# **Wellness Companion AI - 100 Task Development Sequence**

## **🎯 TASK-BY-TASK DEVELOPMENT SEQUENCE**

**Complete all tasks in order. Do not skip or parallelize.**

## **PHASE 1: FOUNDATION & DOCKERIZATION (Tasks 1-15)**

### **TASK 1: Project Root Structure**

**Folder: wellness-companion-ai/ (root) What: Create complete 6-layer folder structure as per directory specification**

### **TASK 2: Docker Compose Setup**

**Folder: wellness-companion-ai/ What: Create docker-compose.yml with all services (postgres, redis, qdrant, nginx placeholders)**

### **TASK 3: Environment Configuration**

**Folder: wellness-companion-ai/ What: Create .env.example with all required environment variables, setup .gitignore**

### **TASK 4: PostgreSQL Docker Service**

**Folder: services/infrastructure/docker/postgres/ What: Create PostgreSQL Dockerfile, init scripts, and docker-compose integration**

### **TASK 5: Redis Docker Service**

**Folder: services/infrastructure/docker/redis/ What: Create Redis Dockerfile, configuration, and docker-compose integration**

### **TASK 6: Qdrant Docker Service**

**Folder: services/data-layer/ What: Add Qdrant service to docker-compose with persistent volumes**

### **TASK 7: Ollama Docker Service**

**Folder: services/aiml-orchestration/ What: Create Ollama Docker container setup with model volume mounting**

### **TASK 8: AI/ML Service Container**

**Folder: services/aiml-orchestration/ What: Create Dockerfile for AI/ML service with Python dependencies**

### **TASK 9: Core Backend Container**

**Folder: services/core-backend/ What: Create Dockerfile for FastAPI backend service**

### **TASK 10: Data Layer Container**

**Folder: services/data-layer/ What: Create Dockerfile for data processing service**

### **TASK 11: Test Docker Environment**

**Folder: wellness-companion-ai/ What: Run docker-compose up and verify all services start without errors**

### **TASK 12: Service Health Checks**

**Folder: services/\*/ What: Add health check endpoints to all Docker services**

### **TASK 13: Inter-Service Networking**

**Folder: wellness-companion-ai/ What: Configure Docker networks for service-to-service communication**

### **TASK 14: Volume Management**

**Folder: wellness-companion-ai/ What: Setup Docker volumes for data persistence (postgres, qdrant, models)**

### **TASK 15: Environment Variables Setup**

**Folder: wellness-companion-ai/ What: Configure environment variables for all services in docker-compose**

## **PHASE 2: CORE RAG FOUNDATION (Tasks 16-30)**

### **TASK 16: AI/ML Service Basic Structure**

**Folder: services/aiml-orchestration/src/ What: Create Python module structure with init.py files**

### **TASK 17: Text Processing Pipeline**

**Folder: services/aiml-orchestration/src/pipelines/ What: Build PDF text extraction using PyPDF2/pymupdf**

### **TASK 18: Text Chunking Algorithm**

**Folder: services/aiml-orchestration/src/pipelines/ What: Implement text splitter (512 tokens, 50 overlap) using LangChain**

### **TASK 19: Embedding Model Setup**

**Folder: services/aiml-orchestration/src/embeddings/ What: Initialize sentence-transformers with all-MiniLM-L6-v2 model**

### **TASK 20: Qdrant Client Integration**

**Folder: services/aiml-orchestration/src/search/ What: Create Qdrant client wrapper with connection management**

### **TASK 21: Embedding Generation**

**Folder: services/aiml-orchestration/src/embeddings/ What: Build embedding generation service with batch processing**

### **TASK 22: Vector Storage Operations**

**Folder: services/aiml-orchestration/src/search/ What: Implement vector CRUD operations in Qdrant**

### **TASK 23: Document Ingestion Pipeline**

**Folder: services/aiml-orchestration/src/pipelines/ What: Connect: PDF → chunks → embeddings → Qdrant storage**

### **TASK 24: Ollama Client Setup**

**Folder: services/aiml-orchestration/src/llm/ What: Create Ollama API client with model management**

### **TASK 25: Vector Search Implementation**

**Folder: services/aiml-orchestration/src/search/ What: Build cosine similarity search with configurable threshold**

### **TASK 26: Context Retrieval**

**Folder: services/aiml-orchestration/src/orchestrators/ What: Combine search results into coherent context for LLM**

### **TASK 27: Response Generation**

**Folder: services/aiml-orchestration/src/llm/ What: Send context + query to Ollama, handle response streaming**

### **TASK 28: RAG Orchestrator**

**Folder: services/aiml-orchestration/src/orchestrators/ What: Build main RAG pipeline: query → search → retrieve → generate**

