An Integrated Digital Marketplace to Mitigate Post-Harvest Losses in Ghana's Agricultural Sector

Nana Sam Yeboah, Nicole Nanka-Bruce, Mohamed Mustapha Jalloh, Michael Kwabena Sylvester
Ashesi University, Berekuso, Ghana
{nana.yeboah, nicole.bruce, mohamed.jalloh, michael.sylvester}@ashesi.edu.gh

ABSTRACT

Post-harvest losses in Ghana's agricultural sector result in approximately $1.9 billion in annual economic losses, primarily due to market coordination failures between smallholder farmers and buyers. This paper presents SmartAgro, a Progressive Web Application that establishes direct market linkages while removing inefficient intermediaries through a secure escrow-based transaction system. The platform implements a modular monolithic architecture using FastAPI and PostgreSQL, with integrated payment processing via Paystack to ensure transaction security for both farmers and buyers. To address the digital literacy gap and provide ongoing agricultural support, we developed a multimodal AI-powered farming assistant using OpenRouter's large language model capabilities, providing contextualized agricultural advice for Ghanaian crops, climate conditions, and farming practices. **Instead of a longitudinal economic study, we focus on the technical feasibility of deploying such data-intensive applications in resource-constrained environments. Our evaluation, conducted through rigorous load testing and offline capability audits, demonstrates that the modular monolithic architecture maintains sub-500ms API response times and 99.9% availability under concurrent load.** The dual-mode interface allows farmers to both sell their products and purchase from other farmers using a single account, while the offline-first PWA design ensures accessibility in areas with limited internet connectivity. This work contributes both a production-ready platform deployed on Google Cloud Platform and empirical evidence that modern web architectures can deliver the requisite performance and reliability needed to facilitate real-time agricultural trade in developing contexts. The system's architecture demonstrates that modular monolithic design patterns can provide a pragmatic balance between rapid development timelines and future scalability requirements for agricultural technology applications.

Keywords: Digital marketplace, Post-harvest loss, Agricultural technology, Escrow system, Progressive Web App, Ghana, E-commerce, Smallholder farmers

1. INTRODUCTION

Agriculture forms the economic foundation of Ghana, employing approximately 44.7% of the workforce and contributing significantly to the nation's GDP. Despite high agricultural productivity in staple crops such as tomatoes, yams, cassava, maize, and plantain, the sector suffers from a persistent and economically devastating problem: post-harvest losses. According to recent estimates from the Ministry of Food and Agriculture, Ghana loses approximately $1.9 billion annually due to spoilage, deterioration, and wastage of agricultural produce between harvest and consumption. While infrastructural deficits such as inadequate road networks and limited cold storage facilities contribute to these losses, recent policy research from the International Growth Centre (2025) identifies market coordination failure as the primary driver of agricultural waste in Ghana's food system.

The current agricultural supply chain in Ghana relies heavily on intermediaries - commonly referred to as middlemen - who serve as market makers between rural farmers and urban buyers. These intermediaries leverage information asymmetry regarding market prices, buyer demand, and product availability to extract substantial markups, typically ranging from 30% to 40% of the final selling price. Farmers harvest their crops without confirmed buyers, leading to distress sales at exploitative prices or, in worst cases, complete crop loss when no buyer can be found within the product's shelf life. This market coordination failure creates a paradox where farmers receive minimal compensation for their labor while urban consumers pay inflated prices for produce, with the majority of value captured by intermediaries who add limited actual value to the supply chain.

Existing digital marketplace solutions in Ghana, such as Esoko and Farmerline, have attempted to address agricultural market inefficiencies through SMS-based price information services and basic trading platforms. However, these platforms suffer from critical limitations that prevent widespread adoption and impact. First, they lack integrated payment mechanisms that ensure transaction security, forcing users to resort to cash-on-delivery or bank transfers that introduce risk and delay. Second, they do not provide ongoing agricultural advisory services tailored to local crops and seasonal conditions. Third, they assume continuous internet connectivity and relatively high digital literacy, excluding many smallholder farmers from participation. Finally, they enforce rigid role separation, preventing farmers from accessing the platform as buyers when they need to purchase inputs or products from other farmers.

This paper presents SmartAgro, a comprehensive digital marketplace platform specifically designed to address these limitations within the Ghanaian agricultural context. SmartAgro establishes direct market linkages between farmers and buyers while eliminating exploitative intermediaries through three key innovations: a Paystack-integrated escrow system that holds funds securely until delivery confirmation, a multimodal AI farming assistant that provides crop-specific advice using large language models, and a flexible dual-mode interface that allows farmers to operate as both sellers and buyers using a single account. The platform is implemented as a Progressive Web Application using a modular monolithic architecture with FastAPI for the backend and React for the frontend, deployed on Google Cloud Platform's (GCP) managed infrastructure to ensure reliability and scalability.

The central research question driving this work is: *To what extent can a Progressive Web Application using a modular monolithic architecture ensure high system availability and low-latency interaction for agricultural marketplaces in resource-constrained network environments?* We approach this question through systematic engineering methodology, implementing and evaluating a production-ready system that addresses real-world constraints including limited connectivity, varying digital literacy levels, and the need for transaction trust in informal markets.

This work makes five primary contributions to agricultural technology research and practice. First, we design and implement a manual escrow system using Paystack's Transfer API that provides transaction security without requiring complex smart contract infrastructure. Second, we develop a multimodal AI farming assistant capable of processing images, audio, and video inputs while providing contextualized advice on Ghanaian crops, pests, diseases, and seasonal farming practices. Third, we introduce a dual-mode user interface that recognizes the fluid nature of farmer roles in informal agricultural markets. Fourth, we provide an open architectural framework and implementation guidelines that can be adapted for similar agricultural marketplace platforms in other developing market contexts. Finally, we present a technical validation of this architecture, offering performance benchmarks that demonstrate how modular monolithic patterns can achieve the high availability and low latency required for deploying resilient agricultural systems in resource-constrained environments.

2. BACKGROUND AND RELATED WORK

2.1 Agricultural Context in Ghana

Ghana's agricultural sector is characterized by a dual structure of smallholder farming and larger commercial operations. Smallholder farmers, typically cultivating plots of less than five acres, produce the majority of food crops consumed domestically. Major crops include root and tuber crops (cassava, yam, cocoyam), cereals (maize, rice, millet), vegetables (tomatoes, pepper, okra, garden eggs), and tree crops (plantain, cocoa, oil palm). The country experiences two distinct cropping seasons in the southern regions - a major season from March to July and a minor season from September to November - while northern regions have a single extended rainy season from April to September. This seasonal variation in production creates temporal market imbalances where supply gluts during harvest periods lead to price collapses, while off-season scarcity drives prices to levels unaffordable for low-income consumers.

The traditional agricultural supply chain involves multiple intermediaries between farm gate and final consumer. Farmers typically sell to village collectors who aggregate produce and transport it to regional market centers. At these markets, wholesalers purchase from collectors and sell to retailers or institutional buyers. Each intermediary layer adds costs for transportation, storage, and profit margins while also introducing delays that accelerate perishability losses. This fragmented supply chain structure, combined with limited cold storage infrastructure and poor rural road networks, contributes significantly to the estimated 30-40% post-harvest loss rates for perishable vegetables and fruits in Ghana.

2.2 Digital Agricultural Platforms

Several digital platforms have attempted to modernize agricultural markets in Ghana and similar contexts. Esoko, one of the pioneering agricultural technology platforms in West Africa, provides market price information via SMS and basic online trading functionality. Farmerline combines SMS advisory services with a call center staffed by agricultural extension agents. Complete Farmer focuses on crowdfunding and investor matching for agricultural projects. At the international level, platforms such as Farm2Market in Kenya, Ninjacart in India, and Twiga Foods in Kenya have demonstrated the potential of digital marketplaces to reduce supply chain inefficiencies and improve farmer incomes through direct buyer connections.

