Project Design Document: AI-Powered Local File Finder

1. Introduction

In the modern digital workspace, users manage vast amounts of data spread across various local directories and cloud services. Locating specific files efficiently becomes a significant challenge. This project aims to develop a cross-platform, AI-powered file search tool that indexes files from user-selected directories (including subdirectories) and connected cloud services. Utilizing a locally hosted CLIP (Contrastive Language-Image Pre-training) model and a vector database, the application provides intelligent search capabilities based on user queries—all while operating entirely locally and free of charge.

2. Objectives and Goals

- **Local Indexing:** Index every file from user-specified directories and their subdirectories.
- **AI Embedding:** Use a newer CLIP model to generate embeddings for files.
- **Vector Database:** Implement a local vector database to store embeddings for similarity searches.
- **Cost Efficiency:** Ensure all components are free to use, requiring no API keys or subscriptions.
- **Cloud Integration:** Allow users to connect and index files from various cloud services.
- **Real-time Updates:** Implement a file system observer to update the index with any file changes.
- **User Control:** Enable users to select directories for indexing and toggle specific file types.
- **Local Operation:** Ensure all functionalities are executed locally to maintain privacy.
- **Cross-Platform Support:** Focus on macOS design while ensuring Windows compatibility.
- **OCR Capability:** Integrate OCR to extract and search text within images.
- **File Type Filters:** Provide options to include or exclude certain file types in searches.

3. Scope

In-Scope

- Development of a desktop application with a graphical user interface (GUI).
- Local indexing and embedding of files from user-selected directories.
- Integration with cloud services through local sync folders.
- Implementation of a local vector database for storing and searching embeddings.
- Real-time monitoring of file system changes.
- OCR integration for image and scanned PDF files.
- Cross-platform functionality for macOS and Windows.
- User customization options for file types and directories.

Out-of-Scope

- Mobile application development.
- Direct integration with cloud services requiring API keys.
- Web-based or server-hosted versions of the application.
- Advanced natural language processing beyond CLIP capabilities.
- Support for less common or proprietary file formats not widely used.

4. System Architecture

Overview

The system consists of the following key components:

- **User Interface (UI):** Facilitates user interactions for directory selection, search queries, and settings adjustments.
- **File Indexer:** Scans and indexes files from selected directories and cloud services.
- **Embedding Engine:** Generates embeddings for files using a locally hosted CLIP model.
- **Vector Database:** Stores file embeddings and enables efficient similarity searches.
- **File System Observer:** Monitors directories for changes and updates the index accordingly.
- **OCR Module:** Extracts text from images and scanned PDFs to enhance searchability.
- **Search Engine:** Processes user queries and retrieves similar files based on embeddings.
- **Cloud Sync Integration:** Incorporates files from cloud services via local sync folders.

Data Flow Diagram Description

1. **User Selection:** Users select directories and configure settings via the UI.
2. **File Indexing:** The File Indexer scans selected directories and gathers file data.
3. **Embedding Generation:** The Embedding Engine creates embeddings for each file.
4. **Storage:** Embeddings are stored in the local Vector Database.
5. **Monitoring:** The File System Observer detects any file changes and triggers re-indexing.

6. **Search Query:** Users input search queries, which are embedded by the Embedding Engine.
7. **Similarity Search:** The Search Engine retrieves files similar to the query from the Vector Database.
8. **Results Display:** Search results are presented to the user through the UI.

---

5. Detailed Design

5.1. File Indexing and Embedding

- **File Indexer:**
  - Recursively scans user-selected directories and subdirectories.
  - Collects metadata such as file name, type, size, and modification date.
  - Respects user-defined file type filters to include or exclude specific formats.
- **Embedding Engine:**
  - Utilizes a locally hosted CLIP model (e.g., OpenAI's CLIP or a similar open-source model).
  - For text-based files, extracts text content for embedding.
  - For images, directly embeds image data and optionally includes OCR-extracted text.
  - Handles other file types using metadata and file names for embedding.
- **Model Deployment:**
  - Downloads and stores the CLIP model locally to avoid external dependencies.
  - Ensures the model is compatible with local hardware constraints.

5.2. Vector Database

- **Database Selection:**
  - Employs FAISS (Facebook AI Similarity Search), an open-source library for efficient similarity search.
  - Stores embeddings in a local database optimized for quick retrieval.
- **Data Management:**
  - Supports incremental updates as new files are added or removed.
  - Handles large datasets by efficiently managing memory and storage.
- **Search Algorithm:**
  - Utilizes cosine similarity for matching query embeddings with stored embeddings.
  - Implements approximate nearest neighbor search for scalability.

5.3. Directory and Cloud Service Integration

- **Directory Selection:**
  - Provides a UI component for users to select and manage directories to index.
  - Allows inclusion or exclusion of subdirectories.
- **Cloud Services:**
  - Integrates with cloud services like Google Drive and Dropbox through their local sync folders.
  - Avoids the need for API keys by leveraging files synced locally on the user's machine.
  - Updates the index as cloud files are updated locally.