### **TASK 29: Confidence Scoring**

**Folder: services/aiml-orchestration/src/orchestrators/ What: Calculate confidence based on similarity scores and result count**

### **TASK 30: Test Core RAG Pipeline**

**Folder: services/aiml-orchestration/tests/ What: Test complete RAG flow with sample PDF document**

## **PHASE 3: WEB SEARCH INTEGRATION (Tasks 31-35)**

### **TASK 31: Tavily API Client**

**Folder: services/aiml-orchestration/src/search/ What: Create Tavily API client with error handling and rate limiting**

### **TASK 32: Web Search Processing**

**Folder: services/aiml-orchestration/src/search/ What: Process Tavily results into structured format**

### **TASK 33: Hybrid Search Logic**

**Folder: services/aiml-orchestration/src/orchestrators/ What: Implement fallback: if confidence < 0.7 → trigger web search**

### **TASK 34: Result Synthesis**

**Folder: services/aiml-orchestration/src/orchestrators/ What: Combine local + web results into enhanced response**

### **TASK 35: Test Hybrid Search**

**Folder: services/aiml-orchestration/tests/ What: Test low-confidence queries trigger web search correctly**

## **PHASE 4: CORE BACKEND APIs (Tasks 36-50)**

### **TASK 36: FastAPI Application Setup**

**Folder: services/core-backend/src/api/ What: Create FastAPI app with CORS, middleware, and basic structure**

### **TASK 37: AI/ML Service Client**

**Folder: services/core-backend/src/integrations/ What: HTTP client for communicating with AI/ML service**

### **TASK 38: Health Check API**

**Folder: services/core-backend/src/api/endpoints/system/ What: GET /health endpoint checking all service dependencies**

### **TASK 39: RAG Query API**

**Folder: services/core-backend/src/api/endpoints/search/ What: POST /api/search/hybrid endpoint for RAG queries**

### **TASK 40: Document Upload API**

**Folder: services/core-backend/src/api/endpoints/documents/ What: POST /api/documents/upload with file validation and processing**

### **TASK 41: Document List API**

**Folder: services/core-backend/src/api/endpoints/documents/ What: GET /api/documents/list with pagination and filtering**

### **TASK 42: Document Details API**

**Folder: services/core-backend/src/api/endpoints/documents/ What: GET /api/documents/{id} with metadata and status**

### **TASK 43: Document Delete API**

**Folder: services/core-backend/src/api/endpoints/documents/ What: DELETE /api/documents/{id} with vector cleanup**

### **TASK 44: Semantic Search API**

**Folder: services/core-backend/src/api/endpoints/search/ What: POST /api/search/semantic for local-only vector search**

### **TASK 45: Web Search API**

**Folder: services/core-backend/src/api/endpoints/search/ What: POST /api/search/web for web-only search**

### **TASK 46: Request Validation**

**Folder: services/core-backend/src/models/ What: Pydantic models for all API request/response schemas**

### **TASK 47: Error Handling Middleware**

**Folder: services/core-backend/src/api/middleware/ What: Global error handling with proper HTTP status codes**

### **TASK 48: API Documentation**

**Folder: services/core-backend/ What: Configure FastAPI automatic OpenAPI/Swagger documentation**

### **TASK 49: CORS Configuration**

**Folder: services/core-backend/src/core/ What: Proper CORS setup for desktop app communication**

### **TASK 50: Test All Backend APIs**

**Folder: services/core-backend/tests/ What: Integration tests for all API endpoints**

## **PHASE 5: DATA PERSISTENCE (Tasks 51-65)**

### **TASK 51: Data Layer Service Setup**

**Folder: services/data-layer/src/ What: Create FastAPI service for data operations**

### **TASK 52: PostgreSQL Models**

**Folder: services/data-layer/src/relational\_db/models/ What: SQLAlchemy models for users, documents, conversations**

### **TASK 53: Database Connection**

**Folder: services/data-layer/src/relational\_db/connection/ What: Database connection pool and session management**

### **TASK 54: Document Repository**

**Folder: services/data-layer/src/relational\_db/repositories/ What: CRUD operations for document metadata**

### **TASK 55: User Repository**

**Folder: services/data-layer/src/relational\_db/repositories/ What: CRUD operations for user profiles**

### **TASK 56: Database Migrations**

**Folder: services/data-layer/src/relational\_db/migrations/ What: Alembic setup and initial schema migrations**

### **TASK 57: File Storage Handler**

**Folder: services/data-layer/src/file\_storage/ What: Local file storage with organized directory structure**

### **TASK 58: Embedding Pipeline Service**

**Folder: services/data-layer/src/embedding\_pipeline/ What: FastAPI endpoints for document processing pipeline**

### **TASK 59: Embed+Store Checker**

**Folder: services/data-layer/src/embedding\_pipeline/ What: Check if document already exists to avoid reprocessing**

### **TASK 60: Document Processing API**

**Folder: services/data-layer/src/api/endpoints/ What: Process document → store metadata → trigger embedding**

### **TASK 61: Vector Operations API**

**Folder: services/data-layer/src/api/endpoints/ What: CRUD operations for vector data in Qdrant**

### **TASK 62: Redis Cache Integration**

**Folder: services/data-layer/src/cache/ What: Redis client for caching search results and embeddings**

### **TASK 63: Data Layer Client**

**Folder: services/core-backend/src/integrations/ What: HTTP client for core backend → data layer communication**

### **TASK 64: Update Backend APIs**

**Folder: services/core-backend/src/api/endpoints/ What: Modify APIs to use data layer service instead of direct DB**

### **TASK 65: Test Data Persistence**

**Folder: services/data-layer/tests/ What: Test document upload, storage, and retrieval persistence**

## **PHASE 6: AUTHENTICATION SYSTEM (Tasks 66-75)**

### **TASK 66: JWT Infrastructure**

**Folder: services/core-backend/src/services/auth\_service/ What: JWT token generation, validation, and refresh logic**

### **TASK 67: Google OAuth Setup**

**Folder: services/core-backend/src/integrations/oauth\_providers/ What: Google OAuth client configuration and flow handling**

### **TASK 68: User Session Management**

**Folder: services/core-backend/src/services/auth\_service/ What: Session creation, validation, and cleanup using Redis**

### **TASK 69: Authentication Middleware**

**Folder: services/core-backend/src/api/middleware/ What: JWT validation middleware for protected routes**

### **TASK 70: Auth API Endpoints**

**Folder: services/core-backend/src/api/endpoints/auth/ What: Login, callback, refresh, logout, profile endpoints**

### **TASK 71: User Profile Management**

**Folder: services/core-backend/src/api/endpoints/auth/ What: User profile CRUD operations and preferences**

### **TASK 72: Multi-user Document Isolation**

**Folder: services/data-layer/src/relational\_db/ What: Ensure users only access their own documents**

### **TASK 73: Rate Limiting**

**Folder: services/core-backend/src/api/middleware/ What: Redis-based rate limiting per user and per IP**

### **TASK 74: Security Headers**

**Folder: services/core-backend/src/api/middleware/ What: Security middleware with proper headers (HSTS, CSP, etc.)**

### **TASK 75: Test Authentication System**

**Folder: services/core-backend/tests/ What: Test complete auth flow and multi-user isolation**

## **PHASE 7: DESKTOP APPLICATION (Tasks 76-88)**

### **TASK 76: PyQt6 Application Structure**

**Folder: services/desktop-app/src/ What: Basic PyQt6 app with main window and resource management**

### **TASK 77: Core Orchestrator**

**Folder: services/desktop-app/src/core/orchestrator/ What: HTTP client for backend communication with error handling**

### **TASK 78: Authentication Manager**

**Folder: services/desktop-app/src/core/auth\_manager/ What: OAuth flow handling and secure token storage using keyring**

### **TASK 79: Login Interface**

**Folder: services/desktop-app/src/ui/dialogs/ What: Login dialog with embedded browser for OAuth**

### **TASK 80: Main Window Layout**

**Folder: services/desktop-app/src/ui/main\_window/ What: Main application window with menu bar and status bar**

### **TASK 81: Document Upload Interface**

**Folder: services/desktop-app/src/ui/main\_window/ What: Drag-and-drop file upload with progress indication**

### **TASK 82: Chat Interface**

**Folder: services/desktop-app/src/ui/main\_window/ What: Chat area with message history and input field**

### **TASK 83: Document Manager**

**Folder: services/desktop-app/src/ui/main\_window/ What: Document list with preview and management options**

### **TASK 84: Settings Panel**

**Folder: services/desktop-app/src/ui/dialogs/ What: Settings dialog for model selection and preferences**

### **TASK 85: Theme Engine**

**Folder: services/desktop-app/src/ui/themes/ What: QSS-based theming system with dark/light themes**

### **TASK 86: System Tray Integration**

**Folder: services/desktop-app/src/ui/system\_tray/ What: System tray icon with basic menu and notifications**

### **TASK 87: Floating Overlay**

**Folder: services/desktop-app/src/ui/floating\_overlay/ What: Minimal query overlay with hotkey activation**

### **TASK 88: Test Desktop Application**

**Folder: services/desktop-app/tests/ What: End-to-end test of complete desktop workflow**

## **PHASE 8: REAL-TIME FEATURES (Tasks 89-93)**

### **TASK 89: WebSocket Server**

**Folder: services/core-backend/src/api/websocket/ What: FastAPI WebSocket endpoints for real-time communication**

### **TASK 90: WebSocket Client**

**Folder: services/desktop-app/src/handlers/ What: Desktop WebSocket client for real-time updates**

### **TASK 91: Real-time Chat**

**Folder: services/desktop-app/src/ui/main\_window/ What: Streaming chat responses via WebSocket**

### **TASK 92: Progress Notifications**

**Folder: services/desktop-app/src/handlers/ What: Real-time document processing progress updates**

### **TASK 93: Test Real-time Features**

**Folder: services/desktop-app/tests/ What: Test WebSocket communication and real-time updates**

## **PHASE 9: PRODUCTION OPTIMIZATION (Tasks 94-100)**

### **TASK 94: Cython Core Modules**

**Folder: services/aiml-orchestration/src/utils/cython\_core/ What: Convert text processing and vector operations to Cython**

### **TASK 95: Application Packaging**

**Folder: services/desktop-app/ What: Briefcase configuration for cross-platform builds**

### **TASK 96: Production Logging**

**Folder: services/cicd-logging/logging/ What: Structured logging across all services with centralization**

### **TASK 97: Error Monitoring**

**Folder: services/cicd-logging/logging/sentry/ What: Sentry integration for error tracking and performance monitoring**

### **TASK 98: Performance Monitoring**

**Folder: services/cicd-logging/logging/prometheus/ What: Prometheus metrics collection and Grafana dashboards**

### **TASK 99: CI/CD Pipeline**

**Folder: services/cicd-logging/.github/ What: GitHub Actions for automated testing and deployment**

### **TASK 100: Production Deployment**

**Folder: services/infrastructure/ What: Complete production deployment with monitoring and alerting**

## **🚨 CRITICAL RULES**

1. **Complete each task 100% before moving to next**
2. **Test every task within Docker environment**
3. **Never skip tasks or work on multiple tasks simultaneously**
4. **All development happens inside containers**
5. **Document each completed task and Docker configurations**

## **📊 PROGRESS TRACKING**

* **Tasks 1-15: Docker foundation (3-5 days)**
* **Tasks 16-30: Core RAG in containers (7-10 days)**
* **Tasks 31-35: Web search integration (2-3 days)**
* **Tasks 36-50: Backend APIs (7-10 days)**
* **Tasks 51-65: Data persistence (7-10 days)**
* **Tasks 66-75: Authentication (5-7 days)**
* **Tasks 76-88: Desktop application (10-15 days)**
* **Tasks 89-93: Real-time features (3-5 days)**
* **Tasks 94-100: Production readiness (5-7 days)**

**Total estimated time: 49-72 days**