# **Veda - Wellness Companion AI: Complete Technical Documentation**

## **Project Overview**

**WellnessCompanionAI** is a privacy-first desktop AI application designed for local document processing with semantic search capabilities, web fallback functionality, and optional cloud synchronization. The application enables users to upload personal documents, perform intelligent searches across their knowledge base, and receive AI-powered responses while maintaining data privacy through local processing.

### **Core Architecture Philosophy**

* **6-Layer Architecture**: Modular design with clear separation of concerns
* **Privacy-First**: Local processing by default with optional cloud enhancement
* **Offline-Capable**: Full functionality without internet connectivity
* **Cross-Platform**: Desktop application for Windows, macOS, and Linux

## **Technology Stack**

### **Frontend Technologies**

| **Component** | **Technology** | **Purpose** |
| --- | --- | --- |
| **Desktop Framework** | PyQt6 | Cross-platform GUI development |
| **Styling Engine** | QSS + QDarkStyle | Modern UI theming and styling |
| **State Management** | Python Classes | Application state coordination |
| **Local Storage** | SQLite | Client-side data persistence |
| **Packaging** | Briefcase + Code Signing | Cross-platform distribution |

### **Backend Technologies**

| **Component** | **Technology** | **Purpose** |
| --- | --- | --- |
| **Web Framework** | FastAPI | RESTful API server |
| **ASGI Server** | Uvicorn | High-performance async serving |
| **Task Queue** | Celery + Redis | Background job processing |
| **Caching Layer** | Redis | Performance optimization |
| **Authentication** | Google OAuth + AWS Cognito | User identity management |

### **AI/ML Technologies**

| **Component** | **Technology** | **Purpose** |
| --- | --- | --- |
| **LLM Framework** | OLLAMA | Local language model execution |
| **RAG Framework** | LangChain/LlamaIndex | Retrieval-augmented generation |
| **Vector Database** | QDRANT | Semantic similarity search |
| **Embeddings** | Sentence Transformers | Text vectorization |
| **Web Search** | Tavily API | External knowledge retrieval |

### **Infrastructure Technologies**

| **Component** | **Technology** | **Purpose** |
| --- | --- | --- |
| **Cloud Provider** | AWS | Scalable cloud infrastructure |
| **Database** | PostgreSQL (RDS) | Structured data storage |
| **File Storage** | S3 | Document and media storage |
| **Containerization** | Docker + ECS | Deployment consistency |
| **API Gateway** | AWS API Gateway + NGINX | Traffic management |
| **Monitoring** | Sentry + CloudWatch | Error tracking and metrics |

## **6-Layer Architecture Deep Dive**

### **Layer 1: Desktop Layer**

**Purpose**: User interface and client-side orchestration

**Key Components**:

* **PyQt6 Interface**: Main application window, dialogs, and widgets
* **Core Orchestrator**: Frontend-backend communication hub
* **Theme Engine**: QSS-based styling system
* **System Tray**: Background operation and quick access

**Critical Functions**:

# Core Orchestrator Functions

def initialize\_backend\_connection() # [Authentication] [desktop\_layer/core\_orchestrator/] [Layer 1]

def handle\_document\_upload() # [File Management] [desktop\_layer/document\_interface/] [Layer 1]

def process\_user\_query() # [Chat Interface] [desktop\_layer/chat\_interface/] [Layer 1]

def update\_system\_status() # [Health Check] [desktop\_layer/health\_check/] [Layer 1]

### **Layer 2: Core Backend Layer**

**Purpose**: API gateway, authentication, and request routing

**Key Components**:

* **FastAPI Gateway**: RESTful API endpoints
* **Authentication System**: OAuth and JWT management
* **Rate Limiting**: Redis-based request throttling
* **WebSocket Handlers**: Real-time communication

**Critical Functions**:

# API Gateway Functions

def authenticate\_user() # [Authentication] [core\_backend\_layer/authentication/] [Layer 2]

def validate\_jwt\_token() # [Security] [core\_backend\_layer/authentication/] [Layer 2]

def route\_document\_upload() # [Document API] [core\_backend\_layer/document\_api/] [Layer 2]

def handle\_search\_request() # [Search API] [core\_backend\_layer/search\_api/] [Layer 2]

### **Layer 3: AI/ML Orchestration Layer**

**Purpose**: RAG pipeline coordination and AI model management

**Key Components**:

* **RAG Orchestrator**: Retrieval-augmented generation pipeline
* **Vector Search**: QDRANT similarity search logic
* **OLLAMA Integration**: Local LLM management
* **Response Synthesizer**: AI response generation

**Critical Functions**:

# RAG Orchestration Functions

def orchestrate\_rag\_pipeline() # [RAG Coordination] [aiml\_orchestration\_layer/rag\_orchestrator/] [Layer 3]

def perform\_vector\_search() # [Similarity Search] [aiml\_orchestration\_layer/vector\_search/] [Layer 3]

def generate\_llm\_response() # [Response Generation] [aiml\_orchestration\_layer/ollama\_integration/] [Layer 3]

def calculate\_confidence\_score() # [Quality Assessment] [aiml\_orchestration\_layer/confidence\_scoring/] [Layer 3]

### **Layer 4: Data Layer**

**Purpose**: Data storage, processing, and management

**Key Components**:

* **Embedding Pipeline**: Document processing and vectorization
* **Vector Databases**: QDRANT for similarity search
* **Relational Database**: PostgreSQL for structured data
* **File Storage**: S3 and local file management

**Critical Functions**:

# Data Processing Functions

def process\_document() # [Document Processing] [data\_layer/embedding\_pipeline/document\_processors/] [Layer 4]

def generate\_embeddings() # [Vectorization] [data\_layer/embedding\_pipeline/embedding\_generators/] [Layer 4]

def store\_vectors() # [Vector Storage] [data\_layer/vector\_databases/qdrant\_handlers/] [Layer 4]

def manage\_file\_storage() # [File Management] [data\_layer/file\_storage/s3\_handlers/] [Layer 4]

### **Layer 5: Infrastructure Deployment Layer**

**Purpose**: Cloud infrastructure and deployment management

**Key Components**:

* **AWS Infrastructure**: Cloud resource management
* **Docker Containers**: Application containerization
* **Security Management**: SSL, certificates, and access control
* **Monitoring Systems**: CloudWatch and Sentry integration

**Critical Functions**:

# Infrastructure Functions

def deploy\_to\_aws() # [Cloud Deployment] [infrastructure\_deployment\_layer/aws\_infrastructure/] [Layer 5]

def manage\_containers() # [Containerization] [infrastructure\_deployment\_layer/docker\_configs/] [Layer 5]

def monitor\_system\_health() # [Monitoring] [infrastructure\_deployment\_layer/monitoring/] [Layer 5]

def handle\_ssl\_certificates() # [Security] [infrastructure\_deployment\_layer/security/] [Layer 5]

### **Layer 6: CI/CD Pipeline and Logging**

**Purpose**: Automation, packaging, and monitoring

**Key Components**:

* **GitHub Actions**: Automated testing and deployment
* **Briefcase Packaging**: Cross-platform application packaging
* **Logging Systems**: Centralized logging and audit trails
* **Quality Assurance**: Testing and code quality enforcement

