K-Square Programme Onboarding Agent: Complete Project Specification for LLM Development

Alberto Espinosa
KSquare Group

1 August 2025

Abstract

The K-Square Programme Onboarding Agent is a sophisticated multi-agent system designed to streamline client onboarding processes through intelligent automation. This document provides comprehensive technical specifications, architectural details, implementation guidelines, and complete project understanding for Large Language Model (LLM) assisted development. The system comprises a React TypeScript frontend, FastAPI Python backend, and SQLite database, implementing five specialized AI agents that collaborate to analyze client data, generate insights, process meetings, and facilitate programme setup.

1 Introduction

The K-Square Programme Onboarding Agent addresses the challenge of manual onboarding by providing execution teams with rapid access to client profiles, domain knowledge, and actionable insights. This document serves as a complete technical reference for LLM-assisted development, providing all necessary context for understanding, modifying, and extending the system.

2 Key Capabilities

The system provides automated client profile analysis and completion, meeting transcription and intelligent analysis, actionable insights generation from client data, domain-specific knowledge base management, interactive programme setup guidance, and real-time dashboard analytics and monitoring.

3 System Architecture

The system follows a microservices-inspired architecture with clear separation of concerns. The Frontend Layer uses React TypeScript SPA with Vite build system. The API Layer implements FastAPI REST API with automatic OpenAPI documentation. The Agent Layer contains five specialized AI agents with distinct responsibilities. The Orchestration Layer manages workflow orchestrator for agent interactions. The Data Layer uses SQLite database with structured schema.

4 Agent Architecture

The Client Profile Agent analyzes incomplete client profiles, identifies missing information and data gaps, suggests profile completion strategies, and validates profile completeness and accuracy. The Meetings Agent processes audio meeting recordings, generates structured meeting summaries, extracts key decisions and action items, and identifies stakeholder concerns and requirements. The Actionable Insights Agent analyzes client data patterns and trends, generates strategic recommendations, identifies optimization opportunities,

and provides risk assessment and mitigation strategies. The Domain Knowledge Agent maintains industry-specific knowledge base, provides contextual information retrieval, supports semantic search capabilities, and manages knowledge categorization and tagging. The Programme Setup Agent guides interactive programme configuration, provides step-by-step setup assistance, validates configuration completeness, and generates setup documentation and summaries.

5 Data Flow

The system implements a structured data flow. Input Processing handles client data, meeting recordings, and user interactions. Agent Processing performs specialized analysis and processing by relevant agents. Orchestration coordinates agent interactions through the workflow orchestrator. Data Persistence stores results in SQLite database. API Exposure uses FastAPI endpoints to expose processed data. Frontend Consumption allows React components to consume and display data.

6 Technical Implementation

The backend is organized with agents directory containing actionable$_{i}nsights.py, client_{p}rofile.py, domain_{k}nowledge.py, meet$

7 Backend Components

The Main Application (main.py) handles FastAPI application initialization, CORS configuration for frontend integration, API endpoint definitions and routing, request/response models using Pydantic, and error handling and validation. The Database Manager (database/db$_{m}anager.py)managesSQLitedatabaseconnection, schemadefiniti$
$agentcommunicationprotocols, handlesworkflowstatemanagement, andprovideserrorhandlingandrecoverymechanisms.$

8 Frontend Implementation

The frontend follows modern React patterns with src directory containing components (charts, common, forms, layout), pages (ClientProfile.tsx, Dashboard.tsx, Insights.tsx, KnowledgeBase.tsx, Meetings.tsx, ProgrammeSetup.tsx, Settings.tsx), services (ApiService.ts), App.tsx, and main.tsx. The root contains package.json and vite.config.ts.

9 Frontend Technologies

The system uses React 18 as component-based UI framework, TypeScript for type-safe development, Vite as fast build tool and development server, React Query for server state management and caching, Framer Motion for animation and transitions, Tailwind CSS for utility-first styling, Chart.js for data visualization, and Lucide React as icon library.

10 Database Schema

The SQLite database implements five core tables. The client$_{p}rofilestablecontainsid(INTEGERPRIMARYKEY), name(T$

11 API Specification

The FastAPI backend exposes RESTful endpoints. Client Profile Endpoints include GET /api/client-profiles (retrieve all), GET /api/client-profiles/id (retrieve specific), POST /api/client-profiles (create new), PUT /api/client-profiles/id (update), DELETE /api/client-profiles/id (delete), and POST /api/client-profiles/id/analyze (trigger analysis). Meeting Endpoints include GET /api/meetings (retrieve all), GET /api/meetings/id (retrieve specific), POST /api/meetings/upload (upload recording), POST /api/meetings/id/analyze (trigger

analysis), and GET /api/meetings/summary (get statistics). Insights Endpoints include GET /api/insights (retrieve all), GET /api/insights/id (retrieve specific), POST /api/insights/generate (generate new), PUT /api/insights/id (update), and GET /api/insights/summary (get statistics). Knowledge Base Endpoints include GET /api/knowledge (retrieve items), POST /api/knowledge/search (search), GET /api/knowledge/categories (get categories), and POST /api/knowledge/id/bookmark (bookmark item). Programme Setup Endpoints include POST /api/setup/programme (handle interactions), GET /api/setup/session$_id(retrievesession), andPUT/api/setup$

12 User Interface Specification

The main dashboard provides comprehensive system overview with key metrics (total workflows, active clients, insights generated, meetings analyzed), workflow status charts, industry distribution, performance trends, recent activity feeds, quick actions, and system health monitoring. The client profiles view offers comprehensive client management with profile listing, completeness tracking, profile editor, stakeholder management, and analysis triggers. The meetings view provides meeting management and analysis interface with meeting upload, transcript display, summary generation, action items extraction, and key points highlighting. The insights view handles insights generation and management with insight cards, category filtering, priority sorting, status tracking, and recommendation display. The knowledge base view offers knowledge management and search with advanced search interface, category navigation, content display, bookmarking, and relevance scoring. The programme setup view provides interactive setup guidance with chat interface, progress tracking, step navigation, and configuration summary.