5.4. File System Observer

- **Functionality:**
  - Monitors selected directories for file additions, deletions, and modifications.
  - Triggers re-indexing and updates embeddings in real-time.
- **Implementation:**
  - On macOS, uses the `FSEvents` API for efficient file system monitoring.
  - On Windows, utilizes the `ReadDirectoryChangesW` API.
- **Performance:**
  - Designed to have minimal impact on system resources.
  - Handles events asynchronously to maintain application responsiveness.

5.5. User Interface

- **Framework:**
  - Developed using Electron.js for cross-platform compatibility.
  - Ensures a native look and feel on macOS and Windows.
- **Components:**

- **Dashboard:** Displays indexing status and recent activity.
- **Directory Management:** Allows adding/removing directories and viewing indexing progress.
- **Search Bar:** Provides a simple interface for entering search queries.
- **Results View:** Presents search results with file previews and metadata.
- **Settings:** Includes options for OCR, file type filters, and cloud service management.

- **Design Principles:**
  - Follows macOS Human Interface Guidelines for aesthetics and usability.
  - Prioritizes simplicity and ease of use.

5.6. OCR Implementation

- **Technology:**
  - Integrates Tesseract OCR, an open-source OCR engine, for text extraction from images and PDFs.
- **Operation:**
  - Processes images and scanned PDFs during indexing if OCR is enabled.
  - Extracted text is included in the embedding process to improve search relevance.
- **User Control:**
  - Provides a toggle in the settings to enable or disable OCR processing.
  - Allows users to limit OCR to specific directories or file types to optimize performance.

5.7. File Type Filtering

- **Customization:**
  - Offers checkboxes for users to select which file types to include in the index.
  - Common categories include documents, images, audio, video, and archives.
- **Dynamic Indexing:**
  - Changes to file type selections prompt the application to update the index accordingly.
  - Ensures that searches return results only from selected file types.

---

6. User Interface Design

Key Screens and Elements

- **Home Screen:**
  - Provides an overview of the application's status and recent indexing activity.
  - Includes quick access buttons for common actions.
- **Directory Selection Screen:**
  - Displays a list of currently indexed directories.
  - Allows users to add new directories or remove existing ones.
- **Search Interface:**
  - Features a prominent search bar.
  - Includes options for advanced search filters (file types, date ranges).
- **Settings Menu:**
  - Contains toggles for OCR, file type filters, and cloud service options.
  - Provides access to help documentation and application information.
- **Results Display:**
  - Shows search results with thumbnails, file names, and brief previews.
  - Supports sorting and filtering of results.

Design Aesthetics

- **macOS Focused Design:**
  - Utilizes native macOS UI elements and icons.
  - Ensures consistency with the operating system's look and feel.
- **Accessibility:**
  - Supports keyboard navigation and screen readers.
  - Adheres to accessibility standards for color contrast and text size.

---

7. Implementation Plan

Phase 1: Planning and Setup (Week 1)

- **Tasks:**
  - Define project milestones and deliverables.
  - Set up version control and development environments.
  - Assign roles:
    - **Member 1:** UI/UX Design and Development
    - **Member 2:** File Indexing and Embedding Engine
    - **Member 3:** Vector Database and Search Engine

Phase 2: Core Development (Weeks 2-6)

- **File Indexing and Embedding:**
  - Implement recursive directory scanning.
  - Integrate the CLIP model for embedding generation.
- **Vector Database Setup:**
  - Install and configure FAISS.
  - Develop functions for storing and retrieving embeddings.
- **Search Functionality:**
  - Create algorithms for similarity search.
  - Develop the search engine to process user queries.

Phase 3: UI Development (Weeks 7-9)

- **UI Design:**
  - Create wireframes and prototypes.
  - Develop the UI using Electron.js.
- **Backend Integration:**
  - Connect UI elements with backend functionalities.
  - Ensure real-time updates and responsiveness.

Phase 4: Feature Integration (Weeks 10-12)

- **OCR Integration:**
  - Incorporate Tesseract OCR into the indexing process.
  - Add OCR settings to the UI.
- **File System Observer:**
  - Implement real-time monitoring for both macOS and Windows.
  - Test responsiveness to file system changes.
- **Cloud Services:**
  - Enable indexing of cloud-synced directories.
  - Provide UI options for managing cloud directories.

Phase 5: Testing and Optimization (Weeks 13-14)

- **Testing:**
  - Conduct unit, integration, and performance tests.
  - Perform cross-platform testing on macOS and Windows.
- **Optimization:**
  - Improve indexing and search performance.
  - Optimize resource usage during OCR processing.

Phase 6: Deployment Preparation (Week 15)

- **Packaging:**
  - Create installers for macOS (DMG) and Windows (MSI).
  - Ensure all dependencies are included.
- **Documentation:**
  - Prepare user guides and installation instructions.
  - Document code for future maintenance.

8. Testing and Quality Assurance

Testing Strategies

- **Unit Testing:**
  - Test individual modules like the File Indexer and Embedding Engine.
- **Integration Testing:**
  - Ensure that the UI correctly interacts with backend components.
- **Performance Testing:**
  - Assess indexing and search speeds with varying dataset sizes.
- **Cross-Platform Testing:**
  - