# **Wellness Companion AI - Feature-by-Feature Development Roadmap**

## **🎯 Development Philosophy**

**Build → Test → API → Integrate → Next Feature**

Each phase must be **completely working** before moving to the next. No parallel development until core is solid.

## **🔥 PHASE 1: CORE RAG FOUNDATION**

**Goal:** Basic document ingestion → vectorization → search → response **Time:** 2-3 weeks

### **1.1 Core RAG Logic (/aiml\_orchestration\_layer/)**

📁 Folders to Create First:

├── aiml\_orchestration\_layer/rag\_orchestrator/

├── aiml\_orchestration\_layer/vector\_search/

├── data\_layer/embedding\_pipeline/

└── shared/utils/

**Tasks:**

1. **Document Text Extraction** - PDF/TXT parsing
2. **Text Chunking** - Split into 512-token chunks
3. **Embedding Generation** - Sentence Transformers
4. **Vector Storage** - Local QDRANT instance
5. **Similarity Search** - Cosine similarity retrieval
6. **Response Generation** - OLLAMA local LLM

**Test:** Single document → upload → query → get answer

### **1.2 Web Search Fallback (/aiml\_orchestration\_layer/tavily\_integration/)**

📁 Additional Folders:

└── aiml\_orchestration\_layer/tavily\_integration/

**Tasks:**

1. **Confidence Scoring** - Threshold-based decision (< 0.7)
2. **Tavily API Integration** - Web search when local fails
3. **Result Merging** - Combine local + web results
4. **Source Attribution** - Track where answers come from

**Test:** Query with low confidence → triggers web search → returns combined result

## **🔌 PHASE 2: API LAYER**

**Goal:** Expose core functionality via FastAPI **Time:** 1-2 weeks

### **2.1 Basic FastAPI Setup (/core\_backend\_layer/)**

📁 Folders to Create:

├── core\_backend\_layer/api\_gateway/

├── core\_backend\_layer/health\_endpoints/

└── shared/models/

**APIs to Build (5 endpoints):**

1. POST /documents/upload - Single document upload
2. GET /search/semantic - Vector search
3. POST /search/web - Web search fallback
4. POST /search/hybrid - Combined search
5. GET /health - System health check

**Test:** API calls work from Postman/curl → same functionality as Phase 1

### **2.2 Document Management APIs (/core\_backend\_layer/document\_api/)**

📁 Additional Folders:

└── core\_backend\_layer/document\_api/

**APIs to Add (4 endpoints):**

1. GET /documents/list - List uploaded documents
2. GET /documents/{id} - Get document details
3. DELETE /documents/{id} - Delete document
4. GET /documents/{id}/chunks - View document chunks

**Test:** Full document CRUD operations via API

## **💾 PHASE 3: DATABASE & STORAGE**

**Goal:** Persistent storage for all data **Time:** 2-3 weeks

### **3.1 Database Setup (/data\_layer/)**

📁 Folders to Create:

├── data\_layer/relational\_database/

├── data\_layer/vector\_databases/

├── data\_layer/file\_storage/

└── infrastructure\_deployment\_layer/docker\_configs/

**Tasks:**

1. **PostgreSQL Setup** - User data, document metadata
2. **QDRANT Production Setup** - Persistent vector storage
3. **Local File Storage** - Document file management
4. **Database Migrations** - Schema versioning

**New APIs (3 endpoints):**

1. GET /documents/status/{job\_id} - Processing status
2. POST /documents/process - Manual reprocessing
3. GET /admin/system/stats - Storage statistics

**Test:** Data persists across app restarts → all previous functionality works

### **3.2 Redis Caching (/data\_layer/redis\_cache/)**

📁 Additional Folders:

└── data\_layer/redis\_cache/

**Tasks:**

1. **Query Result Caching** - Cache search results
2. **Session Storage** - Temporary data storage
3. **Rate Limiting Data** - Request throttling