**Critical Functions**:

# CI/CD Functions

def run\_automated\_tests() # [Testing] [cicd\_pipeline\_logging/github\_actions/] [Layer 6]

def package\_application() # [Distribution] [cicd\_pipeline\_logging/packaging/] [Layer 6]

def aggregate\_logs() # [Logging] [cicd\_pipeline\_logging/logging/] [Layer 6]

def perform\_security\_scan() # [Security] [cicd\_pipeline\_logging/github\_actions/security\_scans/] [Layer 6]

## **Complete Task Breakdown (1-100)**

### **Phase 1: Core RAG Foundation (Tasks 1-20)**

#### **Tasks 1-5: Environment Setup**

**Task 1: Project Structure Setup**

* **Folder**: wellness-companion-ai/ (root)
* **Function**: create\_directory\_structure() [Project Setup] [All Layers]
* **What**: Create complete 6-layer folder hierarchy
* **Dependencies**: None
* **Output**: Complete directory structure with all subfolders

**Task 2: Basic Environment Setup**

* **Folder**: config/development/
* **Function**: setup\_python\_environment() [Environment] [config/] [All Layers]
* **What**: Python virtual environment and basic dependencies
* **Dependencies**: Python 3.11+
* **Key Imports**: venv, pip, requirements.txt

**Task 3: Install Local QDRANT**

* **Folder**: data\_layer/vector\_databases/qdrant\_handlers/
* **Function**: initialize\_qdrant\_local() [Vector DB] [data\_layer/] [Layer 4]
* **What**: Docker QDRANT instance with basic connection
* **Dependencies**: Docker, qdrant-client
* **Key Imports**: qdrant\_client, docker

**Task 4: Install Local OLLAMA**

* **Folder**: aiml\_orchestration\_layer/ollama\_integration/
* **Function**: setup\_ollama\_local() [LLM Management] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Local OLLAMA installation with Gemma:3b model
* **Dependencies**: OLLAMA binary
* **Key Imports**: requests, subprocess

**Task 5: PDF Text Extraction**

* **Folder**: data\_layer/embedding\_pipeline/document\_processors/
* **Function**: extract\_pdf\_text() [Document Processing] [data\_layer/] [Layer 4]
* **What**: PDF parsing using PyPDF2
* **Dependencies**: PyPDF2, python-docx
* **Key Imports**: PyPDF2, docx, pathlib

#### **Tasks 6-10: Document Processing Pipeline**

**Task 6: Text Chunking Algorithm**

* **Folder**: data\_layer/embedding\_pipeline/text\_splitters/
* **Function**: chunk\_text() [Text Processing] [data\_layer/] [Layer 4]
* **What**: Split text into 512-token chunks with 50-token overlap
* **Dependencies**: tiktoken, transformers
* **Key Imports**: tiktoken, transformers

**Task 7: Embedding Generation**

* **Folder**: data\_layer/embedding\_pipeline/embedding\_generators/
* **Function**: generate\_embeddings() [Vectorization] [data\_layer/] [Layer 4]
* **What**: Convert text chunks to vectors using sentence-transformers
* **Dependencies**: sentence-transformers
* **Key Imports**: sentence\_transformers, torch, numpy

**Task 8: Vector Storage in QDRANT**

* **Folder**: data\_layer/vector\_databases/qdrant\_handlers/
* **Function**: store\_embeddings() [Vector Storage] [data\_layer/] [Layer 4]
* **What**: Store vectors with metadata in QDRANT
* **Dependencies**: qdrant-client
* **Key Imports**: qdrant\_client, uuid, json

**Task 9: Document Ingestion Pipeline**

* **Folder**: data\_layer/embedding\_pipeline/
* **Function**: ingest\_document() [Pipeline Orchestration] [data\_layer/] [Layer 4]
* **What**: Connect Tasks 5-8 into unified pipeline
* **Dependencies**: All previous tasks
* **Key Imports**: Custom modules from tasks 5-8

**Task 10: Test Document Upload**

* **Folder**: tests/unit\_tests/
* **Function**: test\_document\_ingestion() [Testing] [tests/] [Layer 6]
* **What**: Verify complete document processing pipeline
* **Dependencies**: pytest, test documents
* **Key Imports**: pytest, custom pipeline modules

#### **Tasks 11-15: Search and Response Generation**

**Task 11: Vector Similarity Search**

* **Folder**: aiml\_orchestration\_layer/vector\_search/
* **Function**: search\_similar\_vectors() [Similarity Search] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Query QDRANT for similar vectors using cosine similarity
* **Dependencies**: qdrant-client, numpy
* **Key Imports**: qdrant\_client, numpy, typing

**Task 12: Context Retrieval**

* **Folder**: aiml\_orchestration\_layer/rag\_orchestrator/
* **Function**: retrieve\_context() [Context Assembly] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Combine relevant chunks into LLM context
* **Dependencies**: Previous search results
* **Key Imports**: Custom search modules, typing

**Task 13: OLLAMA Response Generation**

* **Folder**: aiml\_orchestration\_layer/ollama\_integration/
* **Function**: generate\_response() [Response Generation] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Send context + query to OLLAMA for response
* **Dependencies**: requests, OLLAMA running
* **Key Imports**: requests, json, asyncio

**Task 14: Basic RAG Pipeline**

* **Folder**: aiml\_orchestration\_layer/rag\_orchestrator/
* **Function**: execute\_rag\_pipeline() [RAG Coordination] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Connect Tasks 11-13 into complete RAG flow
* **Dependencies**: All previous RAG components
* **Key Imports**: Custom modules from tasks 11-13

**Task 15: Test Complete RAG Flow**

* **Folder**: tests/unit\_tests/
* **Function**: test\_end\_to\_end\_rag() [Integration Testing] [tests/] [Layer 6]
* **What**: Test document upload → query → response flow
* **Dependencies**: pytest, complete RAG pipeline
* **Key Imports**: pytest, custom RAG modules

#### **Tasks 16-20: Web Search Integration**

**Task 16: Confidence Scoring**

* **Folder**: aiml\_orchestration\_layer/confidence\_scoring/
* **Function**: calculate\_confidence() [Quality Assessment] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Score relevance based on similarity and result quality
* **Dependencies**: numpy, scipy
* **Key Imports**: numpy, scipy.stats

**Task 17: Tavily API Integration**

* **Folder**: aiml\_orchestration\_layer/tavily\_integration/
* **Function**: setup\_tavily\_client() [Web Search Setup] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Configure Tavily API for web search
* **Dependencies**: requests, Tavily API key
* **Key Imports**: requests, os, json

**Task 18: Web Search Handler**

* **Folder**: aiml\_orchestration\_layer/tavily\_integration/
* **Function**: perform\_web\_search() [Web Search] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Process queries for web search and format results
* **Dependencies**: Tavily API client
* **Key Imports**: requests, json, typing

**Task 19: Hybrid Search Logic**

* **Folder**: aiml\_orchestration\_layer/rag\_orchestrator/
* **Function**: execute\_hybrid\_search() [Hybrid Search] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Trigger web search when confidence < 0.7
* **Dependencies**: Confidence scoring, web search
* **Key Imports**: Custom confidence and web search modules

