# Comprehensive Documentation Plan for CRAD

## 1. 1. Overview

The CRAD Microservices Architecture represents a complex system designed to provide agricultural recommendations and insights. Proper documentation is not just beneficial but essential for the success of this project. This section outlines the critical aspects of our documentation strategy.

## Importance of Comprehensive Documentation

Comprehensive documentation for the CRAD microservices architecture serves multiple critical functions:

* **Knowledge Transfer:** Enables new team members to quickly understand the system architecture, reducing onboarding time from weeks to days
* **System Consistency:** Provides a single source of truth for architectural decisions, preventing drift between services and maintaining cohesion
* **Maintenance Support:** Facilitates troubleshooting and debugging by clearly mapping dependencies and interactions between services
* **Scalability Planning:** Documents current capacity limits and scaling strategies for each microservice, supporting future growth
* **Security Compliance:** Tracks authentication flows, data access patterns, and security measures implemented across the system
* **Technical Debt Management:** Identifies temporary solutions and planned improvements, helping prioritize refactoring efforts

Without thorough documentation, the distributed nature of microservices can lead to siloed knowledge, inconsistent implementations, and difficulties in system-wide updates.

## Stakeholder Engagement Strategy

Our documentation must serve diverse stakeholders with varying technical expertise and needs:

**Farmers (End Users):**

* Simple visual guides explaining how to interpret recommendations.
* Non-technical explanations of data requirements for accurate results.
* FAQ documents addressing common concerns about data privacy and system reliability.

**Agronomists (Domain Experts):**

* Detailed explanations of agricultural algorithms and recommendation logic.
* Documentation on how to customize system parameters for different regions.
* Guides for providing feedback on recommendation accuracy.

**System Administrators:**

* Complete deployment guides with environment variable references.
* Monitoring and alerting setup documentation.
* Backup and recovery procedures for all data stores.

**Developers:**

* API contracts with example requests/responses and error handling.
* Service interaction diagrams showing message flows and dependencies.
* Local development setup guides with mock data generation.
* Testing strategies for each service and integration points.

**Product Managers:**

* Feature roadmaps aligned with microservice development milestones.
* Performance metrics and SLAs for each service.
* User feedback collection points throughout the system.

**Our engagement approach includes:**

* **Regular Documentation Reviews:** Scheduled sessions with representatives from each stakeholder group to validate documentation accuracy and usefulness
* **Feedback Channels:** Dedicated channels for reporting documentation gaps or unclear sections
* **Living Documentation:** Integration with CI/CD pipelines to ensure documentation stays current with code changes
* **Multi-format Delivery:** Providing documentation in various formats (interactive web pages, PDFs, videos) to suit different learning preferences

## 2. User Personas

User personas are essential for designing an effective CRAD system that meets the needs of diverse stakeholders. These personas help development teams understand user motivations, challenges, and goals throughout the product lifecycle.

## Development Process for Creating Personas

Our approach to developing comprehensive user personas followed these structured steps:

## 1. Data Collection

* Conducted 27 field interviews with actual farmers in Rwanda across various regions and farm sizes.
* Shadowed 5 agronomists during their regular consultation activities.
* Held workshops with 3 agricultural ministries to understand administrative needs.
* Analyzed usage patterns from similar agricultural recommendation systems.

## 2. Pattern Identification

* Used affinity mapping to group common behaviors, challenges, and goals.
* Identified recurring technical comfort levels and device access patterns.
* Mapped communication preferences and information consumption habits.

## 3. Persona Creation

* Developed preliminary personas based on identified patterns.
* Created detailed narrative profiles with demographic information.
* Assigned realistic goals, motivations, frustrations, and technical abilities.
* Developed visual representations for each persona.

## 4. Validation and Refinement

* Reviewed personas with actual users from each category.
* Conducted validation interviews to verify accuracy.
* Refined personas based on feedback until they accurately represented user groups.

## 5. Persona Implementation

* Integrated personas into development workflows.
* Referenced personas during sprint planning and feature prioritization.
* Used personas during usability testing to verify design decisions.

## Key Characteristics for Each Role

Our research identified four primary user types, each with distinct needs and usage patterns:

## 1. Super Admin Persona: “Claudine, Agricultural Data Coordinator”

**Demographics:**

* 35-45 years old with advanced degree in agricultural sciences.
* Works for the Ministry of Agriculture or major agricultural NGO.
* High technical literacy and data management expertise.

**Goals and Motivations:**

* Ensure system contains accurate, up-to-date agricultural information.
* Maintain data quality across all information sources.
* Generate insights about system usage to improve recommendation quality.
* Support policy decisions with reliable agricultural data.

**Key Needs:**

* Comprehensive dashboard showing system health and usage statistics.
* Powerful document ingestion tools with quality verification.
* User management capabilities to onboard and support agronomists.
* Audit trails for all system modifications and recommendations.

**Pain Points:**

* Managing large volumes of agricultural documents in multiple languages.
* Ensuring consistent data quality across diverse information sources.
* Coordinating with technical and non-technical stakeholders.

**Usage Patterns:**

* Uses system 3-5 times weekly for 1-2hour administrative sessions.
* Primarily desktop access from office environment.
* Performs bulk operations and system configuration.

## 2. Agronomist Persona: “Emmanuel, District Agricultural Advisor”

**Demographics:**

* 30-50 years old with formal agricultural education.
* Serves 200-300 farmers across multiple villages.
* Moderate to high technical literacy.
* Expertise in regional crops and farming practices.

**Goals and Motivations:**

* Provide accurate and timely agricultural advice to many farmers.
* Increase crop yields and sustainability in their region.
* Build trust with farmers through reliable recommendations.
* Document interventions and track outcomes.

**Key Needs:**

* Mobile-friendly interface for field consultations.
* Ability to manage multiple farmer profiles efficiently.
* Quick access to soil analysis interpretation and recommendation history.
* Tools to customize recommendations based on local knowledge.
* Offline capabilities for areas with limited connectivity.

**Pain Points:**

* Balancing quality advice with the need to serve many farmers.
* Working with varying levels of data completeness among farmers.
* Adapting generalized recommendations to local conditions.
* Convincing traditional farmers to adopt new practices.

**Usage Patterns:**

* Daily use, often in brief 5-15minute sessions while in the field.
* Both mobile and desktop access depending on location.
* Heavy use during planting seasons and after weather events.

## 3. Farmer Persona: “Mutoni, Smallholder Vegetable Farmer”

**Demographics:**

* 25-65 years old with varying education levels.
* Manages 0.5-2 hectares of land.
* Limited to moderate technical literacy.
* Primarily speaks local language with some official language proficiency.

**Goals and Motivations:**

* Increase crop yields and income from limited land.
* Reduce crop losses from pests, diseases, and weather.
* Access better market prices for produce.
* Ensure sustainable farming for future generations.

**Key Needs:**

* Simple, visual interfaces with minimal text.
* Clear, actionable recommendations with timing information.
* Local language support with voice interface options.
* Low-bandwidth operation on basic smart.
* Contextual help and educational content.

**Pain Points:**

* Limited access to agricultural expertise and timely information.
* Difficulty interpreting complex agricultural recommendations.
* Concerns about recommendation costs versus expected benefits.
* Limited time to learn new technologies during busy farming periods.

**Usage Patterns:**

* Irregular usage tied to specific decision points (planting, fertilization, pest management).
* Primarily mobile access, often shared devices.
* Preference for voice and image-based interactions.

## 4. Viewer Persona: “Jean-Pierre, Agricultural Extension Officer”

**Demographics:**

* 30-45 years old with intermediate agricultural training.
* Works for government extension services or NGO.
* Moderate technical literacy.
* Responsible for monitoring agricultural interventions.

**Goals and Motivations:**

* Track agricultural interventions across multiple regions.
* Generate reports on adoption rates and outcomes.
* Identify successful practices for broader promotion.
* Provide evidence for policy and funding decisions.

**Key Needs:**

• Read-only access to anonymized farmer data and recommendations

• Aggregated views and statistical reports

• Export capabilities for reporting to stakeholders

• Filtering tools to focus on specific regions or crop types

**Pain Points:**

• Getting reliable data from remote or underserved areas

• Reconciling system recommendations with existing programs

• Limited visibility into farmer adoption and compliance

**Usage Patterns:**

• Getting reliable data from remote or underserved areas

• Reconciling system recommendations with existing programs

• Limited visibility into farmer adoption and compliance

## Persona Application in System Design

These personas directly influence our microservices architecture in several ways:

**Frontend Considerations:**

• Multiple UI versions optimized for different personas (simplified for farmers, comprehensive for admins)

• Responsive design prioritizing mobile-first for farmers and agronomists

• Offline capabilities for areas with intermittent connectivity

**Backend Architecture Impacts:**

• Authentication service with role-based access control aligned to persona types

• Performance optimization for farmer-facing services (low latency, minimal data transfer)

• Comprehensive data access for admin tooling with appropriate security controls

**Integration Points:**

• SMS notification service for farmers with limited internet access

• Data export capabilities for viewers and administrators

• API design supporting various client applications and usage patterns

These personas will be regularly revisited and updated throughout the development process to ensure the system continues to meet the evolving needs of all stakeholders.

## 3. 3. Feature Requirements

Below are the detailed feature requirements organized by priority and microservice. Each requirement includes corresponding user stories to ensure alignment with stakeholder needs.

