# **Wellness Companion AI - Complete Workflow**

## **System Integration Flow**

### **Phase 1: Application Initialization**

#### **Pathway A: Desktop Startup Flow**

User launches DMG/EXE → Health Check → Authentication → Dashboard

**Components Involved:**

* PyQt6 Interface triggers Core Orchestrator
* Core Orchestrator calls FastAPI health endpoints
* FastAPI validates system components (Ollama, Qdrant, Redis)
* **Integration Point**: Health check failure blocks dashboard access

#### **Pathway B: Authentication Flow**

Login Click → Google OAuth → JWT Generation → Keyring Storage

**Components Involved:**

* Desktop UI → FastAPI Auth endpoints → Google OAuth → AWS Cognito
* **Integration Point**: JWT token stored locally + FastAPI session management

### **Phase 2: Document Processing Pipeline**

#### **Pathway C: Document Ingestion Flow**

File Upload → Validation → Processing → Embedding → Storage

**Step-by-Step Process:**

1. **Desktop Layer**: File dialog selection
2. **Core Backend**: FastAPI receives chunked upload
3. **Data Layer**: Embedding pipeline activated
4. **AI/ML Layer**: Document processing begins

**Component Interactions:**

* **C1**: Desktop → FastAPI (/upload endpoint)
* **C2**: FastAPI → Embedding Pipeline (Data Layer)
* **C3**: Text Splitter checks "Embed+Store" condition
* **C4**: If NEW document → Sentence Transformers generate embeddings
* **C5**: Qdrant stores vectors + PostgreSQL stores metadata
* **C6**: Original file → S3 bucket with encryption
* **Integration Point**: Embedding pipeline interfaces with both vector DB and metadata DB

### **Phase 3: Query Processing System**

#### **Pathway D: Primary Query Flow**

User Query → Intent Analysis → Vector Search → LLM Response → UI Display

**Component Chain:**

1. **Desktop**: PyQt6 captures query
2. **Core Orchestrator**: Preprocesses and routes to FastAPI
3. **FastAPI**: Triggers LangChain orchestration
4. **AI/ML Layer**: Vector similarity search in Qdrant
5. **Confidence Check**: If score > 0.7 → Local LLM (Ollama)
6. **Response Synthesis**: Format and return to desktop

**Integration Points:**

* **D1**: Core Orchestrator ↔ FastAPI (bidirectional JSON)
* **D2**: FastAPI ↔ LangChain (RAG orchestration)
* **D3**: LangChain ↔ Qdrant (vector retrieval)
* **D4**: LangChain ↔ Ollama (response generation)

#### **Pathway E: Fallback Query Flow**

Low Confidence (≤0.7) → Web Search → Enhanced Response → UI Display

**Fallback Trigger Process:**

1. **AI/ML Layer**: Confidence score evaluation
2. **Response Synthesizer**: Triggers Tavily API
3. **Web Search**: Query optimization and results
4. **Enhanced Processing**: Local context + Web results → LLM
5. **Source Attribution**: Combined response with citations

**Integration Points:**

* **E1**: Response Synthesizer ↔ Tavily API
* **E2**: Web results merge with local context in LangChain
* **E3**: Enhanced response follows same return pathway as D4-D1

### **Phase 4: Real-Time Communication**

#### **Pathway F: System Tray Integration**

Background Service → WebSocket Connection → Real-Time Updates

**Component Flow:**

* **Desktop Layer**: System tray service runs independently
* **Core Backend**: WebSocket endpoint for real-time communication
* **Integration Point**: Shares JWT authentication with main application

#### **Pathway G: Floating Overlay System**

Hotkey/Trigger → Overlay Display → Quick Query → Instant Response

**Mini-App Workflow:**

* **Independent UI**: Minimal PyQt6 overlay
* **Shared Backend**: Same FastAPI endpoints as main app
* **Integration Point**: Uses cached embeddings and models

### **Phase 5: Data Synchronization**

#### **Pathway H: Cloud Sync Flow**

Local Changes → Change Detection → AWS Sync → Multi-Device Update

**Sync Components:**

* **Data Layer**: PostgreSQL change logs
* **Infrastructure**: AWS RDS synchronization
* **Integration Points**:
  + **H1**: Local PostgreSQL ↔ AWS RDS
  + **H2**: Local S3 cache ↔ AWS S3 bucket
  + **H3**: Qdrant local ↔ Qdrant cloud instance

## **Critical Integration Points**

### **I1: Core Orchestrator Hub**

**Connects:** Desktop Layer ↔ Core Backend Layer **Function:** Request routing, response formatting, error handling **Dependencies:** All UI interactions pass through this component

### **I2: FastAPI Central Hub**

**Connects:** Core Backend ↔ AI/ML ↔ Data Layers **Function:** API gateway, authentication, rate limiting **Dependencies:** Health checks, LangChain triggering, database operations

### **I3: LangChain Orchestration**

**Connects:** AI/ML Layer components **Function:** RAG pipeline management, model coordination **Dependencies:** Vector search, LLM inference, response synthesis

### **I4: Data Layer Convergence**

**Connects:** Qdrant + PostgreSQL + S3 + Redis **Function:** Unified data operations **Dependencies:** Embedding pipeline, metadata management, caching

### **I5: Infrastructure Gateway**

**Connects:** All layers ↔ AWS Services **Function:** Cloud deployment, scaling, monitoring **Dependencies:** Docker containers, NGINX routing, SSL termination

## **Error Handling & Fallback Chains**

### **Error Chain A: Component Failure**

Health Check Fail → Graceful Degradation → User Notification

### **Error Chain B: LLM Unavailable**

Ollama Down → Web Search Only → Cached Responses → Error State

### **Error Chain C: Network Issues**

Cloud Sync Fail → Local Mode → Background Retry → Success Recovery

## **Development Workflow Phases**

### **Phase 1: Core Foundation**

* Embedding pipeline + Vector storage
* Basic FastAPI endpoints
* Local LLM integration

### **Phase 2: RAG Implementation**

* Document ingestion workflow (Pathway C)
* Query processing (Pathway D)
* Confidence-based fallback (Pathway E)

### **Phase 3: Authentication & Security**

* OAuth integration (Pathway B)
* JWT management
* Rate limiting implementation

### **Phase 4: Desktop Interface**

* PyQt6 main application
* System tray integration (Pathway F)
* Floating overlay (Pathway G)

### **Phase 5: Cloud Integration**

* AWS deployment (Infrastructure Layer)
* Cloud sync implementation (Pathway H)
* Monitoring and logging

### **Phase 6: Production Optimization**

* Cythonization of core components
* Docker containerization
* CI/CD pipeline setup

## **Key Decision Points**

| **Component** | **Decision Trigger** | **Action Taken** |
| --- | --- | --- |
| **Embed+Store** | Document exists check | Skip processing OR Create embeddings |
| **Confidence Score** | Vector similarity < 0.7 | Trigger web search fallback |
| **Health Check** | Component unavailable | Block feature OR Graceful degradation |
| **Authentication** | Token expired | Refresh OR Re-authenticate |
| **Cloud Sync** | Network available | Sync data OR Queue for later |