**Task 20: Test Hybrid Search**

* **Folder**: tests/unit\_tests/
* **Function**: test\_hybrid\_search\_flow() [Hybrid Testing] [tests/] [Layer 6]
* **What**: Test low-confidence queries trigger web search
* **Dependencies**: pytest, hybrid search components
* **Key Imports**: pytest, custom hybrid search modules

### **Phase 2: API Layer (Tasks 21-32)**

#### **Tasks 21-27: Core API Development**

**Task 21: Basic FastAPI Setup**

* **Folder**: core\_backend\_layer/api\_gateway/
* **Function**: create\_fastapi\_app() [API Setup] [core\_backend\_layer/] [Layer 2]
* **What**: FastAPI app with CORS, middleware, health endpoint
* **Dependencies**: fastapi, uvicorn
* **Key Imports**: fastapi, fastapi.middleware.cors, uvicorn

**Task 22: Document Upload API**

* **Folder**: core\_backend\_layer/document\_api/
* **Function**: upload\_document\_endpoint() [Document API] [core\_backend\_layer/] [Layer 2]
* **What**: POST /documents/upload for file uploads
* **Dependencies**: fastapi, python-multipart
* **Key Imports**: fastapi, fastapi.UploadFile, custom pipeline modules

**Task 23: Semantic Search API**

* **Folder**: core\_backend\_layer/search\_api/
* **Function**: semantic\_search\_endpoint() [Search API] [core\_backend\_layer/] [Layer 2]
* **What**: POST /search/semantic for vector search
* **Dependencies**: fastapi, custom search modules
* **Key Imports**: fastapi, custom vector search modules

**Task 24: Web Search API**

* **Folder**: core\_backend\_layer/search\_api/
* **Function**: web\_search\_endpoint() [Web Search API] [core\_backend\_layer/] [Layer 2]
* **What**: POST /search/web for Tavily web search
* **Dependencies**: fastapi, custom Tavily modules
* **Key Imports**: fastapi, custom Tavily integration modules

**Task 25: Hybrid Search API**

* **Folder**: core\_backend\_layer/search\_api/
* **Function**: hybrid\_search\_endpoint() [Hybrid Search API] [core\_backend\_layer/] [Layer 2]
* **What**: POST /search/hybrid for combined search
* **Dependencies**: fastapi, custom hybrid modules
* **Key Imports**: fastapi, custom hybrid search modules

**Task 26: Health Check API**

* **Folder**: core\_backend\_layer/health\_endpoints/
* **Function**: health\_check\_endpoint() [Health API] [core\_backend\_layer/] [Layer 2]
* **What**: GET /health for system status
* **Dependencies**: fastapi, system monitoring
* **Key Imports**: fastapi, psutil, custom health modules

**Task 27: Test All APIs**

* **Folder**: tests/integration\_tests/
* **Function**: test\_all\_api\_endpoints() [API Testing] [tests/] [Layer 6]
* **What**: Verify all 5 APIs work with sample data
* **Dependencies**: pytest, requests, test data
* **Key Imports**: pytest, requests, json

#### **Tasks 28-32: Extended Document APIs**

**Task 28: Document List API**

* **Folder**: core\_backend\_layer/document\_api/
* **Function**: list\_documents\_endpoint() [Document Management] [core\_backend\_layer/] [Layer 2]
* **What**: GET /documents/list for document listing
* **Dependencies**: fastapi, database connection
* **Key Imports**: fastapi, custom database modules

**Task 29: Document Details API**

* **Folder**: core\_backend\_layer/document\_api/
* **Function**: get\_document\_details() [Document API] [core\_backend\_layer/] [Layer 2]
* **What**: GET /documents/{id} for document metadata
* **Dependencies**: fastapi, database modules
* **Key Imports**: fastapi, fastapi.Path, custom database modules

**Task 30: Document Delete API**

* **Folder**: core\_backend\_layer/document\_api/
* **Function**: delete\_document\_endpoint() [Document API] [core\_backend\_layer/] [Layer 2]
* **What**: DELETE /documents/{id} for document removal
* **Dependencies**: fastapi, database and vector DB modules
* **Key Imports**: fastapi, custom database and vector modules

**Task 31: Document Chunks API**

* **Folder**: core\_backend\_layer/document\_api/
* **Function**: get\_document\_chunks() [Document API] [core\_backend\_layer/] [Layer 2]
* **What**: GET /documents/{id}/chunks for chunk viewing
* **Dependencies**: fastapi, vector database modules
* **Key Imports**: fastapi, custom vector database modules

**Task 32: Test Document CRUD**

* **Folder**: tests/integration\_tests/
* **Function**: test\_document\_crud\_flow() [CRUD Testing] [tests/] [Layer 6]
* **What**: Test document lifecycle via APIs
* **Dependencies**: pytest, API client, test documents
* **Key Imports**: pytest, requests, test utilities

### **Phase 3: Database & Persistence (Tasks 33-45)**

#### **Tasks 33-40: Database Setup and Integration**

**Task 33: PostgreSQL Setup**

* **Folder**: data\_layer/relational\_database/
* **Function**: setup\_postgresql\_connection() [Database Setup] [data\_layer/] [Layer 4]
* **What**: Local PostgreSQL installation and connection handler
* **Dependencies**: psycopg2, sqlalchemy
* **Key Imports**: psycopg2, sqlalchemy, os

**Task 34: User Data Models**

* **Folder**: data\_layer/relational\_database/user\_models/
* **Function**: create\_user\_models() [Data Models] [data\_layer/] [Layer 4]
* **What**: SQLAlchemy models for users, documents, conversations
* **Dependencies**: sqlalchemy
* **Key Imports**: sqlalchemy, sqlalchemy.orm, datetime

**Task 35: Document Metadata Storage**

* **Folder**: data\_layer/relational\_database/document\_metadata/
* **Function**: store\_document\_metadata() [Metadata Management] [data\_layer/] [Layer 4]
* **What**: Store document metadata in PostgreSQL
* **Dependencies**: sqlalchemy, custom models
* **Key Imports**: sqlalchemy, custom user models

**Task 36: Database Migrations**

* **Folder**: data\_layer/relational\_database/migration\_tools/
* **Function**: setup\_alembic\_migrations() [Migration Management] [data\_layer/] [Layer 4]
* **What**: Alembic for database schema versioning
* **Dependencies**: alembic, sqlalchemy
* **Key Imports**: alembic, sqlalchemy

**Task 37: Persistent QDRANT Setup**

* **Folder**: data\_layer/vector\_databases/qdrant\_handlers/
* **Function**: configure\_persistent\_qdrant() [Vector DB Persistence] [data\_layer/] [Layer 4]
* **What**: QDRANT with persistent storage configuration
* **Dependencies**: qdrant-client, docker
* **Key Imports**: qdrant\_client, docker, pathlib

**Task 38: File Storage Handler**

* **Folder**: data\_layer/file\_storage/local\_cache/
* **Function**: manage\_local\_file\_storage() [File Management] [data\_layer/] [Layer 4]
* **What**: Local file storage with proper management
* **Dependencies**: pathlib, shutil
* **Key Imports**: pathlib, shutil, os, hashlib