## Core Service Features by Priority

## Farmer Profile Service (P0 - Highest Priority)

**User Registration and Authentication**

• As a farmer, I want to create an account with minimal information so I can quickly access the system

• As an agronomist, I want to register with my credentials and organization details so I can manage my farmer clients

• As an administrator, I want to manage user roles and permissions so I can control system access appropriately

**Farm Profile Management**

• As a farmer, I want to register my farm location, size, and current crops so I can receive relevant recommendations

• As a farmer, I want to update my farm details when changes occur so my profile remains accurate

• As an agronomist, I want to create and manage multiple farmer profiles so I can serve my client base efficiently

**Data Storage and Retrieval**

• As a system user, I want my profile data to be securely stored so my information remains private

• As a system user, I want fast access to my profile information so I can use the system efficiently

## Soil & Sensor Ingest Service (P1 - High Priority)

**Soil Analysis Management**

• As a farmer, I want to input soil test results so I can receive tailored recommendations

• As an agronomist, I want to upload soil analysis PDFs so the system can extract and store relevant data

• As a system user, I want to view historical soil test results so I can track changes over time

**Sensor Data Integration**

• As a farmer with IoT sensors, I want my sensor data automatically ingested so I can receive real-time recommendations

• As an agronomist, I want to connect client farm sensors to the system so I can monitor conditions remotely

• As a system administrator, I want to support multiple sensor types and protocols so we can accommodate diverse user equipment

**Environmental Data Collection**

• As a system user, I want access to local weather data so recommendations account for current conditions

• As a system user, I want to receive alerts about extreme weather events so I can take preventive actions

## Recommendation Service (P1 - High Priority)

**LLM-Powered Recommendations**

• As a farmer, I want to receive personalized crop recommendations based on my soil data so I can maximize yields

• As an agronomist, I want the system to generate evidence-based recommendations so I can provide quality advice to farmers

• As a system user, I want recommendations that consider local conditions and constraints so they’re practically applicable

**Fertilizer and Amendment Calculations**

• As a farmer, I want precise fertilizer recommendations with application rates so I can apply the right amount

• As an agronomist, I want to review and adjust system-generated recommendations before sharing them with farmers

• As a system user, I want recommendations that consider cost-effectiveness so I can maximize ROI

**Intervention Tracking**

• As a farmer, I want to record which recommendations I implemented so I can track their effectiveness

• As an agronomist, I want to document interventions across multiple farms so I can analyze patterns

• As a viewer, I want to see aggregated intervention data so I can report on program effectiveness

## Knowledge Service (P2 - Medium Priority)

**Document Management**

• As a system administrator, I want to ingest agricultural reference documents so they can inform recommendations.

• As an agronomist, I want to upload region-specific best practices so recommendations include local knowledge.

• As a viewer, I want to access a searchable repository of agricultural documents so I can find relevant information.

**Vector Database for RAG**

• As a system user, I want AI recommendations to incorporate relevant agricultural knowledge so they’re scientifically sound.

• As a system administrator, I want to maintain and update the knowledge base so it remains current and accurate.

**Educational Content**

• As a farmer, I want access to educational resources that explain recommendations so I understand why they matter.

• As an agronomist, I want to create and share educational content with farmers so they can learn new techniques.

## Cross-Cutting Feature Requirements (P0-P2)

**Offline Capabilities**

• As a farmer with limited connectivity, I want to access core features offline so I can use the system in remote areas

• As an agronomist working in the field, I want data to sync when connectivity is restored so I don’t lose information

**Multilingual Support**

• As a non-English speaking user, I want the interface in my local language so I can understand all features

• As a system administrator, I want to add new languages easily so we can serve diverse regions

**Analytics and Reporting**

• As an agronomist, I want performance dashboards for my farmers so I can track their progress

• As a viewer, I want to generate reports on intervention outcomes so I can evaluate program effectiveness

• As a system administrator, I want usage analytics so I can optimize the system based on user behavior

These feature requirements will guide development priorities across all microservices, ensuring we deliver the most valuable functionality first while maintaining alignment with user needs. Each requirement will be implemented according to its priority level, with P0 features delivered in the initial release.

## 4. 4. Workflow Scenarios

Below are detailed workflow scenarios that illustrate how the system components interact to deliver value to different user types. Each scenario maps the complete journey from initial trigger to final outcome.

## Document Ingestion Workflow

This workflow describes how agricultural documents are processed and made available for the recommendation engine.

• Trigger: Administrator or agronomist uploads a new agricultural document

• Step 1: The Knowledge Service receives the document through the API Gateway

• Step 2: Document metadata is extracted (author, publication date, topic categories)

• Step 3: Text extraction processes the document based on format (PDF, DOCX, etc.)

• Step 4: Content is chunked into appropriate segments for embedding

• Step 5: Vector embeddings are generated for each chunk

• Step 6: Embeddings and metadata are stored in the vector database

• Step 7: The document is indexed and made available for RAG operations

• Step 8: Uploading user receives confirmation of successful processing

## Soil Analysis Workflow

This workflow demonstrates how soil test results are processed and used for recommendations.