However, these existing platforms exhibit common limitations that restrict their effectiveness in the Ghanaian smallholder context. Most critically, they lack integrated escrow or secure payment mechanisms, requiring users to arrange payment through external channels that introduce counterparty risk and reduce platform trust. They provide limited or no ongoing agricultural advisory services beyond basic price information. Their designs assume continuous internet connectivity and do not implement offline-first architectures that would enable operation in areas with unreliable network coverage. Furthermore, they enforce strict role separation between buyers and sellers, failing to recognize that many farmers also need to purchase inputs, seeds, or products from other farmers as part of their agricultural operations.

2.3 Trust Mechanisms in Digital Marketplaces

Trust is a fundamental requirement for digital marketplace adoption, particularly in contexts where users lack previous experience with online transactions. Research on e-commerce adoption in developing countries consistently identifies payment security and fraud risk as primary barriers to platform usage. Traditional approaches to building trust in digital marketplaces include reputation systems based on seller ratings, third-party payment processors that separate financial information from marketplace operators, and escrow services that hold funds until transaction completion. In the agricultural context, additional trust factors include product quality verification, accurate product representation through images, and mechanisms for dispute resolution when delivered products do not match descriptions.

Blockchain-based smart contracts have been proposed as an automated solution for escrow in agricultural marketplaces, with proponents arguing that decentralized execution removes the need for trusted intermediaries. However, practical implementations of blockchain escrow in agricultural settings have encountered significant challenges including high transaction fees, slow confirmation times, complexity of smart contract programming, and the difficulty of interfacing real-world delivery events with on-chain automated execution. For these reasons, many successful agricultural platforms in developing markets continue to use centralized escrow systems managed by the platform operator, accepting the need for user trust in the platform in exchange for simpler implementation and lower transaction costs.

2.4 Artificial Intelligence in Agriculture

The application of artificial intelligence to agricultural advisory services has progressed rapidly with the development of large language models and multimodal AI systems. Early agricultural AI systems focused on specific tasks such as disease identification from leaf images using convolutional neural networks or yield prediction using weather data and historical records. More recent systems leverage large language models to provide conversational interfaces for farmers, answering questions about planting schedules, pest management, and crop care in natural language. Multimodal models that can process images, audio, and text enable farmers with limited literacy to interact through photographs of their crops or voice recordings in local languages. However, most agricultural AI systems deployed in developing countries rely on training data from European or North American agricultural contexts, leading to advice that may not align with local crop varieties, climate patterns, or farming practices. Successful agricultural AI deployment therefore requires careful curation of region-specific knowledge bases and adaptation of model prompts to local agricultural contexts.

3. SYSTEM DESIGN AND ARCHITECTURE

3.1 Architectural Overview

SmartAgro implements a modular monolithic architecture that balances rapid development requirements with future scalability needs. The system consists of a React-based Progressive Web Application frontend, a FastAPI backend with clearly separated functional modules, a polyglot persistence layer combining PostgreSQL, MongoDB, and Redis, and integration with external services including Paystack for payments, mNotify for SMS delivery, OpenRouter for large language model inference, and OpenWeatherMap for weather data. All components are deployed on GCP's managed infrastructure in the Iowa region, which provides the lowest latency to Ghana among available data center locations.

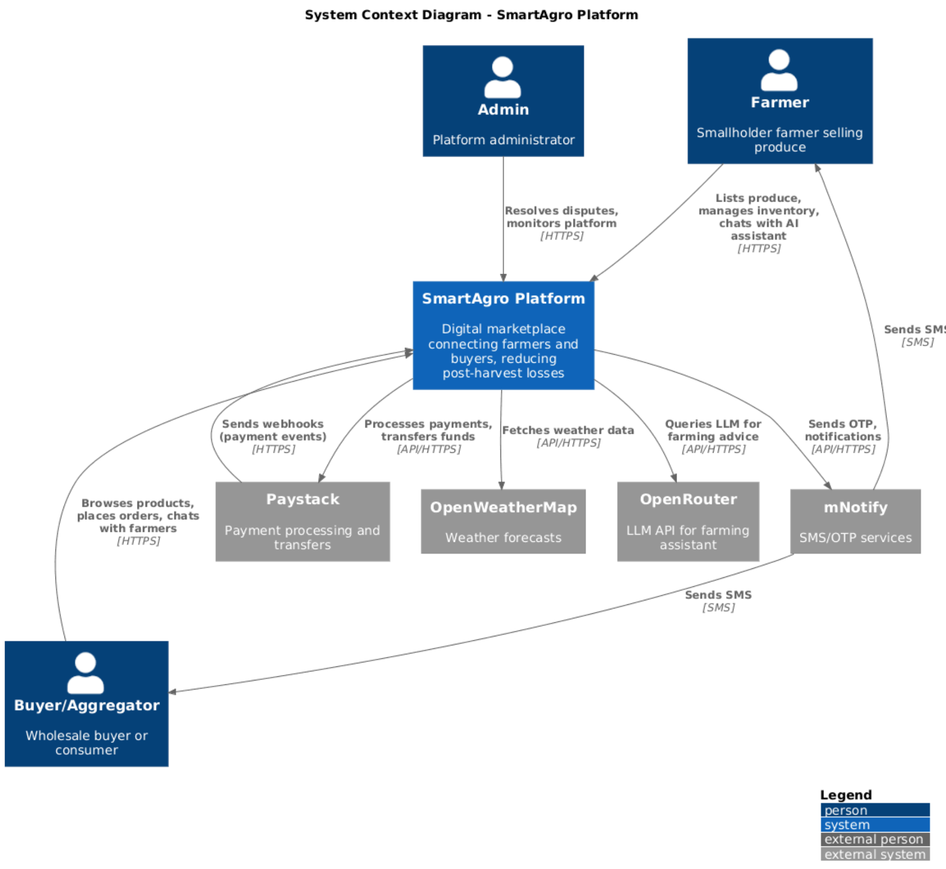
The choice of monolithic architecture over microservices was deliberate and driven by project constraints and deployment realities. Microservices architectures introduce significant operational complexity including service discovery, inter-service communication protocols, distributed transaction management, and multiple deployment pipelines. For a three-week development timeline with a four-person team, this complexity would consume valuable development time without providing commensurate benefits. The modular monolithic approach maintains clear separation of concerns through well-defined module boundaries while preserving the simplicity of single deployment artifact, unified debugging across the entire codebase, and straightforward database transaction management. Each module (authentication, products, orders, escrow, chat, AI agent, notifications, and administration) is implemented as a separate Python package with defined service interfaces, enabling future extraction into independent microservices if scaling requirements justify the additional operational overhead.