**Task 39: Update APIs for Persistence**

* **Folder**: core\_backend\_layer/document\_api/
* **Function**: update\_apis\_for\_persistence() [API Enhancement] [core\_backend\_layer/] [Layer 2]
* **What**: Modify APIs to use PostgreSQL and file storage
* **Dependencies**: Updated database modules
* **Key Imports**: Custom database and file storage modules

**Task 40: Test Data Persistence**

* **Folder**: tests/integration\_tests/
* **Function**: test\_data\_persistence() [Persistence Testing] [tests/] [Layer 6]
* **What**: Verify data persists across system restarts
* **Dependencies**: pytest, database modules
* **Key Imports**: pytest, custom database modules

#### **Tasks 41-45: Caching and Performance**

**Task 41: Redis Setup**

* **Folder**: data\_layer/redis\_cache/
* **Function**: setup\_redis\_connection() [Cache Setup] [data\_layer/] [Layer 4]
* **What**: Local Redis installation and connection handler
* **Dependencies**: redis
* **Key Imports**: redis, os, json

**Task 42: Query Result Caching**

* **Folder**: data\_layer/redis\_cache/query\_cache/
* **Function**: cache\_search\_results() [Result Caching] [data\_layer/] [Layer 4]
* **What**: Cache search results with TTL for performance
* **Dependencies**: redis, custom search modules
* **Key Imports**: redis, json, hashlib

**Task 43: Session Storage**

* **Folder**: data\_layer/redis\_cache/session\_store/
* **Function**: manage\_session\_data() [Session Management] [data\_layer/] [Layer 4]
* **What**: Store temporary session data in Redis
* **Dependencies**: redis
* **Key Imports**: redis, json, uuid

**Task 44: Rate Limiting Storage**

* **Folder**: data\_layer/redis\_cache/rate\_limit\_store/
* **Function**: implement\_rate\_limiting() [Rate Limiting] [data\_layer/] [Layer 4]
* **What**: Store rate limiting counters in Redis
* **Dependencies**: redis, slowapi
* **Key Imports**: redis, slowapi, time

**Task 45: Test Caching**

* **Folder**: tests/integration\_tests/
* **Function**: test\_caching\_performance() [Cache Testing] [tests/] [Layer 6]
* **What**: Verify caching improves query performance
* **Dependencies**: pytest, time measurement
* **Key Imports**: pytest, time, custom cache modules

### **Phase 4: Authentication System (Tasks 46-53)**

#### **Tasks 46-53: User Management and Security**

**Task 46: Google OAuth Setup**

* **Folder**: core\_backend\_layer/authentication/
* **Function**: setup\_google\_oauth() [OAuth Setup] [core\_backend\_layer/] [Layer 2]
* **What**: Google OAuth application and flow handler
* **Dependencies**: authlib, httpx
* **Key Imports**: authlib, httpx, os

**Task 47: JWT Token Management**

* **Folder**: core\_backend\_layer/authentication/
* **Function**: manage\_jwt\_tokens() [Token Management] [core\_backend\_layer/] [Layer 2]
* **What**: JWT access/refresh token generation and validation
* **Dependencies**: pyjwt, cryptography
* **Key Imports**: jwt, cryptography, datetime

**Task 48: User Profile Storage**

* **Folder**: data\_layer/relational\_database/user\_models/
* **Function**: store\_user\_profiles() [User Storage] [data\_layer/] [Layer 4]
* **What**: Store user profiles and preferences in PostgreSQL
* **Dependencies**: sqlalchemy, custom models
* **Key Imports**: sqlalchemy, custom user models

**Task 49: Session Management**

* **Folder**: core\_backend\_layer/authentication/
* **Function**: manage\_user\_sessions() [Session Management] [core\_backend\_layer/] [Layer 2]
* **What**: Manage user sessions using Redis
* **Dependencies**: redis, custom session modules
* **Key Imports**: redis, custom session storage modules

**Task 50: Authentication APIs**

* **Folder**: core\_backend\_layer/authentication/
* **Function**: create\_auth\_endpoints() [Auth APIs] [core\_backend\_layer/] [Layer 2]
* **What**: Login, callback, refresh, logout, profile APIs
* **Dependencies**: fastapi, custom auth modules
* **Key Imports**: fastapi, custom authentication modules

**Task 51: User Management APIs**

* **Folder**: core\_backend\_layer/user\_management/
* **Function**: create\_user\_management\_apis() [User APIs] [core\_backend\_layer/] [Layer 2]
* **What**: User profile, preferences, usage statistics APIs
* **Dependencies**: fastapi, custom user modules
* **Key Imports**: fastapi, custom user management modules

**Task 52: Multi-user Document Isolation**

* **Folder**: data\_layer/relational\_database/
* **Function**: implement\_data\_isolation() [Data Security] [data\_layer/] [Layer 4]
* **What**: Ensure users can only access their own data
* **Dependencies**: sqlalchemy, custom security modules
* **Key Imports**: sqlalchemy, custom security modules

**Task 53: Test Multi-user System**

* **Folder**: tests/integration\_tests/
* **Function**: test\_multiuser\_isolation() [Security Testing] [tests/] [Layer 6]
* **What**: Test multiple users cannot access each other's data
* **Dependencies**: pytest, multiple test accounts
* **Key Imports**: pytest, custom auth modules, test utilities

### **Phase 5: Desktop Application (Tasks 54-69)**

#### **Tasks 54-62: Core Desktop Interface**

**Task 54: Basic PyQt6 Application**

* **Folder**: desktop\_layer/ui\_components/
* **Function**: create\_main\_application() [UI Framework] [desktop\_layer/] [Layer 1]
* **What**: Basic PyQt6 window with menu bar and main area
* **Dependencies**: PyQt6
* **Key Imports**: PyQt6.QtWidgets, PyQt6.QtCore, PyQt6.QtGui

**Task 55: Core Orchestrator**

* **Folder**: desktop\_layer/core\_orchestrator/
* **Function**: setup\_backend\_communication() [Communication Hub] [desktop\_layer/] [Layer 1]
* **What**: Communication layer between UI and backend APIs
* **Dependencies**: requests, PyQt6
* **Key Imports**: requests, PyQt6.QtCore, json

**Task 56: Login Interface**

* **Folder**: desktop\_layer/authentication\_ui/
* **Function**: create\_login\_interface() [Auth UI] [desktop\_layer/] [Layer 1]
* **What**: Login screen with OAuth flow in embedded browser
* **Dependencies**: PyQt6, PyQtWebEngine
* **Key Imports**: PyQt6.QtWidgets, PyQt6.QtWebEngineWidgets

**Task 57: Document Upload Interface**

* **Folder**: desktop\_layer/document\_interface/
* **Function**: create\_upload\_interface() [Document UI] [desktop\_layer/] [Layer 1]
* **What**: Drag-and-drop file upload with progress bars
* **Dependencies**: PyQt6
* **Key Imports**: PyQt6.QtWidgets, PyQt6.QtCore

**Task 58: Chat Interface**

* **Folder**: desktop\_layer/chat\_interface/
* **Function**: create\_chat\_interface() [Chat UI] [desktop\_layer/] [Layer 1]
* **What**: Chat window with input, history, and response display
* **Dependencies**: PyQt6
* **Key Imports**: PyQt6.QtWidgets, PyQt6.QtCore