• Trigger: Farmer or agronomist uploads soil test results

• Step 1: Soil & Sensor Ingest Service receives the data (manual entry or PDF upload)

• Step 2: For PDF uploads, OCR and structured data extraction occur

• Step 3: Data validation ensures completeness and accuracy

• Step 4: Normalized soil data is stored in the database with farm ID association

• Step 5: The system notifies the Recommendation Service of new soil data

• Step 6: Recommendation Service generates updated recommendations

• Step 7: User is notified that new recommendations are available

## Recommendation Generation Workflow

This workflow details how the system generates personalized recommendations for farmers.

• Trigger: New soil data or scheduled recommendation update

• Step 1: Recommendation Service retrieves farm profile data

• Step 2: Service collects relevant soil test results and sensor data

• Step 3: Environmental data is gathered (local weather, season, etc.)

• Step 4: RAG query is constructed and sent to Knowledge Service

• Step 5: Knowledge Service retrieves relevant information from vector database

• Step 6: LLM orchestrator combines all inputs with a specialized prompt

• Step 7: Domain-specific tools validate feasibility of recommendations

• Step 8: Final recommendations are formatted with application rates

• Step 9: Recommendations are stored and notifications sent to relevant users

## AI Chat Interaction Workflow

This workflow illustrates how users interact with the AI assistant for agricultural advice.

• Trigger: User initiates chat with a question or concern

• Step 1: User query is received through the Web App

• Step 2: API Gateway routes the request to the Recommendation Service

• Step 3: User context is gathered (farm profile, recent soil tests, location)

• Step 4: Chat history is retrieved to maintain conversation context

• Step 5: Knowledge Service is queried with RAG to find relevant information

• Step 6: LLM generates a contextually relevant response

• Step 7: Response is augmented with links to specific recommendations or educational content

• Step 8: Response is delivered to user with any relevant visualizations

• Step 9: Interaction is logged for continuous improvement of the system

## Complete User Journey: From Farm Creation to Recommendations

This scenario traces the complete user journey from initial registration to receiving actionable recommendations.

**Phase 1: Onboarding**

• User registers an account (farmer or agronomist)

• User completes profile information including contact details

• Farmer creates farm profile with location, size, and current crops

• System assigns the user to appropriate regional knowledge base

**Phase 2: Data Collection**

• User uploads initial soil test results or requests soil sampling kit

• User configures any available sensors for integration

• System begins collecting environmental data for the farm location

• User answers additional questions about farming practices and constraints

**Phase 3: Initial Assessment**

• System processes all available data to create baseline assessment

• Preliminary recommendations are generated based on available information

• User receives notification about initial recommendations

• System identifies data gaps and requests additional information if needed

**Phase 4: Recommendation Delivery**

• User accesses detailed recommendations through web or mobile interface

• Recommendations include justification, application guidance, and expected outcomes

• User can ask clarifying questions through the AI chat interface

• System provides educational resources related to recommendations

**Phase 5: Implementation & Feedback**

• User marks recommendations as implemented or rejected

• User provides feedback on ease of implementation

• System schedules follow-up reminders for intervention assessment

• User can report observed results after implementation

**Phase 6: Continuous Improvement**

• System analyzes implementation results across similar farms

• Recommendations are refined based on observed outcomes

• User receives increasingly personalized advice over time

• System suggests soil retesting schedule based on interventions

These workflow scenarios provide a comprehensive view of how users interact with the system and how data flows between microservices to deliver value. Each scenario will be further refined during implementation to ensure optimal user experience and system performance.

## 5. Traceability Matrix

The traceability matrix serves as a crucial documentation tool that connects requirements, design components, code implementations, and test cases across our microservices architecture. This bidirectional traceability ensures that every requirement is implemented and tested properly, while also helping to assess the impact of changes.

## 5.1 Requirements Traceability

Each requirement is assigned a unique identifier with the following format:

• REQ-[Service Prefix]-[Category]-[Number]

• Example: REQ-FPS-DATA-001 (Farmer Profile Service, Data Management, Requirement #1)

• Service Prefixes: FPS (Farmer Profile), SIS (Soil & Sensor), REC (Recommendation), KNW (Knowledge)

The matrix maps each requirement to:

* The microservice(s) responsible for implementation
* Design documents and architectural components
* Code modules and functions that implement the requirement
* Test cases that verify the requirement
* User stories and acceptance criteria from section 4.3

## 5.2 Test Coverage Mapping

Test cases follow a similar identification scheme:

• TEST-[Service Prefix]-[Type]-[Number]

• Example: TEST-REC-UNIT-005 (Recommendation Service, Unit Test, Test #5)

• Test Types: UNIT, INT (Integration), E2E (End-to-End), PERF (Performance)

The matrix ensures:

* 100% coverage of P0 requirements with automated tests
* At least 90% coverage of P1 requirements with automated tests
* Clear identification of manual test procedures for requirements that cannot be fully automated
* Traceability of cross-service integration points and their associated tests

