**Local AI Agent Architecture & Implementation Plan**

**Core Architecture**

**Technology Stack Recommendations**

**Primary Framework:**

* **Electron** or **Tauri** for cross-platform thick client
  + Electron: JavaScript/TypeScript, larger bundle but mature ecosystem
  + Tauri: Rust backend with web frontend, smaller footprint, better security

**AI/ML Backend:**

* **Ollama** for local LLM management (handles model downloads, switching, optimization)
* **LangChain** or **LlamaIndex** for agent orchestration and tool integration
* **ChromaDB** or **Qdrant** for local vector database (document embeddings)
* **Whisper.cpp** for local speech recognition (meeting transcription)

**OS Integration:**

* **Node.js native modules** (if using Electron)
* **Rust system APIs** (if using Tauri)
* **PowerShell/WMI** (Windows), **AppleScript** (macOS), **D-Bus** (Linux)

**System Components**

**1. Core Agent Engine**

┌─────────────────────────────────────┐

│ Agent Core │

├─────────────────────────────────────┤

│ • Task Planning & Execution │

│ • Context Management │

│ • Memory System │

│ • Tool Orchestration │

└─────────────────────────────────────┘

**2. Local AI Stack**

┌─────────────────────────────────────┐

│ Local LLM (Ollama) │

├─────────────────────────────────────┤

│ • Primary reasoning model │

│ • Code generation model │

│ • Embedding model │

└─────────────────────────────────────┘

**3. Knowledge Management**

┌─────────────────────────────────────┐

│ Knowledge Layer │

├─────────────────────────────────────┤

│ • Vector Database (ChromaDB) │

│ • Document Indexing │

│ • Conversation Memory │

│ • User Preferences │

└─────────────────────────────────────┘

**4. OS Integration Layer**

┌─────────────────────────────────────┐

│ System Integration │

├─────────────────────────────────────┤

│ • File System Access │

│ • Application Control │

│ • Calendar/Email Integration │

│ • Screen Capture/Analysis │

└─────────────────────────────────────┘

**5. Internet Gateway**

┌─────────────────────────────────────┐

│ Selective Internet │

├─────────────────────────────────────┤

│ • API Call Manager │

│ • Web Search (when needed) │

│ • Model Updates │

│ • External Integrations │

└─────────────────────────────────────┘

**Recommended Implementation Approach**

**Phase 1: Foundation**

1. **Set up Ollama** with a suitable model (Llama 3.1 8B or Mistral 7B)
2. **Create basic Electron/Tauri app** with system tray integration
3. **Implement basic chat interface** with the local model
4. **Add file system permissions** and basic file operations

**Phase 2: Core Workflows**

1. **Document Search & Analysis**
   * Index local documents using embeddings
   * Implement semantic search
   * Document summarization
2. **Meeting Assistant**
   * Calendar integration
   * Local speech recognition
   * Meeting notes generation
3. **File Management**
   * Intelligent file organization
   * Duplicate detection
   * Content-based search

**Phase 3: Advanced Features**

1. **Learning System**
   * User interaction learning
   * Custom workflow creation
   * Personal data integration
2. **Tool Integration**
   * Custom tool/plugin system
   * API integrations
   * Automation workflows

**Key Technical Considerations**

**Local Model Selection**

* **Llama 3.1 8B**: Good balance of capability and resource usage
* **Mistral 7B**: Efficient, good for reasoning tasks
* **CodeLlama**: Specialized for code-related tasks
* **Embedding Model**: all-MiniLM-L6-v2 or similar for document indexing

**Data Storage Strategy**

~/.local-ai-agent/

├── models/ # Downloaded LLM models

├── knowledge/ # Vector database

├── documents/ # Indexed document cache

├── conversations/ # Chat history

├── workflows/ # Custom workflows

└── config/ # User settings

**Security & Privacy**

* All data remains local by default
* Encrypted storage for sensitive information
* User controls for internet access
* Audit log for external requests
* Sandboxed execution for custom tools

**Resource Management**

* Model switching based on task complexity
* Background indexing with CPU throttling
* Memory management for large document sets
* Efficient embedding caching

**Basic Project Structure**

local-ai-agent/

├── src/

│ ├── main/ # Main process (Electron) or backend (Tauri)

│ │ ├── agent/ # Core agent logic

│ │ ├── models/ # LLM integration

│ │ ├── tools/ # System integration tools

│ │ ├── knowledge/ # Vector DB management

│ │ └── api/ # Internet gateway

│ ├── renderer/ # Frontend UI

│ │ ├── components/ # React/Vue components

│ │ ├── pages/ # Application pages

│ │ └── services/ # Frontend services

│ └── shared/ # Shared utilities

├── resources/ # Static resources

├── scripts/ # Build and setup scripts

└── docs/ # Documentation

**Next Steps**

1. **Choose your stack** (Electron vs Tauri)
2. **Set up Ollama** and test local model integration
3. **Create basic application shell** with system permissions
4. **Implement first workflow** (document search is a good starting point)
5. **Add vector database integration** for knowledge management

Would you like me to create starter code for any specific component, or would you prefer to dive deeper into any particular aspect of this architecture?