**Task 59: Settings Panel**

* **Folder**: desktop\_layer/settings\_ui/
* **Function**: create\_settings\_panel() [Settings UI] [desktop\_layer/] [Layer 1]
* **What**: Settings interface for model selection and preferences
* **Dependencies**: PyQt6
* **Key Imports**: PyQt6.QtWidgets, PyQt6.QtCore

**Task 60: Theme Engine**

* **Folder**: desktop\_layer/theme\_engine/
* **Function**: implement\_theme\_system() [Theme Management] [desktop\_layer/] [Layer 1]
* **What**: QSS styling system for dark/light themes
* **Dependencies**: PyQt6, QDarkStyle
* **Key Imports**: PyQt6.QtWidgets, qdarkstyle

**Task 61: System Tray Integration**

* **Folder**: desktop\_layer/system\_tray/
* **Function**: setup\_system\_tray() [System Integration] [desktop\_layer/] [Layer 1]
* **What**: System tray icon with menu and notifications
* **Dependencies**: PyQt6
* **Key Imports**: PyQt6.QtWidgets, PyQt6.QtGui

**Task 62: Test Desktop Application**

* **Folder**: tests/e2e\_tests/
* **Function**: test\_desktop\_workflow() [E2E Testing] [tests/] [Layer 6]
* **What**: Test complete desktop app workflow
* **Dependencies**: pytest-qt, PyQt6
* **Key Imports**: pytest, pytestqt, custom desktop modules

#### **Tasks 63-69: Real-time Features and Enhancements**

**Task 63: WebSocket Server Setup**

* **Folder**: core\_backend\_layer/websocket\_handlers/
* **Function**: setup\_websocket\_server() [Real-time Communication] [core\_backend\_layer/] [Layer 2]
* **What**: Add WebSocket support to FastAPI
* **Dependencies**: fastapi, websockets
* **Key Imports**: fastapi, websockets, asyncio

**Task 64: Real-time Chat**

* **Folder**: desktop\_layer/chat\_interface/
* **Function**: implement\_realtime\_chat() [Real-time Chat] [desktop\_layer/] [Layer 1]
* **What**: Real-time chat with streaming responses via WebSocket
* **Dependencies**: PyQt6, websockets
* **Key Imports**: PyQt6.QtCore, websockets, asyncio

**Task 65: Upload Progress Tracking**

* **Folder**: desktop\_layer/document\_interface/
* **Function**: track\_upload\_progress() [Progress Tracking] [desktop\_layer/] [Layer 1]
* **What**: Real-time upload progress via WebSocket
* **Dependencies**: PyQt6, websockets
* **Key Imports**: PyQt6.QtCore, websockets

**Task 66: System Notifications**

* **Folder**: desktop\_layer/system\_tray/
* **Function**: handle\_system\_notifications() [Notifications] [desktop\_layer/] [Layer 1]
* **What**: Real-time notifications for processing updates
* **Dependencies**: PyQt6, plyer
* **Key Imports**: PyQt6.QtWidgets, plyer

**Task 67: Document Preview**

* **Folder**: desktop\_layer/document\_interface/
* **Function**: implement\_document\_preview() [Document Preview] [desktop\_layer/] [Layer 1]
* **What**: PDF/text preview functionality in document manager
* **Dependencies**: PyQt6, PyMuPDF
* **Key Imports**: PyQt6.QtWidgets, fitz

**Task 68: Search History**

* **Folder**: desktop\_layer/chat\_interface/
* **Function**: manage\_search\_history() [History Management] [desktop\_layer/] [Layer 1]
* **What**: Store and display previous queries and conversations
* **Dependencies**: PyQt6, SQLite
* **Key Imports**: PyQt6.QtWidgets, sqlite3

**Task 69: Test Real-time Features**

* **Folder**: tests/e2e\_tests/
* **Function**: test\_realtime\_features() [Real-time Testing] [tests/] [Layer 6]
* **What**: Test all real-time features work correctly
* **Dependencies**: pytest-qt, websockets
* **Key Imports**: pytest, pytestqt, websockets

### **Phase 6: Local Packaging (Tasks 70-75)**

#### **Tasks 70-75: Application Distribution**

**Task 70: Briefcase Configuration**

* **Folder**: cicd\_pipeline\_logging/packaging/briefcase\_configs/
* **Function**: configure\_briefcase\_packaging() [Packaging Setup] [cicd\_pipeline\_logging/] [Layer 6]
* **What**: Cross-platform packaging for Windows, macOS, Linux
* **Dependencies**: briefcase
* **Key Imports**: briefcase, toml

**Task 71: Code Signing Setup**

* **Folder**: cicd\_pipeline\_logging/packaging/code\_signing/
* **Function**: setup\_code\_signing() [Security Signing] [cicd\_pipeline\_logging/] [Layer 6]
* **What**: Code signing certificates for security compliance
* **Dependencies**: Platform-specific signing tools
* **Key Imports**: subprocess, os, pathlib

**Task 72: Application Logging**

* **Folder**: cicd\_pipeline\_logging/logging/application\_logs/
* **Function**: implement\_application\_logging() [Logging System] [cicd\_pipeline\_logging/] [Layer 6]
* **What**: Comprehensive logging throughout application
* **Dependencies**: logging, loguru
* **Key Imports**: logging, loguru, json

**Task 73: Error Handling**

* **Folder**: shared/exceptions/
* **Function**: implement\_error\_handling() [Error Management] [shared/] [All Layers]
* **What**: Proper error handling and user-friendly messages
* **Dependencies**: Custom exception classes
* **Key Imports**: traceback, logging, custom exception classes

**Task 74: Build Installers**

* **Folder**: cicd\_pipeline\_logging/packaging/
* **Function**: build\_platform\_installers() [Installer Creation] [cicd\_pipeline\_logging/] [Layer 6]
* **What**: Create DMG, EXE, and AppImage installers
* **Dependencies**: briefcase, platform tools
* **Key Imports**: briefcase, subprocess, shutil

**Task 75: Test Installers**

* **Folder**: tests/e2e\_tests/
* **Function**: test\_installer\_functionality() [Installer Testing] [tests/] [Layer 6]
* **What**: Test installers on clean machines for all platforms
* **Dependencies**: Virtual machines, test environments
* **Key Imports**: Test environment specific

### **Phase 7: Cloud Infrastructure (Tasks 76-86)**

#### **Tasks 76-80: AWS Core Services**

**Task 76: AWS RDS Setup**

* **Folder**: infrastructure\_deployment\_layer/aws\_infrastructure/rds\_configs/
* **Function**: setup\_aws\_rds() [Cloud Database] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: PostgreSQL on AWS RDS for cloud database
* **Dependencies**: boto3, psycopg2
* **Key Imports**: boto3, psycopg2, json

**Task 77: AWS S3 Setup**

* **Folder**: infrastructure\_deployment\_layer/aws\_infrastructure/s3\_configs/
* **Function**: setup\_aws\_s3() [Cloud Storage] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: S3 buckets for file storage with security
* **Dependencies**: boto3
* **Key Imports**: boto3, botocore, json