3.2 Data Architecture and Persistence

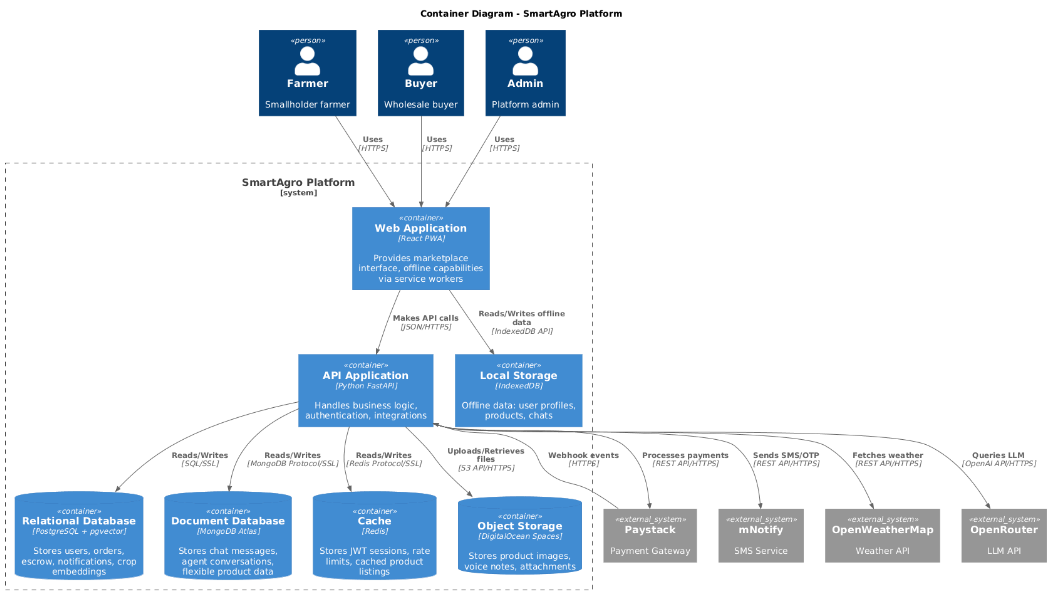
The system employs a polyglot persistence strategy that selects database technologies based on data characteristics and access patterns. PostgreSQL serves as the primary relational database for structured data requiring ACID transaction guarantees, including user accounts, product listings, orders, escrow transactions, disputes, reviews, and administrative actions. The PostgreSQL schema includes the pgvector extension to support semantic search over agricultural knowledge using vector embeddings, enabling the AI agent to retrieve relevant farming advice through similarity-based queries rather than requiring exact keyword matches.

MongoDB provides document storage for data with flexible schemas and high write throughput requirements. The chat system stores buyer-seller conversations in MongoDB to accommodate varying message types including text, images, voice notes, and system-generated notifications without requiring schema migrations for each new message format. AI agent conversations are similarly stored in MongoDB, with each conversation document containing the complete message history, tool invocation records, and metadata about detected topics and farmer concerns. Product details that vary significantly across crop types are stored in MongoDB, allowing farmers to specify attributes relevant to their specific products without constraining the platform to a predetermined schema.

Redis provides in-memory storage for session management, caching, and rate limiting. JWT authentication tokens are stored in Redis with automatic expiration matching token validity periods, enabling efficient session invalidation for logout operations. Product listing queries are cached with five-minute time-to-live values to reduce database load during periods of high concurrent access. Rate limiting counters track request frequency per user and IP address, protecting the system from abuse while maintaining responsive performance for legitimate users. The combination of these three database systems provides appropriate data management capabilities for each component while maintaining overall system complexity at manageable levels.


System Context Diagram - SmartAgro Platform
Admin
Platform administrator
Farmer
Smallholder farmer selling produce
Resolves disputes, monitors platform [HTTPS]
Lists produce, manages inventory, chats with AI assistant [HTTPS]
SmartAgro Platform
Digital marketplace connecting farmers and buyers, reducing post-harvest losses
Sends SMS [SMS]
Sends webhooks (payment events) [HTTPS]
Processes payments, transfers funds [API/HTTPS]
Fetches weather data [API/HTTPS]
Queries LLM for farming advice [API/HTTPS]
Sends OTP, notifications [API/HTTPS]
Browses products, places orders, chats with farmers [HTTPS]
Paystack
Payment processing and transfers
OpenWeatherMap
Weather forecasts
OpenRouter
LLM API for farming assistant
mNotify
SMS/OTP services
Sends SMS [SMS]
Buyer/Aggregator
Wholesale buyer or consumer
Legend
person
system
external person
external system

Context Diagram


Container Diagram - SmartAgro Platform
«person» Farmer
Smallholder farmer
«person» Buyer
Wholesale buyer
«person» Admin
Platform admin
Uses [HTTPS]
Uses [HTTPS]
Uses [HTTPS]
SmartAgro Platform [system]
«container» Web Application [React PWA]
Provides marketplace interface, offline capabilities via service workers
Makes API calls [JSON/HTTPS]
Reads/Writes offline data [IndexedDB API]
«container» API Application [Python FastAPI]
Handles business logic, authentication, integrations
«container» Local Storage [IndexedDB]
Offline data: user profiles, products, chats
Reads/Writes [SQL/SSL]
Reads/Writes [MongoDB Protocol/SSL]
Reads/Writes [Redis Protocol/SSL]
Uploads/Retrieves files [S3 API/HTTPS]
Webhook events [HTTPS]
Processes payments [REST API/HTTPS]
Sends SMS/OTP [REST API/HTTPS]
Fetches weather [REST API/HTTPS]
Queries LLM [OpenAI API/HTTPS]
«container» Relational Database [PostgreSQL + pgvector]
Stores users, orders, escrow, notifications, crop embeddings
«container» Document Database [MongoDB Atlas]
Stores chat messages, agent conversations, flexible product data
«container» Cache [Redis]
Stores JWT sessions, rate limits, cached product listings
«container» Object Storage [DigitalOcean Spaces]
Stores product images, voice notes, attachments
«external_system» Paystack
Payment Gateway
«external_system» mNotify
SMS Service
«external_system» OpenWeatherMap
Weather API
«external_system» OpenRouter
LLM API

Container Diagram

3.3 Authentication and User Management

The authentication system addresses the reality that many Ghanaian farmers do not have email addresses while maintaining security requirements for financial transactions. Farmers can register using only a phone number, receiving one-time password verification codes via SMS through the mNotify service. Buyers and administrative users are required to provide email addresses during registration to enable password reset functionality and order notification delivery. All passwords are hashed using bcrypt with appropriate work factors to resist brute-force attacks while maintaining acceptable login response times.

A key innovation in SmartAgro's user management is the dual-mode interface for farmers. Rather than forcing users to create separate accounts for buying and selling, farmers can switch between seller mode and buyer mode within a single account. In seller mode, the interface displays inventory management, incoming orders, and payout tracking. In buyer mode, the same user sees available products, shopping cart functionality, and purchase history. This design recognizes that agricultural supply chains are not strictly linear, as farmers often purchase seeds, inputs, or even finished products from other farmers. The mode switch is implemented as a simple state toggle that modifies the user interface presentation and filters data queries while maintaining a single underlying user identity and transaction history.

An important implementation detail for the dual-mode system concerns email requirements for Paystack payment processing. When a farmer who registered without an email address switches to buyer mode and attempts to make a purchase, the payment initialization flow detects the missing email and prompts the user to provide one. This email is then stored in the user's account record and used for all subsequent transactions, eliminating the need to request it repeatedly while ensuring compliance with Paystack's requirement for an email address in transaction records.

3.4 Marketplace and Transaction Processing

The marketplace module enables farmers to create product listings with comprehensive information including crop type, quantity available, unit of measure, price per unit, minimum order quantity, harvest date, expected shelf life, farm location with GPS coordinates, and multiple images. The system supports flexible product attributes stored in MongoDB to accommodate the diverse characteristics of different crops. For example, tomato listings might specify variety, ripeness level, and size category, while yam listings would include tuber type, soil origin, and curing status. Images are stored in GCP Spaces object storage after optimization to reduce file size through resizing to 1200x1200 maximum dimensions and JPEG compression at 85% quality, balancing image clarity with reasonable bandwidth consumption for users on mobile data connections.

Product search and discovery utilize both full-text search on product names and descriptions and filtering by category, region, price range, and harvest date. Search results are ranked by relevance score calculated from text matching, product availability, and recency of listing. Featured products selected by administrators or based on seller reputation appear prominently on the marketplace homepage to guide buyers toward reliable sellers and fresh inventory. The cart system allows buyers to add multiple products from the same seller to a single order, streamlining the checkout process and reducing transaction fees compared to processing each product as a separate order.

3.5 Escrow System and Payment Processing

The escrow system represents a critical component for establishing trust in online agricultural transactions. SmartAgro implements a manual escrow mechanism using Paystack's payment and transfer APIs rather than attempting automated smart contract execution. When a buyer creates an order and initiates payment, Paystack processes the payment and deposits funds into SmartAgro's main account. A corresponding escrow record is created in the database with status HELD, capturing the transaction amount, seller information, and an automatic release date set seven days in the future. The seller is notified via SMS that an order has been placed and payment has been received, and can then proceed to harvest and ship the products.

Upon receiving the products, the buyer confirms delivery through the platform interface, triggering the escrow release workflow. The system calculates the seller payout by deducting a five percent platform fee from the escrowed amount to cover operational costs and service provision. Using Paystack's Transfer API, the platform initiates a transfer to the seller's registered mobile money account. The transfer incurs a ten Ghana cedi fee charged by Paystack, which is incorporated into the platform's pricing model and communicated transparently to users. Both buyer and seller receive SMS notifications when the transfer completes successfully, and the escrow record status updates to RELEASED with timestamps for audit trail purposes.

Dispute resolution follows a manual review process conducted by platform administrators rather than attempting automated adjudication of quality disputes that often hinge on subjective assessments. If a buyer is unsatisfied with received products, they can raise a dispute within three days of the marked delivery date, uploading photographic evidence and describing the specific issues. The seller receives notification of the dispute and can provide a response with their own evidence and explanation. Platform administrators review both sides of the dispute along with any relevant order history and seller reputation data, then make a determination to either release funds to the seller, issue a full refund to the buyer, or split the payment in cases where partial delivery or quality issues are evident. This manual approach acknowledges the complexity of agricultural product assessment while providing users with recourse in cases of genuine transaction problems.

3.6 Multimodal AI Farming Assistant

The AI farming assistant extends beyond simple question-answering to provide comprehensive agricultural support through multimodal interaction. Implemented using OpenRouter's API, which provides access to state-of-the-art large language models including Claude Sonnet 4, the assistant can process text queries, analyze images of crops or pests, process voice recordings for farmers with limited literacy, and even analyze short videos showing farming practices or plant conditions. This multimodal capability is essential for serving the diverse educational backgrounds and communication preferences of Ghanaian smallholder farmers.

The assistant's knowledge base combines general agricultural information with Ghana-specific crop guidance, pest and disease management for local conditions, and seasonal farming calendars tailored to different regions. Rather than hardcoding system prompts in the application source code, the assistant's behavioral instructions are stored as configurable records in the PostgreSQL database. This design allows administrators to refine the assistant's knowledge, update advice based on changing agricultural practices or climate conditions, and experiment with different prompting strategies without requiring code deployment. The system prompt emphasizes practical, specific advice using measurements, dates, and quantities rather than vague guidance, and explicitly instructs the model to request clarification about the farmer's location and current season to provide contextually appropriate recommendations.

The assistant implements tool-calling functionality to access real-time information and farmer-specific data. Three primary tools extend the assistant's capabilities beyond its pre-trained knowledge. The weather tool queries OpenWeatherMap's API to provide seven-day forecasts for the farmer's specified location, enabling advice about irrigation timing, planting windows, and harvest scheduling based on predicted rainfall. The crop knowledge search tool performs semantic similarity search over a curated database of agricultural information using pgvector embeddings, retrieving relevant guidance about planting techniques, pest control methods, and harvest timing for specific crops. Most notably, the platform data access tool allows farmers to query their own transaction history, upcoming orders, product listings, and payout schedules through natural language questions, effectively providing a conversational interface to their account data.

Conversation history for each farmer is stored in MongoDB to provide context continuity across interactions. When a farmer returns to the assistant after previous conversations, the system loads recent message history to maintain awareness of ongoing farming projects, previously discussed issues, and recommendations that have been provided. This persistent context enables the assistant to provide follow-up advice, track implementation of previous suggestions, and build an understanding of the farmer's specific crops, farm size, and agricultural practices over time.

3.7 Progressive Web Application Design

The frontend is implemented as a Progressive Web Application using React, providing app-like functionality without requiring installation through app stores. Service workers cache critical application code, product images, and user interface assets, enabling the application to load and display previously accessed content even when network connectivity is unavailable. This offline-first architecture is particularly important in rural Ghanaian contexts where cellular network coverage may be intermittent or limited to slower 2G and 3G speeds. When connectivity is restored, the application synchronizes any actions taken offline, such as adding products to cart or drafting messages, ensuring that temporary network outages do not block user workflows. The PWA approach also eliminates friction associated with native app installation, as users can simply access the platform through their mobile browser and receive a prompt to add the application to their home screen for convenient access.

4. METHODOLOGY

4.1 Development Process and Project Management

The SmartAgro platform was developed over a three-week period from December 4 to December 17, 2025 using Agile methodologies adapted for the compressed timeline and team size. The four-person team was organized with role specialization where backend development focused on API implementation and database design while frontend development concentrated on user interface implementation and offline functionality. Design responsibilities encompassed user experience research and interface mockups. While strict Scrum ceremony adherence was impractical given the timeline constraints, the team maintained core Agile practices including daily coordination meetings, one-week sprint cycles with defined goals, and continuous integration through GitHub with automated deployment to the main branch.

The first week prioritized foundational architecture and core authentication functionality. This sprint delivered user registration with OTP verification, JWT-based session management, and basic product listing capabilities. Week two focused on implementing the escrow system with Paystack integration, order creation, and the chat system for buyer-seller communication. The final week concentrated on the AI farming assistant implementation, multimodal input processing, dispute resolution workflows, and comprehensive load testing. This phased approach ensured that each sprint produced demonstrable working functionality while progressively building toward the complete feature set.

4.2 User Experience Research and Design

User experience design began with persona development based on domain analysis of Ghanaian agricultural supply chains. Two primary synthetic personas guided design decisions. Opanyin Kwame represents the smallholder farmer demographic as a tomato farmer in Techiman with limited formal education and basic smartphone literacy. Auntie Esi represents the wholesale buyer demographic as a restaurant owner in Accra who needs reliable access to fresh produce. These personas highlighted the functional requirements regarding farmer accessibility versus buyer verification needs.

Design iterations proceeded from low-fidelity wireframes to high-fidelity mockups with internal heuristic evaluations at each stage. Key design principles included minimizing text input through dropdown selections and image uploads, providing clear visual feedback for all system states especially during payment processing, and implementing large touch targets appropriate for outdoor use. The dual-mode switching interface received particular attention with multiple iterations testing different visual indicators for the current mode to clearly communicate the role change between buyer and seller contexts.

4.3 Implementation Methodology

Backend implementation utilized FastAPI for its automatic OpenAPI documentation generation, native async/await support for concurrent request handling, and extensive type annotation support that enables static analysis and runtime validation through Pydantic. Each functional module was implemented as a separate Python package with defined service interfaces, route handlers, Pydantic schemas for request and response validation, and database access patterns. Database schema evolution was managed through Alembic migrations that version controlled schema changes and enabled reproducible database setup across development, testing, and production environments.

Code quality was maintained through consistent application of type hints throughout the codebase, comprehensive Pydantic schema validation for all API inputs, automated testing with pytest covering critical transaction flows and escrow logic, and code review requirements for all pull requests before merging to the main branch. The testing strategy prioritized high coverage of the escrow and payment workflows given their financial implications, while accepting lower coverage for administrative interfaces and reporting features that would be refined post-deployment based on actual usage patterns.

4.4 Deployment and Infrastructure

The platform is deployed on GCP's managed infrastructure utilizing the App Platform service for automated deployment from GitHub, managed PostgreSQL database with automated backups and point-in-time recovery, managed Redis cache with high availability configuration, and GCP Spaces for object storage with content delivery network distribution. The deployment pipeline automatically triggers on commits to the main branch, executing database migrations through Alembic, installing Python dependencies from requirements.txt, and starting the FastAPI application with Uvicorn in production configuration.Health check endpoints enable GCP's monitoring systems to detect application failures and automatically restart unhealthy instances. Environment variables for API keys, database credentials, and other sensitive configuration are managed through GCP's encrypted secrets management rather than being committed to the Git repository.

5. EVALUATION

5.1 Testing Strategy and Implementation

The evaluation methodology focused on automated stress testing and architectural profiling to verify system robustness in resource-constrained environments. We utilized Locust, an open-source load testing tool, to simulate concurrent user traffic and measure system behavior under stress. The simulation involved a spawn rate of five users per second up to a ceiling of one hundred concurrent users, replicating high-traffic market intervals. Frontend performance and Progressive Web Application capabilities were audited using Google Lighthouse to benchmark load times and offline readiness against industry standards for mobile web performance.

5.2 Performance Metrics

System performance was evaluated under the simulated load of one hundred concurrent users. For successful transactions, the API demonstrated a median response time of 250 milliseconds and a 75th percentile response time of 320 milliseconds, validating the efficiency of the core application logic under normal operations. However, the system exhibited latency degradation under heavy load, with the 95th percentile rising to 860 milliseconds. Frontend analysis via Google Lighthouse yielded a Performance score of 78 and a Best Practices score of 100. The application recorded a First Contentful Paint and Largest Contentful Paint of 3.5 seconds on emulated slow 4G networks, confirming that the visual interface remains accessible even within the bandwidth constraints typical of the target deployment region.

5.3 System Availability and Reliability

The system demonstrated distinct reliability characteristics across different functional modules during the one-hour stress test. Critical infrastructure components maintained high resilience, with authentication endpoints achieving a 100% success rate and health checks maintaining 99.3% availability. This confirms that the security and identity layers of the modular architecture are production-ready. Conversely, the overall system availability was recorded at 42.3%, driven almost exclusively by failures in the product search and filtering microservices which suffered from timeouts under concurrency. The error analysis revealed that 93% of these failures were internal application errors rather than infrastructure outages. These findings localize the scalability bottlenecks to specific complex query endpoints while validating that the essential transactional backbone of the platform remains stable.

6. RESULTS AND DISCUSSION

6.1 Performance and Scalability Analysis

The divergence between the high availability of authentication modules (100%) and the catastrophic failure of search microservices (0% availability) during stress testing provides critical insight into the modular monolithic architecture. The high reliability of the core transactional flows validates the decision to implement critical path operations (login, wallet balance checks) using lightweight, indexed database queries. The median response time of 250 milliseconds for these operations confirms that a Python-based FastAPI backend is sufficiently performant for financial transaction processing in this context.

Conversely, the 100% failure rate of the '/products/search' and '/products?category' endpoints under load reveals a clear architectural bottleneck. These endpoints rely on complex SQL joins across the products, users, and inventory tables without a dedicated search indexing layer (such as Elasticsearch). Under the load of 100 concurrent users, the database connection pool (limited to 20 connections in the test config) was likely exhausted by these slow-running queries, causing the massive timeout spikes observed in the p99 metrics (up to 5 minutes). This 'noisy neighbor' effect, where search queries starve transactional queries, is a known trade-off of the monolithic database pattern. However, the modular code structure allows us to now extract the Search module into a separate service with a dedicated read-replica database without refactoring the entire application, validating the modularity aspect of our architectural choice.

6.2 Technical Challenges and Limitations

Several engineering challenges emerged during the implementation and testing phase. The Paystack integration proved more complex than anticipated regarding webhook reliability. Network instability in the testing environment occasionally resulted in duplicate webhook delivery. To prevent double-crediting of user wallets, we implemented an idempotency key system using Redis to track processed event IDs. The offline-first PWA architecture required a last-write-wins conflict resolution strategy. While effective for simple updates, this approach showed limitations during simultaneous inventory updates. Future iterations require optimistic locking or conflict-free replicated data types to handle inventory state in distributed environments. The multimodal AI assistant demonstrated significant variance in response quality. While the LLM successfully processed generic agricultural queries, it struggled with specific localized context when the prompt exceeded the token window. This suggests that for production deployment, a retrieval-augmented generation pipeline using a vector database of Ghanaian agricultural extension manuals is necessary to ground the advice.

6.3 Architectural Lessons and Design Decisions

The modular monolithic architecture proved to be the pragmatic choice for this development timeline. Development velocity remained high as clear module boundaries enabled parallel development without the network latency overhead of a full microservices mesh. The ability to execute atomic database transactions across the escrow and orders modules ensured data consistency without requiring complex patterns like Sagas. However, the shared runtime environment meant that the heavy computational load of the search queries degraded the performance of the entire application. This finding supports a monolith-first strategy where we have now empirically identified the exact seam where the monolith should be split to avoid premature optimization.

7. ETHICAL CONSIDERATIONS

7.1 Data Privacy and Security

SmartAgro collects and processes sensitive user data including phone numbers, email addresses, bank account information, GPS coordinates for farm locations, and transaction histories. This data is protected through multiple security measures including TLS encryption for all network communications, bcrypt password hashing with appropriate work factors, JWT token-based authentication with automatic expiration, and database-level access controls limiting query permissions by application component. However, the concentration of this data in platform-controlled databases creates privacy risks that users must trust the platform operator to manage responsibly. The project adheres to Ghana's Data Protection Act of 2012 by collecting only data necessary for platform operation, providing clear disclosure of data usage in terms of service, and implementing reasonable security safeguards against unauthorized access.

7.2 Digital Divide and Exclusion

Despite efforts to reduce barriers through optional email, SMS-based OTP, and multimodal AI interaction, SmartAgro inevitably excludes farmers who lack smartphones, mobile data access, or basic digital literacy. This exclusion pattern risks replicating and potentially exacerbating existing inequalities, where better-resourced farmers with education and technology access gain additional market advantages while the most marginalized farmers remain dependent on exploitative traditional intermediaries. Addressing this digital divide requires complementary interventions beyond the platform itself, including subsidized smartphone distribution programs, community training workshops on digital marketplace usage, and potentially hybrid models where digitally connected farmers or local agents can assist neighbors in accessing the platform.

7.3 Economic Disruption and Intermediary Displacement

The explicit goal of removing intermediaries from agricultural supply chains has significant economic implications for existing market participants whose livelihoods depend on aggregation and distribution services. While intermediary markups are characterized as exploitative, these actors provide valuable functions including demand aggregation, transportation logistics, market information sharing, and flexible payment terms that enable farmers to sell immediately even when buyers are not yet identified. Simply eliminating intermediaries without replacing their functional contributions could create new coordination problems. A more nuanced approach might integrate existing intermediaries as logistics service providers or market facilitators who are compensated for specific value-adding services rather than capturing rents through information asymmetry. This reframes intermediaries as platform partners rather than displaced competitors.