**Test:** Repeated queries are faster → cache invalidation works

## **🔒 PHASE 4: AUTHENTICATION**

**Goal:** User accounts and secure access **Time:** 2 weeks

### **4.1 Authentication System (/core\_backend\_layer/authentication/)**

📁 Folders to Create:

├── core\_backend\_layer/authentication/

├── core\_backend\_layer/user\_management/

└── data\_layer/relational\_database/user\_models/

**Tasks:**

1. **Google OAuth Integration** - Social login
2. **JWT Token Management** - Access/refresh tokens
3. **User Profile Storage** - PostgreSQL user tables
4. **Session Management** - Redis-based sessions

**New APIs (8 endpoints):**

1. POST /auth/login - Start OAuth flow
2. GET /auth/callback - Handle OAuth callback
3. POST /auth/refresh - Refresh tokens
4. POST /auth/logout - End session
5. GET /auth/profile - Get user info
6. PUT /auth/profile - Update profile
7. POST /auth/validate - Validate token
8. GET /users/usage - Usage statistics

**Test:** Users can login → their documents are private → multi-user works

## **📱 PHASE 5: DESKTOP UI/UX**

**Goal:** PyQt6 desktop application **Time:** 3-4 weeks

### **5.1 Basic Desktop App (/desktop\_layer/)**

📁 Folders to Create:

├── desktop\_layer/ui\_components/

├── desktop\_layer/core\_orchestrator/

├── desktop\_layer/authentication\_ui/

├── desktop\_layer/document\_interface/

├── desktop\_layer/chat\_interface/

└── desktop\_layer/theme\_engine/

**Tasks:**

1. **Login Screen** - OAuth integration
2. **Document Upload Interface** - Drag & drop
3. **Chat Interface** - Query input + response display
4. **Settings Panel** - Model selection, preferences
5. **System Tray Integration** - Background operation

**Test:** Complete desktop app → all API functionality accessible via UI

### **5.2 Advanced UI Features**

**Tasks:**

1. **Real-time Chat** - WebSocket integration
2. **Document Preview** - PDF/text viewer
3. **Search History** - Previous queries
4. **Dark/Light Themes** - QSS styling

**New APIs (WebSocket):**

1. WS /ws/chat - Real-time messaging
2. WS /ws/documents - Upload progress
3. WS /ws/notifications - System alerts

## **🚀 PHASE 6: DEPLOYMENT & PACKAGING**

**Goal:** Distributable application **Time:** 2-3 weeks

### **6.1 Local Packaging (/cicd\_pipeline\_logging/packaging/)**

📁 Folders to Create:

├── cicd\_pipeline\_logging/packaging/

├── cicd\_pipeline\_logging/logging/

└── desktop\_layer/briefcase\_config/

**Tasks:**

1. **Briefcase Configuration** - Cross-platform packaging
2. **Code Signing** - Security certificates
3. **Local Installers** - DMG/EXE/Linux packages
4. **Basic Logging** - Application logs

**Test:** Installable packages work on different OS → users can install and run

## **🌐 PHASE 7: CLOUD INFRASTRUCTURE (Optional)**

**Goal:** Production cloud deployment **Time:** 3-4 weeks

### **7.1 Basic Cloud Setup**

📁 Folders to Create:

├── infrastructure\_deployment\_layer/aws\_infrastructure/

├── infrastructure\_deployment\_layer/docker\_configs/

└── infrastructure\_deployment\_layer/security/

**Infrastructure Order:**

1. **AWS RDS** - PostgreSQL cloud database
2. **AWS S3** - File storage
3. **AWS EC2** - Application server
4. **Docker** - Application containerization

### **7.2 Advanced Cloud Features**

**Infrastructure Order:**

1. **AWS ElastiCache** - Redis cloud cache
2. **NGINX** - Reverse proxy and load balancing
3. **AWS API Gateway** - API management
4. **prometheus** - Monitoring and alerting
5. **Sentry** - Error tracking

