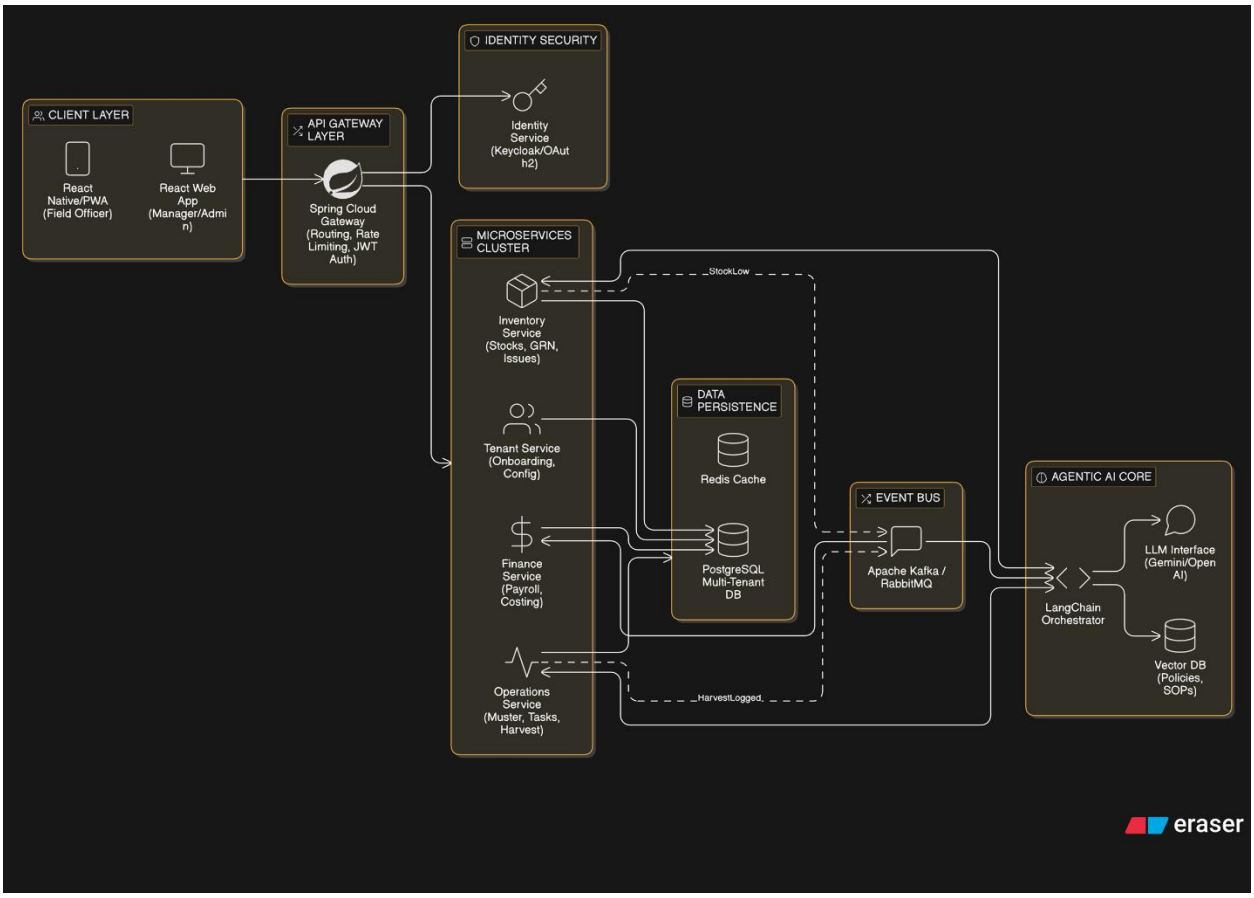
6. Implementation Plan

1. Refactor Database: Add EstateID to all tables (Workers, Crops, Users).
2. Build "Super Admin" Panel: An interface for you to create new tenant instances.
3. Parameterize Hardcoded Values: Convert "GC Division", "FD Division" into dynamic variables fetched from the database.
4. Develop Landing Page: A marketing site (like www.yourproduct.com) where interested estates can request a demo.

1. High-Level System Architecture

We will use a Microservices Architecture with a Shared-Database, Schema-per-Tenant approach (or Discriminator Column) to keep infrastructure costs low while ensuring logical data separation.