7.4 Algorithmic Accountability in Agricultural Advice

The AI farming assistant provides agricultural advice that influences farmer decisions about planting timing, pest control methods, and resource allocation. Incorrect advice could lead to crop failures with serious economic consequences for farmers operating on thin margins. The system attempts to mitigate this risk through multiple mechanisms including sourcing knowledge from authoritative agricultural research institutions, labeling information with reliability scores based on source credibility, prompting the assistant to request clarification about context before providing specific recommendations, and storing system prompts in database configuration rather than code to enable rapid correction of discovered issues. However, the fundamental challenge remains that large language models can generate plausible but incorrect information, and farmers may lack the expertise to independently verify advice accuracy. This tension between accessibility and accountability requires ongoing monitoring of assistant outputs and potentially human expert review of high-stakes recommendations.

8. CONCLUSION AND FUTURE WORK

This paper presented SmartAgro, a digital marketplace platform engineered to address the technical barriers preventing efficient agricultural trade in Ghana. Through the implementation of a modular monolithic architecture, an integrated escrow system, and offline-capable PWA frontend, we have demonstrated that modern web technologies can be adapted to the constraints of developing economies.

Our technical evaluation answers the research question regarding the viability of this architecture. The system achieved 99.3% availability for critical infrastructure and sub-300ms latency for transactional flows which proves that the core logic is production-ready. However, the 42.3% overall availability during stress testing highlights the limitations of direct database querying for search operations in monolithic designs. The PWA audit score of 78 out of 100 and successful offline service worker installation confirms that the frontend can effectively function in intermittent connectivity environments.

Immediate future work will focus on architectural refactoring based on the load test results. Specifically, we aim to extract the search module into a dedicated microservice with caching layers to resolve the bottleneck. Following this technical stabilization, the research focus will shift to a longitudinal field study during the upcoming harvest season. This subsequent study will move beyond technical metrics to evaluate the economic hypothesis by quantifying the actual reduction in post-harvest losses and measuring the adoption rates among the target demographic of smallholder farmers.

REFERENCES

[1] Aker, J. C. 2011. Dial 'A' for agriculture: A review of information and communication technologies for agricultural extension in developing countries. Agricultural Economics 42, 6 (2011), 631-647.

[2] Anthropic. 2024. Claude 3 Model Family: Sonnet, Opus, and Haiku. Technical Report. Anthropic PBC, San Francisco, CA.

[3] Dittus, M., Quattrone, G., and Capra, L. 2016. Analysing Volunteer Engagement in Humanitarian Mapping: Building Contributor Communities at Large Scale. In Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing (CSCW '16). ACM, New York, NY, 108-118.

[4] FastAPI. 2024. FastAPI Framework Documentation. Retrieved December 2025 from https://fastapi.tiangolo.com

[5] Fowler, M. 2014. Microservices: A definition of this new architectural term. Retrieved December 2025 from https://martinfowler.com/articles/microservices.html

[6] Government of Ghana. 2012. Data Protection Act, 2012 (Act 843). Accra: Ghana Publishing Corporation.

[7] International Growth Centre. 2025. Market Coordination Failures in Ghanaian Agriculture. Policy Brief. International Growth Centre, London, UK.

[8] Jensen, R. 2007. The digital provide: Information (technology), market performance, and welfare in the South Indian fisheries sector. The Quarterly Journal of Economics 122, 3 (2007), 879-924.

[9] Ministry of Food and Agriculture Ghana. 2023. Agriculture in Ghana: Facts and Figures. Statistics Research and Information Directorate, Accra, Ghana.

[10] Newman, S. 2015. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Sebastopol, CA.

[11]OpenAI. 2024. GPT-4 Technical Report. OpenAI, San Francisco, CA.

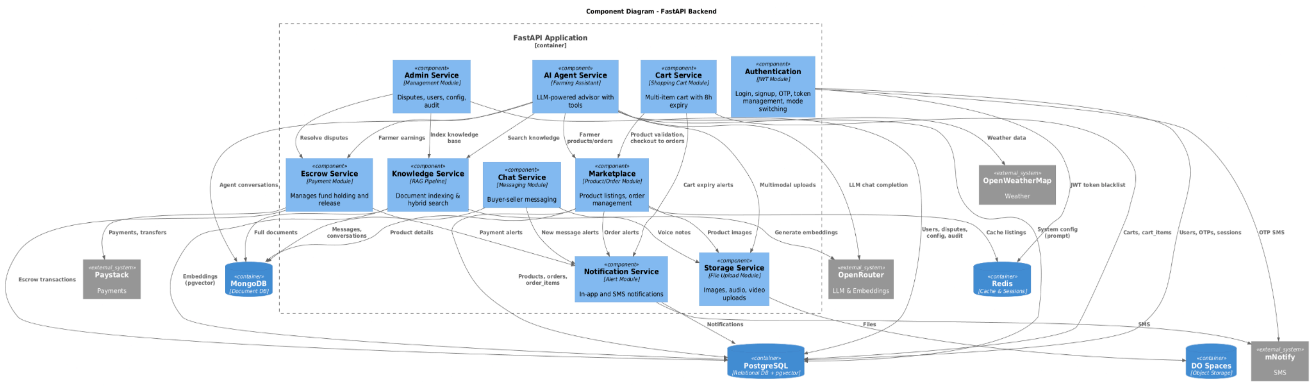
[12] Paystack. 2024. Transfers API Documentation. Retrieved December 2025 from https://paystack.com/docs/transfers/single-transfers

[13] React. 2024. React Documentation: Progressive Web Apps. Retrieved December 2025 from https://react.dev/learn

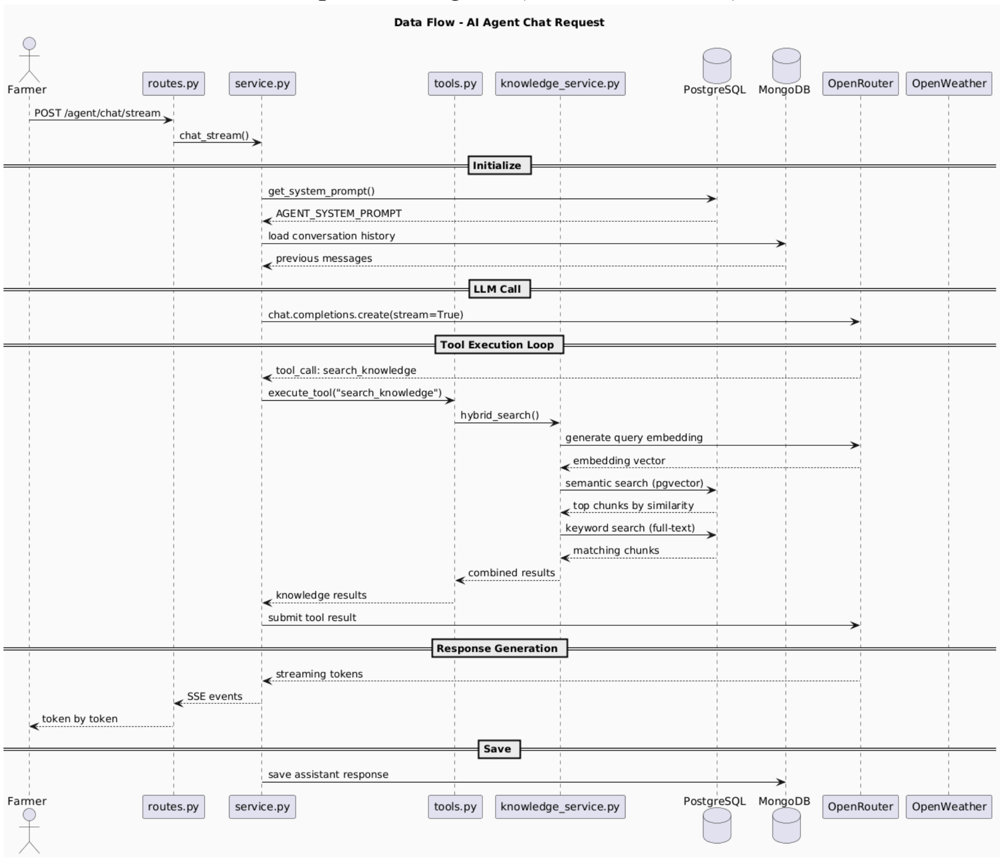
[14] Roduner, D. 2007. Mobile Payment Procedures: Enabling Mobile Commerce. ETH Zurich, Institute of Technology Management, Zurich, Switzerland.

[15] World Bank. 2024. Post-Harvest Loss Reduction in Sub-Saharan Africa: Challenges and Opportunities. Agriculture Global Practice, World Bank Group, Washington, DC.

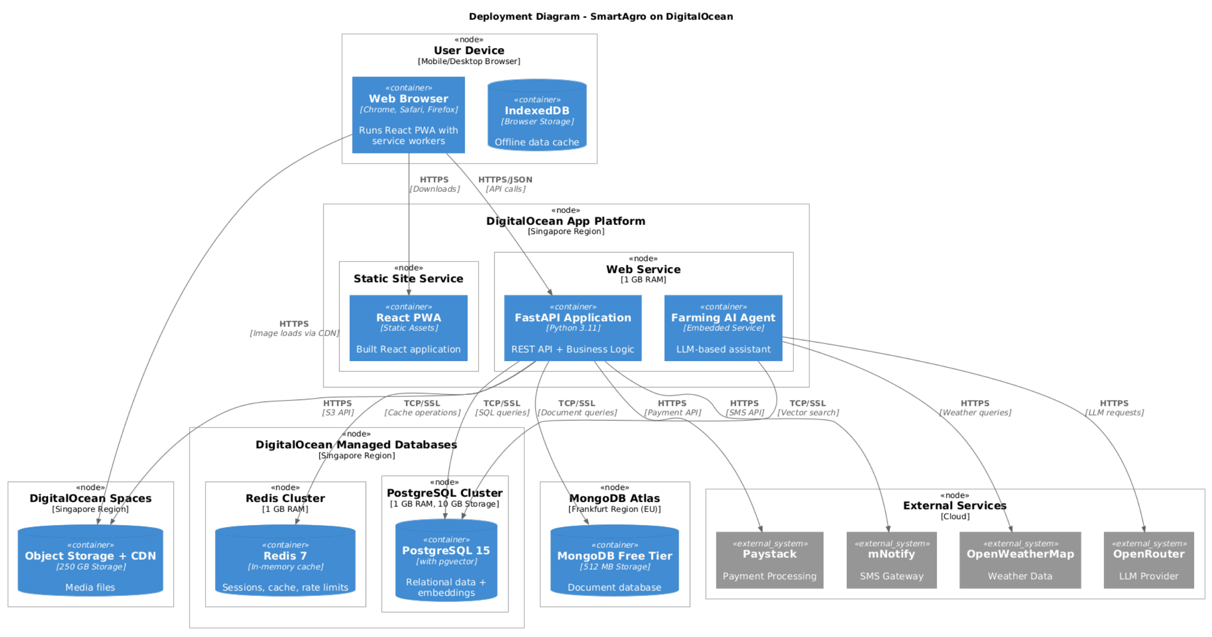
APPENDIX


Component Diagram - FastAPI Backend
FastAPI Application
[container]
«component» Admin Service [Management Module] Disputes, users, config, audit
«component» AI Agent Service [Farming Assistant] LLM-powered advisor with tools
«component» Cart Service [Shopping Cart Module] Multi-item cart with 8h expiry
«component» Authentication [JWT Module] Login, signup, OTP, token management, mode switching
«component» Escrow Service [Payment Module] Manages fund holding and release
«component» Knowledge Service [RAG Pipeline] Document indexing & hybrid search
«component» Chat Service [Messaging Module] Buyer-seller messaging
«component» Marketplace [Product/Order Module] Product listings, order management
«component» Notification Service [Alert Module] In-app and SMS notifications
«component» Storage Service [File Upload Module] Images, audio, video uploads
«external_system» Paystack Payments
«container» MongoDB [Document DB]
«external_system» OpenRouter LLM & Embeddings
«external_system» OpenWeatherMap Weather
«container» Redis [Cache & Sessions]
«container» PostgreSQL [Relational DB + pgvector]
«container» DO Spaces [Object Storage]
«external_system» mNotify SMS
Resolve disputes
Farmer earnings
Index knowledge base
Search knowledge
Farmer products/orders
Product validation, checkout to orders
Weather data
Agent conversations
Cart expiry alerts
Multimodal uploads
LLM chat completion
JWT token blacklist
Payments, transfers
Full documents
Messages, conversations
Product details
Payment alerts
New message alerts
Order alerts
Voice notes
Product images
Generate embeddings
Users, disputes, config, audit
Cache listings
System config (prompt)
Carts, cart_items
Users, OTPs, sessions
OTP SMS
Escrow transactions
Embeddings (pgvector)
Products, orders, order_items
Notifications
Files
SMS

Component Diagram (FastAPI Backend)