**Task 78: AWS EC2 Setup**

* **Folder**: infrastructure\_deployment\_layer/aws\_infrastructure/ec2\_configs/
* **Function**: setup\_aws\_ec2() [Cloud Computing] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: EC2 instance for application server
* **Dependencies**: boto3, paramiko
* **Key Imports**: boto3, paramiko, fabric

**Task 79: Docker Containerization**

* **Folder**: infrastructure\_deployment\_layer/docker\_configs/
* **Function**: containerize\_services() [Containerization] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: Docker containers for all services
* **Dependencies**: docker, docker-compose
* **Key Imports**: docker, yaml

**Task 80: AWS ElastiCache Setup**

* **Folder**: infrastructure\_deployment\_layer/aws\_infrastructure/elasticache\_configs/
* **Function**: setup\_aws\_elasticache() [Cloud Caching] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: Redis on AWS ElastiCache for cloud caching
* **Dependencies**: boto3, redis
* **Key Imports**: boto3, redis, json

#### **Tasks 81-86: Infrastructure Services**

**Task 81: NGINX Configuration**

* **Folder**: infrastructure\_deployment\_layer/docker\_configs/nginx\_proxy/
* **Function**: configure\_nginx\_proxy() [Load Balancing] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: NGINX reverse proxy and load balancing
* **Dependencies**: nginx, docker
* **Key Imports**: Configuration files, subprocess

**Task 82: AWS API Gateway**

* **Folder**: infrastructure\_deployment\_layer/aws\_infrastructure/api\_gateway\_configs/
* **Function**: setup\_aws\_api\_gateway() [API Management] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: AWS API Gateway for API management and security
* **Dependencies**: boto3
* **Key Imports**: boto3, json

**Task 83: SSL Certificate Setup**

* **Folder**: infrastructure\_deployment\_layer/security/ssl\_certificates/
* **Function**: configure\_ssl\_certificates() [Security] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: SSL certificates for HTTPS encryption
* **Dependencies**: certbot, openssl
* **Key Imports**: subprocess, ssl

**Task 84: CloudWatch Monitoring**

* **Folder**: infrastructure\_deployment\_layer/monitoring/cloudwatch\_configs/
* **Function**: setup\_cloudwatch\_monitoring() [Monitoring] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: AWS CloudWatch for system monitoring and alerting
* **Dependencies**: boto3
* **Key Imports**: boto3, json

**Task 85: Sentry Error Tracking**

* **Folder**: infrastructure\_deployment\_layer/monitoring/sentry\_integration/
* **Function**: integrate\_sentry\_monitoring() [Error Tracking] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: Sentry for error tracking and performance monitoring
* **Dependencies**: sentry-sdk
* **Key Imports**: sentry\_sdk, logging

**Task 86: Production Deployment**

* **Folder**: infrastructure\_deployment\_layer/
* **Function**: deploy\_to\_production() [Production Deployment] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: Deploy complete system to AWS and test functionality
* **Dependencies**: All infrastructure components
* **Key Imports**: All infrastructure modules

### **Phase 8: Performance Optimization (Tasks 87-91)**

#### **Tasks 87-91: Cythonization and Performance**

**Task 87: Text Processing Cythonization**

* **Folder**: data\_layer/embedding\_pipeline/cython\_optimized/
* **Function**: cythonize\_text\_processing() [Performance Optimization] [data\_layer/] [Layer 4]
* **What**: Convert text chunking and processing to Cython
* **Dependencies**: Cython, numpy
* **Key Imports**: cython, numpy, pyx files

**Task 88: Vector Operations Cythonization**

* **Folder**: aiml\_orchestration\_layer/cython\_optimized/
* **Function**: cythonize\_vector\_operations() [Performance Optimization] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Convert similarity calculations to Cython
* **Dependencies**: Cython, numpy
* **Key Imports**: cython, numpy, scipy

**Task 89: Embedding Pipeline Cythonization**

* **Folder**: data\_layer/embedding\_pipeline/cython\_optimized/
* **Function**: cythonize\_embedding\_pipeline() [Performance Optimization] [data\_layer/] [Layer 4]
* **What**: Optimize batch embedding generation with Cython
* **Dependencies**: Cython, torch
* **Key Imports**: cython, torch, numpy

**Task 90: Search Algorithm Cythonization**

* **Folder**: aiml\_orchestration\_layer/vector\_search/
* **Function**: cythonize\_search\_algorithms() [Performance Optimization] [aiml\_orchestration\_layer/] [Layer 3]
* **What**: Optimize search result ranking and filtering
* **Dependencies**: Cython, numpy
* **Key Imports**: cython, numpy, scipy

**Task 91: Performance Benchmarking**

* **Folder**: tests/performance\_tests/
* **Function**: benchmark\_system\_performance() [Performance Testing] [tests/] [Layer 6]
* **What**: Run comprehensive performance tests
* **Dependencies**: pytest-benchmark, time
* **Key Imports**: pytest, time, memory\_profiler

### **Phase 9: Production Readiness (Tasks 92-100)**

#### **Tasks 92-100: Final Production Features**

**Task 92: GitHub Actions CI/CD**

* **Folder**: cicd\_pipeline\_logging/github\_actions/
* **Function**: setup\_github\_actions() [CI/CD Pipeline] [cicd\_pipeline\_logging/] [Layer 6]
* **What**: Automated testing, building, and deployment
* **Dependencies**: GitHub Actions, YAML
* **Key Imports**: YAML configuration files

**Task 93: Automated Testing Suite**

* **Folder**: tests/
* **Function**: implement\_comprehensive\_testing() [Testing Suite] [tests/] [Layer 6]
* **What**: Complete test coverage with unit, integration, E2E tests
* **Dependencies**: pytest, pytest-cov
* **Key Imports**: pytest, pytest\_cov, all test modules

**Task 94: Usage Analytics**

* **Folder**: core\_backend\_layer/analytics/
* **Function**: implement\_usage\_analytics() [Analytics] [core\_backend\_layer/] [Layer 2]
* **What**: User behavior and system usage analytics
* **Dependencies**: sqlalchemy, redis
* **Key Imports**: sqlalchemy, redis, json

**Task 95: Admin Dashboard APIs**

* **Folder**: core\_backend\_layer/admin\_api/
* **Function**: create\_admin\_dashboard() [Admin Interface] [core\_backend\_layer/] [Layer 2]
* **What**: Admin APIs for user management and monitoring
* **Dependencies**: fastapi, custom admin modules
* **Key Imports**: fastapi, custom admin modules

**Task 96: Documentation Generation**

* **Folder**: docs/
* **Function**: generate\_project\_documentation() [Documentation] [docs/] [Layer 6]
* **What**: Comprehensive API documentation and user guides
* **Dependencies**: sphinx, mkdocs
* **Key Imports**: sphinx, mkdocs

**Task 97: Security Audit**

* **Folder**: infrastructure\_deployment\_layer/security/
* **Function**: perform\_security\_audit() [Security Assessment] [infrastructure\_deployment\_layer/] [Layer 5]
* **What**: Security audit and best practices implementation
* **Dependencies**: bandit, safety
* **Key Imports**: bandit, safety, security tools

**Task 98: Load Testing**

* **Folder**: tests/performance\_tests/
* **Function**: perform\_load\_testing() [Load Testing] [tests/] [Layer 6]
* **What**: Load testing for expected user capacity
* **Dependencies**: locust, artillery
* **Key Imports**: locust, load testing tools