## 5.3 Change Management Process

When modifications are proposed to the system, the traceability matrix facilitates:

• Impact Analysis

• Identifying all affected requirements, code components, and tests

• Estimating scope of changes across microservices

• Determining potential risks and dependencies

• Change Request Documentation

• CR-[Year]-[Month]-[Number] identification format

• Links to affected requirements and justification

• Required approvals based on impact severity

• Implementation Verification

• Ensuring all impacted tests are updated or created

• Validating that changes maintain backward compatibility where required

• Confirming documentation updates reflect the changes

## 5.4 Traceability Tools and Processes

The project will employ the following tools to maintain the traceability matrix:

• Integrated Requirements Management Tool

• Central repository for requirements, tests, and their relationships

• Automated alerts for potential coverage gaps

• Visualization of dependency networks between services

• CI/CD Pipeline Integration

• Automated verification of test coverage against requirements

• Generation of traceability reports during build processes

• Blocking of deployments if critical requirements lack test coverage

## 5.5 Maintaining the Traceability Matrix

To ensure the traceability matrix remains accurate and useful throughout the project lifecycle:

* Weekly automated audits identify orphaned requirements or tests
* Change review process includes mandatory traceability updates
* Quarterly comprehensive reviews validate end-to-end traceability
* Traceability metrics are included in project status reports

This comprehensive approach to traceability ensures that our microservices architecture maintains high quality and alignment with requirements throughout development and beyond.

## 6. Permission Matrix

The Permission Matrix defines the access control structure for CRAD’s microservices architecture, ensuring data security and appropriate functional access across different user types. This comprehensive approach implements the principle of least privilege while maintaining usability for each role.

## 6.1 Role Definitions

The system implements four primary roles with distinct permission levels:

**super\_admin**

• Full system access with no restrictions

• Ability to create/modify all roles including other super\_admins

• Access to system configuration, metrics, and administrative functions

• Capability to view and modify all farmer data across regions

• Direct access to database management tools and logs

**agronomist**

• View and manage multiple farmer profiles within assigned region(s)

• Create and edit agricultural recommendations

• Access to soil test results and sensor data for assigned farms

• Ability to upload reference materials to Knowledge Service

• Generate reports and analytics across multiple farms

• Limited administrative functions (cannot modify system settings)

**farmer**

• Full access to own farm profile and historical data

• Ability to upload soil tests and connect sensors

• View personalized recommendations and educational content

• Communicate with assigned agronomists

• Access to limited analytics relevant to their farm only

• No cross-farm data access or administrative functions

**viewer**

• Read-only access to specified farm profiles (requires farmer approval)

• View recommendations but cannot modify them

• Access to non-sensitive reference materials

• No ability to upload data or modify any system information

• Typically used for researchers, family members, or stakeholders

## 6.2 Access Control Implementation

The permission system is implemented through multiple layers:

**API Gateway Level**

• JWT-based authentication with role claims

• Token validation and signature verification

• Rate limiting based on role (higher limits for admin/agronomist)

• Initial route filtering before requests reach services

• Centralized audit logging of all authentication events

**Service Level**

• Fine-grained authorization checks within each microservice

• Resource-based permissions (e.g., farm\_id-specific access)

• Attribute-Based Access Control (ABAC) for complex permission scenarios

• Service-to-service authentication using mTLS

• Role-specific field-level data filtering

## 6.3 Detailed Permission Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service/Resource** | **super\_admin** | **agronomist** | **farmer** | **viewer** |
| **Farmer Profile Service** | Create, Read, Update, Delete (All) | Create, Read, Update (Assigned) | Read, Update (Own) | Read (Approved) |
| **Soil & Sensor Service** | Full Access | Create, Read, Update (Assigned) | Create, Read (Own) | Read (Approved) |
| **Recommendation Service** | Full Access | Create, Read, Update | Read, Feedback | Read |
| **Knowledge Service** | Full Access | Create, Read, Update | Read | Read |
| **System Configuration** | Full Access | No Access | No Access | No Access |
| **User Management** | Full Access | Limited (Create Farmers) | No Access | No Access |

## 6.4 Permission Inheritance and Delegation

The system supports hierarchical permission structures:

* Regional permission assignment (agronomists manage specific geographic regions)
* Temporary access delegation (farmers can grant time-limited access to viewers)
* Permission inheritance for organizational structures (e.g., farm cooperatives)
* Role-based access control (RBAC) groups for specialized access patterns

## 6.5 Security Considerations

The permission matrix implementation includes:

* Regular permission audits and reporting
* Automatic privilege revocation after periods of inactivity
* Strict separation of duties for critical operations
* Session management with automatic timeouts based on role sensitivity
* Complete audit trails for permission changes and elevated access events

## 7. Microservices Mapping

The CRAD architecture consists of the following microservices, each with well-defined responsibilities, boundaries, and interaction patterns:

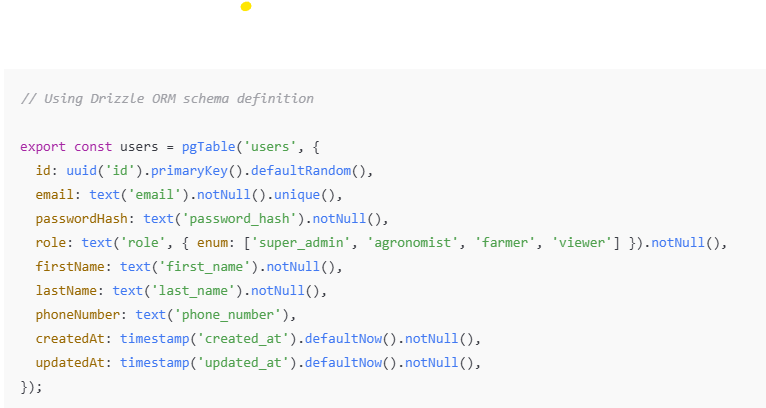
7.1 Service Breakdown

|  |  |  |  |
| --- | --- | --- | --- |
| **Microservice** | **Primary Responsibility** | **Key Data Entities** | **Technical Stack** |
| **API Gateway / BFF** | Request routing, authentication, rate limiting, request aggregation | User sessions, API keys, rate limits | Node.js (NestJS), JWT, Redis |
| **Farmer Profile Service** | Manage farmer profiles, farms, plots, and user accounts | Users, Farms, Plots, Regions | Node.js, Drizzle ORM, PostgreSQL |
| **Soil & Sensor Ingest Service** | Process and store soil test results and IoT sensor data | SoilTests, SensorReadings, SensorDevices | Node.js, Drizzle ORM, PostgreSQL, TimescaleDB extension |
| **Recommendation Service** | Generate personalized farming recommendations using AI | Recommendations, FeedbackRecords | Node.js, OpenAI SDK, Drizzle ORM, PostgreSQL |
| **Knowledge Service** | Manage knowledge base, vector embeddings, and search | Documents, VectorEmbeddings, SearchIndexes | Node.js, PostgreSQL (pgvector), OpenAI Embeddings |
| **Notification Service** | Manage and deliver notifications across channels | NotificationTemplates, DeliveryRecords | Node.js, Redis, PostgreSQL |
| **Analytics Service** | Generate reports and insights from aggregated data | Reports, Metrics, DataAggregations | Node.js, PostgreSQL, Apache Superset |

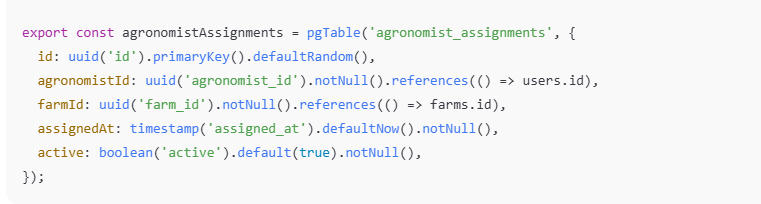
## 7.2 Data Models

Each service maintains its own database with the following core schemas:

**Farmer Profile Service Schema**



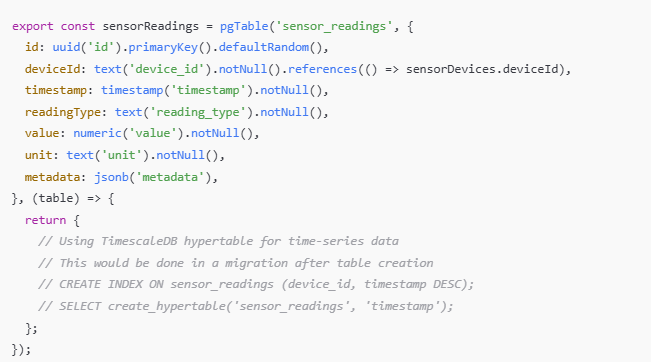




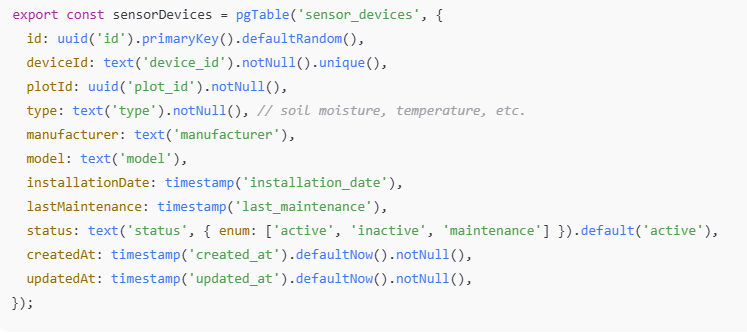


**Soil & Sensor Ingest Service Schema**











**Recommendation Service Schema**







**Knowledge Service Schema**









## 7.3 Service Interaction Patterns

Services in the CRAD architecture communicate through several well-defined patterns:

**API Gateway-Mediated Communication**

• All client requests are routed through the API Gateway

• The Gateway handles authentication, authorization, and rate limiting

• For complex operations, the BFF (Backend for Frontend) pattern aggregates data from multiple services

**Synchronous Service-to-Service Communication**

• Services communicate via REST APIs for immediate responses

• Internal service-to-service calls use mutual TLS authentication

• Circuit breakers and fallback mechanisms handle service failures

• Example: Recommendation Service calling Farmer Profile Service to get plot details

**Asynchronous Event-Based Communication**

• Services publish domain events to a message broker (Apache Kafka)

• Other services subscribe to relevant events

• Ensures eventual consistency across service boundaries

• Example: Soil & Sensor Service publishing a “SoilTestProcessed” event that triggers the Recommendation Service to generate new recommendations

## 7.4 Key Service Interactions

**Farmer Registration Flow**

• Web App → API Gateway → Farmer Profile Service

• Farmer Profile Service creates user account

• Farmer Profile Service publishes “UserCreated” event

• Notification Service consumes event and sends welcome email

• Analytics Service records new user statistics

**Soil Test Processing Flow**

• Web App → API Gateway → Soil & Sensor Ingest Service

• Soil & Sensor Service stores test results

• Soil & Sensor Service publishes “SoilTestProcessed” event

• Recommendation Service consumes event

• Recommendation Service calls Knowledge Service for relevant agricultural knowledge

• Recommendation Service generates new recommendations

• Recommendation Service publishes “RecommendationCreated” event

• Notification Service alerts farmer and agronomist

**AI-Powered Recommendation Generation**

• Recommendation Service retrieves plot data from Farmer Profile Service

• Recommendation Service retrieves soil and sensor data from Soil & Sensor Service

• Recommendation Service calls Knowledge Service for RAG search results

• Recommendation Service processes all inputs through OpenAI orchestration

• Recommendation Service stores generated recommendation

• Notification Service alerts relevant users

## 7.5 API Contract Example





## 7.6 Service Boundaries and Domain Isolation

Each microservice in CRAD is designed around specific domain boundaries:

**Domain-Driven Design Principles**

• Services are organized around business capabilities, not technical functions

• Each service owns its data and business logic

• Domains are isolated with well-defined interfaces

• Shared concepts are defined in a common vocabulary

**Data Ownership and Replication**

• Each service has complete ownership of its domain data

• Required data from other domains is obtained via API calls or event-based replication

• Read models can be maintained for performance optimization

• Example: Recommendation Service keeps a read-only copy of basic plot information from Farmer Profile Service, updated via events

## 7.7 Service Discovery and Registry

The architecture implements service discovery for dynamic service location:

* Kubernetes Service Registry for container-based deployment
* Health check endpoints for each service to verify availability
* Service mesh (Istio) for advanced traffic management and observability
* DNS-based service resolution for development environments

## 8. Review Process

The architecture documentation requires rigorous and systematic review to ensure accuracy, completeness, and alignment with project goals. The following review process has been established:

**Feedback Collection Mechanisms**

• Weekly Architecture Review Meetings: Cross-functional team reviews with representatives from frontend, backend, DevOps, and product teams

• Asynchronous Document Reviews: Using collaborative tools with comment tracking and version history

• Stakeholder Interviews: Regular sessions with business stakeholders to validate that technical decisions align with business requirements

**Continuous Improvement Cycle**

• Documentation Metrics Tracking: Monitoring documentation completeness, accuracy, and usage patterns

• Gap Analysis: Quarterly assessment to identify missing or outdated sections

• Feedback-Driven Updates: Implementing changes based on developer and stakeholder input

• Knowledge Sharing Sessions: Monthly tech talks to ensure the team understands architectural decisions

**Formal Approval Process**

• Architecture Change Request (ACR): Formal proposal for significant architectural changes

• Technical Review Board: Senior engineers and architects evaluate proposed changes

• Impact Assessment: Analysis of how changes affect existing systems, timelines, and resources

• Sign-off Requirements: Approval needed from Tech Lead, Product Owner, and Security Lead

**Documentation Version Control**

• Semantic Versioning: Following SemVer for documentation releases (major.minor.patch)

• Change Log Maintenance: Detailed tracking of all changes with rationale and approvers

• Archiving Strategy: Preserving historical versions for audit and reference purposes

**Compliance and Governance**

• Architecture Compliance Checks: Regular audits to ensure implementation matches documentation

• Security Review Integration: Security team validation for all major architectural decisions

• Regulatory Alignment: Verification that architecture meets relevant agricultural and data privacy regulations