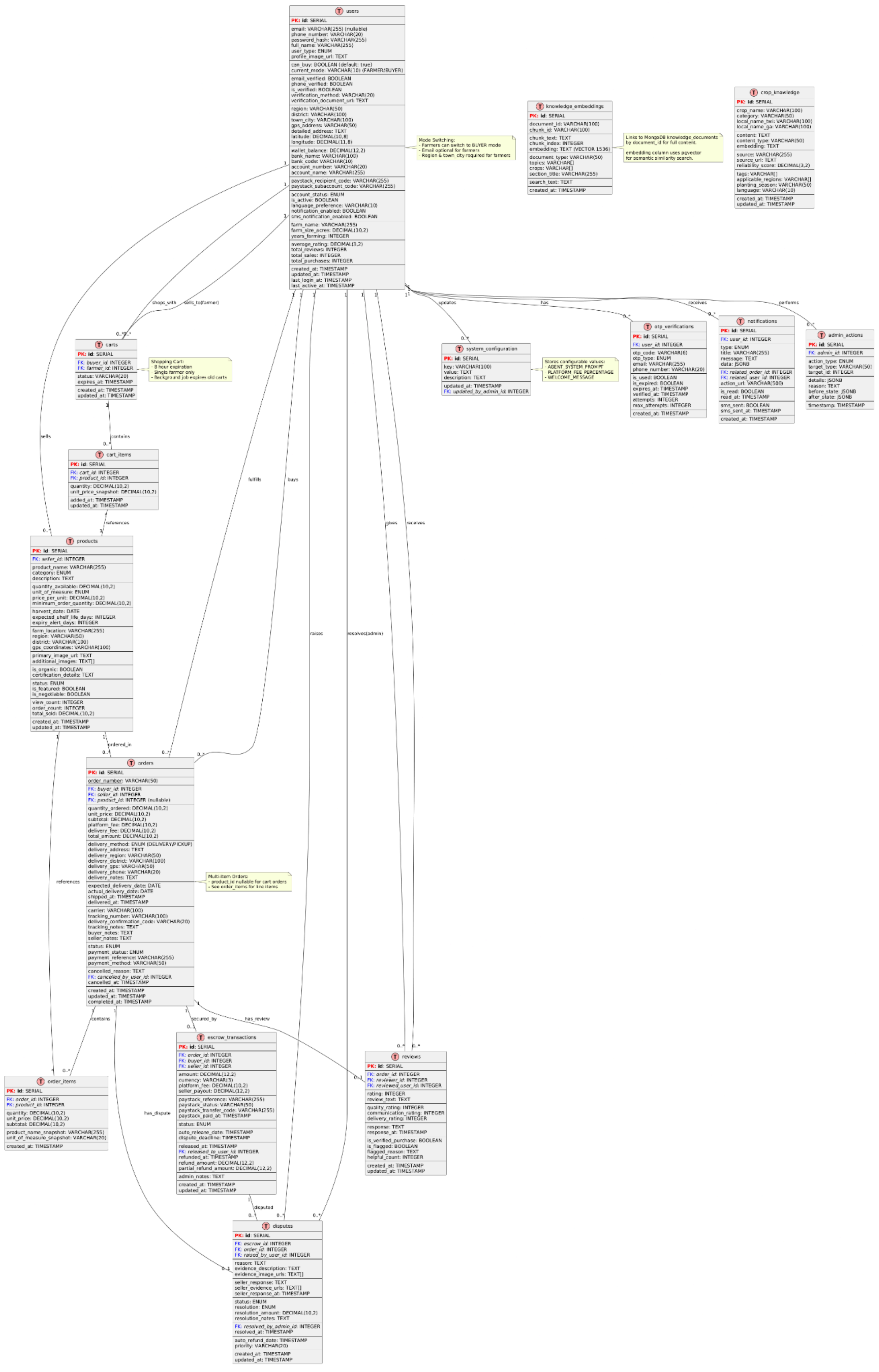


Agent Data Flow Diagram

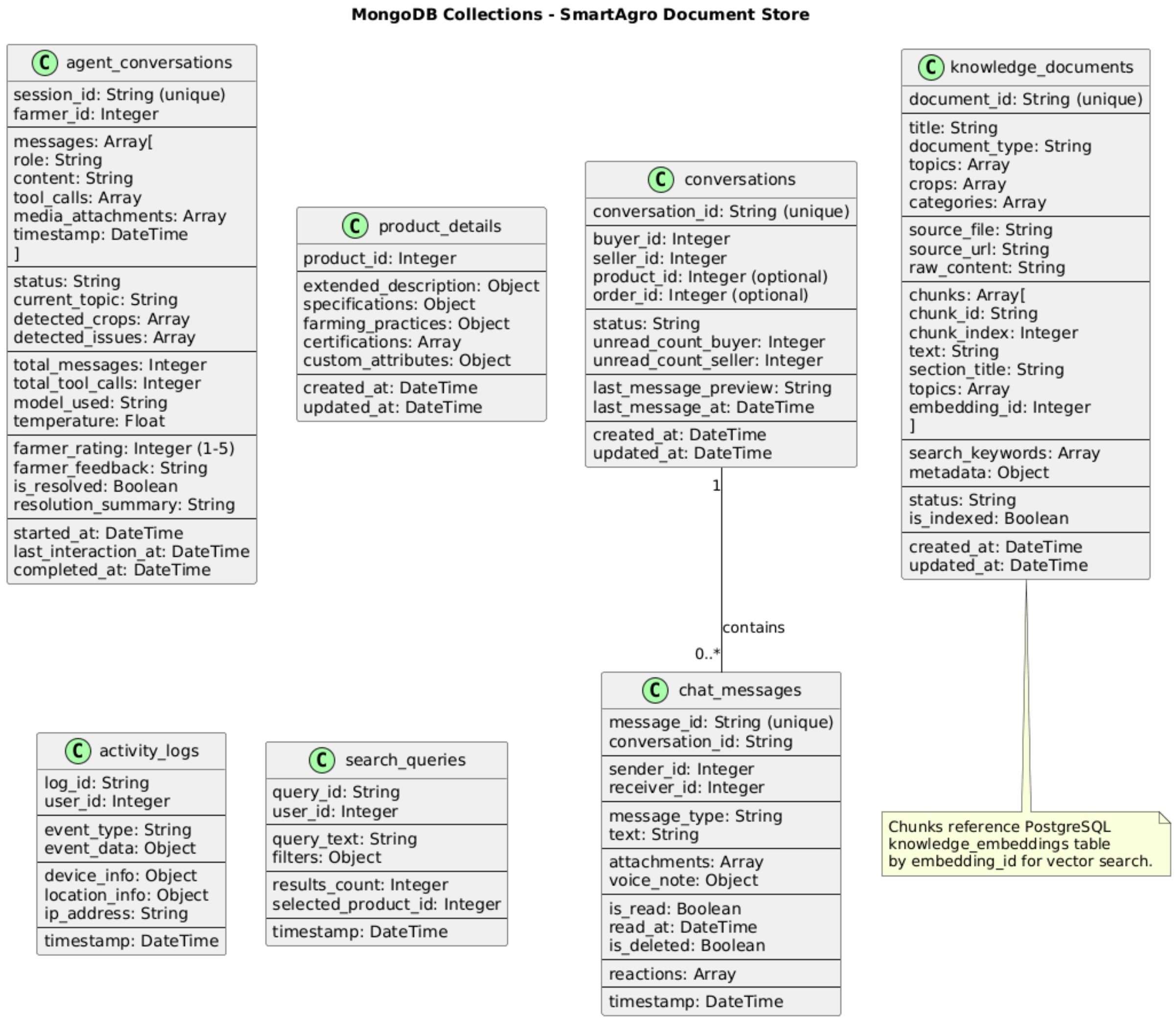


Deployment Diagram

Entity Relationship Diagram - SmartAgro PostgreSQL Schema



ERD Diagram for PostgreSQL



ERD Diagram for MongoDB

AI USE DECLARATION

This work made use of artificial intelligence assistance in the following ways:

Code Generation and Development Support: Claude AI (Anthropic) was used extensively through Claude Code to accelerate backend implementation, suggest architectural patterns, generate boilerplate code for database models and API endpoints, and provide debugging assistance. All generated code was thoroughly reviewed, tested, and modified by the authors to ensure correctness, security, and alignment with project requirements. The AI assistant served as a programming aid rather than an autonomous developer.

Agricultural Knowledge Compilation: Claude AI was consulted to compile agricultural best practices, crop calendars, pest management techniques, and farming advice for the AI farming assistant's knowledge base. All agricultural information provided by the AI was cross-referenced with official sources including Ministry of Food and Agriculture publications, CSIR research outputs, and IITA technical guides to ensure accuracy and local applicability. The AI served as an information aggregator that was subsequently validated against authoritative sources.

System Prompt Engineering: Iterative prompt design for the multimodal farming assistant was conducted with Claude AI's assistance to refine instruction clarity, improve response formatting, and enhance context-awareness. Multiple prompt iterations were tested against sample farmer queries to optimize advice quality and relevance.

Documentation and Paper Writing: Claude AI assisted in generating API documentation from code comments, structuring this research paper according to ACM format requirements, and refining technical explanations for clarity. The core architectural decisions, system design choices, research contributions, and evaluation methodologies are original work by the authors. All substantive claims and findings presented in this paper reflect the authors' analysis and judgment.

The authors take full responsibility for all content in this paper and the implemented system. AI tools were used as assistive technology to enhance productivity and code quality, not as co-authors or decision-makers. All critical engineering decisions, research methodology choices, evaluation interpretations, and ethical analyses represent the independent work and reasoning of the human authors.