Architecture Diagram



2. Technology Stack & Design Patterns

A. Frontend (React)

- Framework: React 18+ (Functional Components) with TypeScript.
- State Management: Zustand or Redux Toolkit (for complex global state like User Permissions and Tenant Config).
- Design Patterns:

    - Atomic Design: Structure components into Atoms (Buttons), Molecules (SearchBox), Organisms (Header), Templates.
    - Container/Presentational Pattern: Separate logic (API calls) from UI rendering.
    - Feature-Sliced Design (FSD): Organize code by business domain (e.g., features/muster, features/harvest) rather than technical type.
    - Custom Hooks: For reusable logic (e.g., useTenantConfig(), useOfflineSync()).
- UI Library: Material UI (MUI) or Ant Design (Enterprise look & feel).

B. Backend (Spring Boot Microservices)

- Framework: Spring Boot 3.x.
- Communication: REST for synchronous, Kafka for asynchronous (Event-Driven).
- Design Patterns:
    - API Gateway Pattern: Single entry point for all clients.
    - Database-per-Service (Logical): Each microservice owns its schema/tables within the Postgres instance.
    - Circuit Breaker (Resilience4j): To handle failures gracefully (e.g., if the AI service is down, manual entry still works).
    - Saga Pattern (Orchestration): For distributed transactions. *Example:* "Approve Stock Request" -> (1) Inventory Service reserves stock -> (2) Finance Service checks budget -> (3) Notification Service alerts Field Officer.
    - Tenant Context Pattern: Use a RequestInterceptor to extract Tenant-ID from the JWT header and inject it into the Hibernate Filter automatically (Row-Level Security).

3. The Agentic AI Component

Given your interest in Agentic AI (and your "Project Genesis" background), this system shouldn't just *display* data; it should *act* on it.

Proposed Agent: "The Plantation Supervisor Agent"

- Role: Semi-autonomous Operations Manager.
- Tools:
    - WeatherAPI: To check rainfall forecasts.

    - DB_Read_Access: To check historical yields and current roster.
    - Roster_Tool: To assign workers to tasks.
- Agentic Workflow (Example):
    1. Trigger: It's 4:00 PM. Agent wakes up to plan tomorrow's work.
    2. Observation: Checks Weather API. Forecast: "Heavy Rain tomorrow morning."
    3. Reasoning: "Rain prevents Tapping (Rubber). Plucking (Tea) is inefficient but possible. Fertilizer application is wasteful (washout risk)."
    4. Action:
        - *Auto-Correction:* Cancels "Fertilizer Application" for Field No. 4.
        - *Re-Assignment:* Moves Rubber Tappers to "Drain Cleaning" or "Indoor Factory Work".
        - *Notification:* Sends a WhatsApp/SMS draft to the Human Manager: *"I've drafted a revised roster due to rain forecast. 12 workers moved from Tapping to Drains. Approve?"*

4. Database Design (PostgreSQL)

Since this is a SaaS product, every table MUST have a tenant_id (UUID) which serves as the partition key.

Entity Relationship Diagram (ERD)

```
TENANTS ||--o{ USERS : "has"
TENANTS ||--o{ DIVISIONS : "owns"
DIVISIONS ||--o{ FIELDS : "contains"
FIELDS ||--o{ HARVEST_LOGS : "produces"
USERS ||--o{ MUSTER_LOGS : "marks"
USERS ||--o{ STOCK_REQUESTS : "requests"

TENANTS {
  uuid tenant_id PK
  string company_name
  jsonb subscription_plan
  boolean is_active
}

USERS {
  uuid user_id PK
  uuid tenant_id FK
  string role "Manager, FieldOfficer"
  string username
  string password_hash
}

FIELDS {
  uuid field_id PK
  uuid tenant_id FK
  string field_name
  string crop_type "Tea, Rubber"
  float acreage
}

HARVEST_LOGS {
  uuid log_id PK
  uuid tenant_id FK
  uuid field_id FK
  date date
  float quantity_kg
  uuid created_by FK
}
```

SQL Schema File (plantation_db_schema.sql)

You can save the following code block as a .sql file. It includes the partitioning logic for multi-tenancy.