## 9. Tools and Techniques

The CRAD microservices architecture employs the following documentation tools and techniques to maintain clarity, accessibility, and relevance throughout the project lifecycle:

**Standardized Documentation Templates**

• Architecture Decision Records (ADRs): Structured templates capturing the context, decision, alternatives considered, and consequences for each architectural choice

• Service Specification Template: Consistent format for documenting each microservice, including API contracts, data models, and integration points

• Component Diagrams: C4 model templates (Context, Container, Component, Code) for visual representation at various abstraction levels

• Implementation Guides: Step-by-step instructions for developers following established patterns

**Version Control Integration**

• Documentation-as-Code: Architecture documentation stored alongside application code in Git repositories

• Markdown + Mermaid: Using lightweight markup with embedded diagrams for version-controlled documentation

• Branch Protection Rules: Requiring peer reviews for documentation changes

• Automated Validation: CI/CD pipeline checks for broken links, formatting issues, and compliance with documentation standards

• GitOps Workflow: Changes to architecture documentation trigger notification workflows to relevant stakeholders

**Document Generation & Publishing**

• API Documentation: OpenAPI/Swagger specifications automatically generated from code annotations

• Static Site Generation: Using tools like Docusaurus or MkDocs to convert Markdown files into searchable documentation portals

• Living Documentation: Database schema visualization generated directly from Drizzle ORM definitions

• Interactive API Explorer: Generated documentation includes runnable examples in development environments

• Documentation Versioning: Automatic tagging of documentation to match software releases

**Collaboration & Accessibility Tools**

• Real-time Collaboration: Using tools like Notion or Confluence for drafting with multiple contributors

• Knowledge Graph: Semantic linking between related documentation artifacts

• Search Optimization: Full-text search with relevance ranking across all documentation

• Feedback Mechanisms: Embedded feedback collection on all documentation pages

• Accessibility Compliance: Documentation tested for screen reader compatibility and WCAG compliance

**DevOps Integration**

• Infrastructure-as-Code Documentation: Automated extraction of comments from Terraform/Kubernetes manifests

• Runbook Generation: Semi-automated creation of operational procedures from deployment configurations

• Monitoring Dashboard Documentation: Auto-generated explanations of metrics and alerting thresholds

• Disaster Recovery Documentation: Automated testing and validation of recovery procedures

## 10. Milestone Documentation

The CRAD microservices project follows a structured milestone approach to ensure systematic development and delivery. Each milestone represents a significant project phase with specific deliverables, success criteria, and documentation requirements.

## Milestone 1: Architecture Foundation (August 15, 2025)

**Deliverables**

• Finalized system architecture diagrams (C4 model)

• Service boundary definitions and communication patterns

• Technology stack documentation with version requirements

• Development environment setup guides

**Success Criteria**

• Architecture review completed with stakeholder sign-off

• Development environments successfully provisioned

• CI/CD pipeline initialized and documented

**Milestone 2: Core Services Implementation (September 1, 2025)**

**Deliverables**

• Farmer Profile Service implementation with Drizzle ORM schema

• API Gateway/BFF service with authentication flows

• Data models and database migration scripts

• Service-to-service communication protocols

**Success Criteria**

• End-to-end tests passing for core user journeys

• Performance benchmarks meeting specified requirements

• Security review completed for authentication mechanisms

**Milestone 3: Data Processing Services (September 15, 2025)**

**Deliverables**

• Soil & Sensor Ingest Service implementation

• Vector database integration for agricultural data

• Data validation and transformation pipelines

• Monitoring and alerting configuration

**Success Criteria**

• Successful ingestion of test sensor data sets

• Data consistency verification across services

• Observability dashboards operational

**Milestone 4: AI Recommendation Engine (September 30, 2025)**

**Deliverables**

• OpenAI integration for agricultural recommendations

• RAG system implementation with vector search

• Recommendation caching and delivery mechanisms

• AI model performance metrics and evaluation framework

**Success Criteria**

• Recommendation quality assessment with domain experts

• Performance testing under various load conditions

• Fallback mechanisms verified for resilience

**Milestone 5: Final Integration and Deployment (October 7, 2025)**

**Deliverables**

• Next.js frontend implementation with responsive UI

• End-to-end integration tests across all services

• Production deployment documentation

• User documentation and support materials

**Success Criteria**

• System performance meeting SLAs in production-like environment

• Security penetration testing completed

• User acceptance testing signed off

• Disaster recovery procedures verified

## Documentation Standards Across All Milestones

• Each milestone requires:

• Technical design documents with implementation details

• API specifications using OpenAPI/Swagger

• Test plans and results documentation

• Deployment and rollback procedures

• Known limitations and technical debt tracking

This comprehensive documentation will serve as a single source of truth for all stakeholders involved in the CRAD microservices project, ensuring clear communication and alignment throughout the development process. All milestone documentation will be version-controlled, peer-reviewed, and made accessible through the project documentation portal.