13 Development Guidelines

The project follows a modular architecture with clear separation of concerns. Backend code is organized into logical modules (agents, database, api, utils) with specific responsibilities, implementing agent patterns with consistent interfaces, repository patterns for database abstraction, service layers for business logic separation, and dependency injection through FastAPI. Frontend components follow React best practices with component composition, custom hooks for shared logic, centralized API interactions through ApiService, and React Query for server state management. Comprehensive error handling includes HTTP status codes and structured responses on the backend, React Query error boundaries and user-friendly messages on the frontend, logging for debugging and monitoring, and graceful degradation for agent failures. The testing strategy encompasses unit tests for individual components and agents, integration tests for API endpoints and user workflows, database migration tests, API mocking for isolated testing, and accessibility testing with above 80

14 Deployment and Operations

The development environment requires Python 3.8+, Node.js 16+, and SQLite, with backend dependencies managed through requirements.txt and frontend dependencies through package.json, using .env files for environment variable configuration across development, staging, and production environments. Production deployment options include containerization with Docker, cloud deployment on platforms like AWS, Azure, or GCP, reverse proxy configuration with Nginx, SSL certificate management, database backup and recovery procedures, and horizontal scaling support for increased load. Monitoring encompasses application performance monitoring, error tracking and alerting, system resource monitoring, user activity analytics, API endpoint performance metrics, structured logging with appropriate log levels, centralized log aggregation, and log retention policies. Regular maintenance includes database optimization and cleanup, dependency updates and security patches, backup verification and recovery testing, performance monitoring and optimization, and user feedback collection and analysis.

15 Extension and Customization

The system supports adding new AI agents by implementing the base agent interface, following established patterns for initialization, processing, and response generation, with agent registration through the workflow orchestrator configuration. Custom workflows can be created by combining existing agents or implementing new processing pipelines, with workflow configuration supporting conditional logic, parallel processing, error handling, and hooks for custom validation and transformation logic. The frontend supports theming and customization through CSS variables and component props, allowing new views to be added by creating React components and registering routes, with the component library providing consistent styling and behavior patterns. The system provides integration points for external services through API endpoints, webhook support, and plugin architecture, with database schema extension capabilities for additional tables and relationships, and configuration management supporting environment-specific customizations.

16 Security Considerations

Data protection measures include encryption at rest and in transit, secure file upload validation, input sanitization and validation, SQL injection prevention through parameterized queries, and regular security audits and updates. Access control implements authentication mechanisms, role-based authorization, session management, API rate limiting, and CORS configuration for cross-origin requests. Privacy compliance includes data retention policies, user consent management, data anonymization capabilities, audit logging for compliance, and GDPR/CCPA compliance considerations. Security best practices include regular dependency updates, secure coding practices, environment variable protection, error message sanitization, and security testing integration.

17 Performance Optimization

Backend optimization strategies include database query optimization with proper indexing, caching mechanisms for frequently accessed data, asynchronous processing for long-running tasks, connection pooling for database connections, and memory management for large datasets. Frontend optimization includes code splitting for reduced bundle sizes, lazy loading for components and routes, image optimization and compression, caching strategies for static assets, and performance monitoring and profiling.

18 Troubleshooting Guide

Common issues include database connection problems, API endpoint errors, frontend build failures, dependency conflicts, and environment configuration issues, with solutions involving checking database file permissions, verifying API endpoint availability, clearing node modules and reinstalling, updating dependencies, and validating environment variables. Performance issues may include slow database queries, high memory usage, slow API responses, and frontend rendering problems, with solutions involving optimizing database queries and indexes, implementing caching strategies, profiling and optimizing code, and monitoring resource usage. Development issues include hot reload problems, CORS errors, import/export errors, and TypeScript compilation errors, with solutions involving restarting development servers, configuring CORS settings, checking import paths and exports, and resolving TypeScript type errors. Deployment issues include build failures, environment variable problems, database migration errors, and service startup failures, with solutions involving checking build configurations, validating environment variables, running database migrations manually, and reviewing service logs.

19 Future Enhancements

Future enhancements include implementing user authentication with role-based access control, session management, and secure API endpoints, supporting multiple authentication providers and user management

capabilities. Advanced analytics capabilities will include predictive modeling, trend analysis, custom reporting, data visualization enhancements, and machine learning models for improved insights generation. Integration capabilities will expand to include external CRM systems, calendar applications, email platforms, third-party analytics tools, webhook support, and API extensions. Mobile applications for iOS and Android platforms will provide core functionality access, offline capabilities, and push notifications for important updates. Scalability improvements will implement horizontal scaling capabilities, microservices architecture, distributed caching, and load balancing for enterprise-level deployments.

20 Conclusion

This comprehensive specification provides complete technical documentation for the K-Square Programme Onboarding Agent. The modular architecture, clear separation of concerns, and detailed implementation guidelines enable efficient development, maintenance, and extension of the system.

The multi-agent approach provides flexibility and scalability, while the modern technology stack ensures maintainability and performance. This documentation serves as a complete reference for LLM-assisted development, providing all necessary context for understanding, modifying, and extending the system.