## **🔧 PHASE 8: OPTIMIZATION & MONITORING**

**Goal:** Production-ready performance **Time:** 2-3 weeks

### **8.1 Performance Optimization**

📁 Folders to Create:

├── aiml\_orchestration\_layer/cython\_optimized/

├── data\_layer/embedding\_pipeline/cython\_optimized/

└── shared/cython\_core/

**Cythonization Order:**

1. **Text Processing** - Chunking algorithms
2. **Vector Operations** - Similarity calculations
3. **Embedding Pipeline** - Batch processing
4. **Search Algorithms** - Result ranking

### **8.2 Monitoring & Analytics**

📁 Additional Folders:

├── cicd\_pipeline\_logging/logging/

└── infrastructure\_deployment\_layer/monitoring/

**Tasks:**

1. **Comprehensive Logging** - All service logs
2. **Performance Metrics** - Response times, throughput
3. **Error Tracking** - Sentry integration
4. **Usage Analytics** - User behavior tracking

## **📊 DEVELOPMENT WORKFLOW PER FEATURE**

### **🔄 Standard Feature Development Cycle**

1. 📁 CREATE FOLDERS

└── Set up required directory structure

2. 🛠️ BUILD CORE LOGIC

└── Write the actual functionality (no API yet)

3. 🧪 LOCAL TESTING

└── Test with Python scripts/jupyter notebooks

4. 🔌 CREATE API ENDPOINTS

└── Expose functionality via FastAPI

5. 📡 API TESTING

└── Test endpoints with Postman/curl

6. 🔗 INTEGRATE WITH EXISTING

└── Connect new feature to existing codebase

7. ✅ END-TO-END TESTING

└── Test complete workflow works

8. 📝 DOCUMENTATION

└── Update API docs and guides

9. ➡️ NEXT FEATURE

└── Move to next item in roadmap

## **🚨 CRITICAL SUCCESS RULES**

### **✅ DO's**

* **One feature at a time** - Never work on multiple features
* **Core first, API second** - Always build logic before API
* **Test everything** - Each feature must work 100% before next
* **Use local services** - PostgreSQL, QDRANT, Redis locally first
* **Simple before complex** - Basic functionality before optimization

### **❌ DON'Ts**

* **No parallel development** - Focus on one thing only
* **No cloud until Phase 7** - Keep everything local first
* **No optimization early** - Cythonization only in Phase 8
* **No complex deployment** - Docker/NGINX only when ready
* **No UI until Phase 5** - API testing with Postman first

## **📈 SUCCESS METRICS PER PHASE**

| **Phase** | **Success Criteria** | **Test Method** |
| --- | --- | --- |
| **Phase 1** | Single doc → query → answer | Python script test |
| **Phase 2** | All APIs return correct responses | Postman collection |
| **Phase 3** | Data persists across restarts | Database inspection |
| **Phase 4** | Multi-user isolation works | Multiple test accounts |
| **Phase 5** | Complete desktop app works | Manual UI testing |
| **Phase 6** | Installable on 3 OS types | Clean machine installs |
| **Phase 7** | Production cloud deployment | Live user testing |
| **Phase 8** | Performance benchmarks met | Load testing results |

## **🎯 IMMEDIATE NEXT STEPS**

### **Week 1-2: Start Phase 1**

1. **Create core folders** (aiml\_orchestration\_layer, data\_layer)
2. **Set up local QDRANT** (Docker container)
3. **Install OLLAMA** (local LLM)
4. **Build document ingestion** (PDF → chunks → embeddings)
5. **Test basic search** (upload doc → query → get answer)

### **Week 3: Complete Phase 1**

1. **Add Tavily integration** (web search fallback)
2. **Implement confidence scoring**
3. **Test hybrid search** (local + web)
4. **Document the core logic**

**Only move to Phase 2 when Phase 1 is 100% working!**