**Task 99: Production Optimization**

* **Folder**: All folders
* **Function**: optimize\_for\_production() [Final Optimization] [All Layers]
* **What**: Final optimization pass for production readiness
* **Dependencies**: All system components
* **Key Imports**: All system modules

**Task 100: Production Launch**

* **Folder**: cicd\_pipeline\_logging/
* **Function**: launch\_production\_system() [Production Launch] [cicd\_pipeline\_logging/] [Layer 6]
* **What**: Deploy to production and monitor stability
* **Dependencies**: All production components
* **Key Imports**: All deployment modules

## **Data Flow Architecture**

### **User Query Processing Flow**

#### **Primary Flow (Local Search)**

1. **User Input** → Desktop Application (PyQt6)
2. **Query Pre-processing** → Core Orchestrator
3. **API Request** → FastAPI Gateway (Core Backend Layer)
4. **Vector Search** → QDRANT Database (Data Layer)
5. **Context Retrieval** → RAG Orchestrator (AI/ML Layer)
6. **LLM Generation** → OLLAMA Local Model
7. **Response Formatting** → Response Synthesizer
8. **UI Display** → Desktop Application

#### **Fallback Flow (Web Search)**

1. **Confidence Scoring** → Confidence < 0.7 threshold
2. **Web Search Trigger** → Tavily API Integration
3. **Query Optimization** → Web search query processing
4. **Result Processing** → Web result filtering and formatting
5. **Hybrid Response** → Combined local + web results
6. **Enhanced Generation** → LLM with enriched context
7. **Source Attribution** → Response with source indicators

### **Document Ingestion Flow**

#### **Document Upload Process**

1. **File Selection** → Desktop drag-and-drop interface
2. **File Validation** → Client-side type and size checks
3. **Upload Transfer** → Chunked HTTP transfer to backend
4. **Virus Scanning** → Backend security validation
5. **Document Processing** → PDF/DOCX/TXT parsing
6. **Text Extraction** → Content extraction and cleaning
7. **Text Chunking** → 512-token chunks with 50-token overlap
8. **Embedding Generation** → Sentence Transformers vectorization
9. **Vector Storage** → QDRANT database with metadata
10. **File Storage** → S3 with encryption
11. **Metadata Indexing** → PostgreSQL document metadata
12. **User Notification** → Processing completion status

### **Authentication Flow**

#### **Initial Authentication**

1. **Sign-in Click** → Desktop application
2. **OAuth Initiation** → Google OAuth flow
3. **Browser Opening** → System browser for authorization
4. **User Authorization** → Google permission grants
5. **Callback Handling** → Authorization code return
6. **Token Exchange** → Access/refresh token generation
7. **Profile Creation** → PostgreSQL user record
8. **JWT Generation** → API access token creation
9. **Token Storage** → Desktop keyring storage
10. **Session Establishment** → Redis session management

### **Real-time Communication Flow**

#### **WebSocket Connection Management**

1. **Connection Establishment** → Desktop client to FastAPI
2. **Authentication** → JWT token validation
3. **Channel Subscription** → User-specific update channels
4. **Message Routing** → Real-time message delivery
5. **Heartbeat Monitoring** → Connection health checks
6. **Reconnection Logic** → Automatic reconnection handling
7. **Graceful Degradation** → HTTP polling fallback

## **API Reference Summary**

### **Authentication APIs (/auth)**

* **POST /auth/login** - Initiate OAuth login flow
* **GET /auth/callback** - Handle OAuth callback
* **POST /auth/refresh** - Refresh access tokens
* **POST /auth/logout** - Terminate user session
* **GET /auth/profile** - Get user profile information
* **PUT /auth/profile** - Update user profile
* **POST /auth/validate** - Validate JWT token
* **POST /auth/revoke** - Revoke refresh token

### **Document Management APIs (/documents)**

* **POST /documents/upload** - Upload single document
* **POST /documents/upload/batch** - Upload multiple documents
* **GET /documents/list** - List user documents
* **GET /documents/{doc\_id}** - Get document details
* **DELETE /documents/{doc\_id}** - Delete document
* **GET /documents/{doc\_id}/download** - Download original file
* **GET /documents/{doc\_id}/preview** - Get document preview
* **GET /documents/{doc\_id}/chunks** - Get document chunks
* **POST /documents/process** - Manually trigger processing
* **GET /documents/status/{job\_id}** - Check processing status

### **Search & Knowledge APIs (/search)**

* **POST /search/semantic** - Vector similarity search
* **POST /search/web** - Web search fallback
* **POST /search/hybrid** - Combined local + web search
* **GET /search/status** - Get search service status
* **POST /search/similar** - Find similar documents
* **POST /search/rebuild** - Rebuild search index
* **GET /search/autocomplete** - Query suggestions

### **Chat & Conversation APIs (/chat)**

* **GET /chat/conversations** - List user conversations
* **POST /chat/conversations** - Create new conversation
* **GET /chat/conversations/{conv\_id}** - Get conversation details
* **DELETE /chat/conversations/{conv\_id}** - Delete conversation
* **GET /chat/conversations/{conv\_id}/messages** - Get messages
* **POST /chat/conversations/{conv\_id}/messages** - Send message
* **POST /chat/quick-query** - One-off query without conversation
* **WebSocket /ws/chat** - Real-time chat streaming

### **System Health & Monitoring APIs (/health)**

* **GET /health** - Basic health check
* **GET /health/detailed** - Comprehensive health information
* **GET /health/database** - Database connectivity status
* **GET /health/ai** - AI services health
* **GET /health/storage** - Storage systems status
* **GET /health/metrics** - System performance metrics

## **Key Import Dependencies by Layer**

### **Desktop Layer (PyQt6 Frontend)**

# Core UI Framework

from PyQt6.QtWidgets import \*

from PyQt6.QtCore import \*

from PyQt6.QtGui import \*

from PyQt6.QtWebEngineWidgets import QWebEngineView

# Styling and Themes

import qdarkstyle

from PyQt6.QtCore import QSettings

# System Integration

import sys

import os

from pathlib import Path

# Communication

import requests

import json

import websockets

import asyncio

### **Core Backend Layer (FastAPI)**

# Web Framework

from fastapi import FastAPI, Depends, HTTPException, UploadFile, WebSocket

from fastapi.middleware.cors import CORSMiddleware

from fastapi.security import OAuth2PasswordBearer

import uvicorn

# Authentication

from authlib.integrations.fastapi\_oauth2 import OAuth2Token

import jwt

from passlib.context import CryptContext

# Database

from sqlalchemy import create\_engine, Column, Integer, String, DateTime

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker, Session

import alembic

# Caching and Rate Limiting

import redis

from slowapi import Limiter, \_rate\_limit\_exceeded\_handler

from slowapi.util import get\_remote\_address

# Background Tasks

import celery

from celery import Celery

### **AI/ML Orchestration Layer**

# LLM Integration

import requests # For OLLAMA API calls

import json

import asyncio

# RAG Framework

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.vectorstores import Qdrant

from langchain.llms import Ollama

from langchain.chains import RetrievalQA

# Alternative: LlamaIndex

from llama\_index import VectorStoreIndex, SimpleDirectoryReader

from llama\_index.vector\_stores import QdrantVectorStore

from llama\_index.embeddings import HuggingFaceEmbedding

# Vector Operations

import numpy as np

from scipy.spatial.distance import cosine

from sklearn.metrics.pairwise import cosine\_similarity

# Web Search

import requests # For Tavily API

from urllib.parse import quote\_plus

# Confidence Scoring

from scipy import stats

import numpy as np

### **Data Layer**

# Document Processing

import PyPDF2

from docx import Document

import textract

from pathlib import Path

import magic # File type detection

# Text Processing

import tiktoken # Token counting

from transformers import AutoTokenizer

import re

import unicodedata

# Embeddings

from sentence\_transformers import SentenceTransformer

import torch

import numpy as np

# Vector Database

from qdrant\_client import QdrantClient

from qdrant\_client.http import models

from qdrant\_client.http.models import Distance, VectorParams

import uuid

# Relational Database

import psycopg2

from sqlalchemy import create\_engine, text

from sqlalchemy.orm import sessionmaker

import alembic

# File Storage

import boto3

from botocore.exceptions import ClientError

import hashlib

import shutil

# Caching

import redis

import json

import pickle

from datetime import timedelta

### **Infrastructure Deployment Layer**

# AWS Services

import boto3

from botocore.exceptions import ClientError, NoCredentialsError

import botocore

# Docker

import docker

from docker.errors import DockerException

import yaml # For docker-compose

# Security

import ssl

import cryptography

from cryptography.fernet import Fernet

import secrets

# Monitoring

import sentry\_sdk

from sentry\_sdk.integrations.fastapi import FastApiIntegration

from sentry\_sdk.integrations.sqlalchemy import SqlalchemyIntegration

# Process Management

import subprocess

import systemd

import nginx

### **CI/CD Pipeline and Logging Layer**

# Packaging

import briefcase

from setuptools import setup, find\_packages

import wheel

# Testing

import pytest

import pytest\_cov

import pytest\_qt # For PyQt testing

import pytest\_asyncio

from unittest.mock import Mock, patch

# Logging

import logging

from loguru import logger

import structlog

import json

# Performance Testing

import locust

from locust import HttpUser, task, between

import time

import memory\_profiler

# Code Quality

import bandit # Security linting

import safety # Dependency vulnerability checking

import black # Code formatting

import isort # Import sorting

import mypy # Type checking

# Build Tools

import cython

from Cython.Build import cythonize

import numpy # For Cython integration

## **Performance Optimization Strategies**

### **Cythonization Targets**

1. **Text Processing Functions**
   * Chunking algorithms: 30-50% speed improvement
   * Token counting: 40% speed improvement
   * Text cleaning: 25% speed improvement
2. **Vector Operations**
   * Cosine similarity calculations: 60% speed improvement
   * Batch embedding processing: 45% speed improvement
   * Search result ranking: 35% speed improvement
3. **Embedding Pipeline**
   * Batch processing optimization: 3x faster indexing
   * Memory management: 50% reduction in memory usage

### **Model Performance Recommendations**

#### **Embedding Models**

| **Model** | **Dimensions** | **Accuracy** | **Speed** | **Memory** | **Use Case** |
| --- | --- | --- | --- | --- | --- |
| all-MiniLM-L12-v2 | 384 | 87% | Fast | 120MB | Balanced performance |
| all-MiniLM-L6-v2 | 384 | 85% | Very Fast | 90MB | Speed-optimized |
| all-mpnet-base-v2 | 768 | 89% | Medium | 420MB | Accuracy-optimized |

#### **LLM Models**

| **Model** | **Size** | **RAM Required** | **Accuracy** | **Speed** | **Use Case** |
| --- | --- | --- | --- | --- | --- |
| Gemma:3b | 2GB | 4GB | 82% | Fast | Resource-constrained |
| LLaMA2:7b | 3.8GB | 8GB | 85% | Medium | General purpose |
| Mistral:7b | 4.1GB | 8GB | 87% | Medium | High accuracy |

### **Database Performance**

| **Database** | **Performance** | **Setup Complexity** | **Scalability** | **Features** |
| --- | --- | --- | --- | --- |
| QDRANT | Excellent | Easy | High | Full-featured vector DB |
| FAISS | Excellent | Medium | High | Requires optimization |
| ChromaDB | Good | Easy | Medium | Basic features |

## **Security Implementation**

### **Authentication Security**

* **OAuth 2.0** with Google for secure user authentication
* **JWT tokens** with refresh token rotation
* **Session management** via Redis with TTL
* **Rate limiting** to prevent abuse
* **CSRF protection** on all state-changing endpoints

### **Data Security**

* **End-to-end encryption** for file storage in S3
* **Data isolation** ensuring users can only access their own data
* **SQL injection prevention** via parameterized queries
* **Input validation** on all user inputs
* **Secure file upload** with virus scanning

### **Infrastructure Security**

* **SSL/TLS encryption** for all communications
* **VPC isolation** for AWS resources
* **IAM roles** with least privilege access
* **Security groups** with minimal required ports
* **Regular security audits** and dependency updates

## **Deployment Architecture**

### **Local Development Environment**

* **QDRANT**: Docker container on localhost:6333
* **PostgreSQL**: Local installation on localhost:5432
* **Redis**: Local installation on localhost:6379
* **OLLAMA**: Local binary with downloaded models
* **FastAPI**: Development server on localhost:8000

### **Production Cloud Environment**

* **AWS RDS**: PostgreSQL database cluster
* **AWS S3**: Encrypted file storage buckets
* **AWS EC2**: Application server instances
* **AWS ElastiCache**: Redis cluster
* **AWS API Gateway**: Request routing and throttling
* **CloudWatch**: Monitoring and alerting
* **Sentry**: Error tracking and performance monitoring

### **Cross-Platform Distribution**

* **Windows**: .exe installer with embedded Python
* **macOS**: .dmg package with code signing
* **Linux**: AppImage for universal compatibility
* **Code Signing**: Platform-specific certificates for trust

## **Development Workflow**

### **Feature Development Cycle**

1. **Folder Creation**: Set up required directory structure
2. **Core Logic**: Build functionality without API layer
3. **Local Testing**: Test with Python scripts/Jupyter notebooks
4. **API Creation**: Expose functionality via FastAPI endpoints
5. **API Testing**: Validate endpoints with Postman/curl
6. **Integration**: Connect with existing codebase
7. **E2E Testing**: Test complete workflow
8. **Documentation**: Update API docs and guides
9. **Next Feature**: Move to next roadmap item

### **Quality Assurance**

* **Unit Tests**: Individual component testing with pytest
* **Integration Tests**: Cross-service testing
* **E2E Tests**: Complete workflow testing with pytest-qt
* **Performance Tests**: Benchmarking with pytest-benchmark
* **Security Tests**: Vulnerability scanning with bandit
* **Code Quality**: Linting with black, isort, mypy

### **Continuous Integration**

* **GitHub Actions**: Automated testing on commit
* **Automated Testing**: Full test suite on pull requests
* **Security Scanning**: Dependency vulnerability checks
* **Performance Monitoring**: Benchmark regression testing
* **Automated Deployment**: Production deployment on merge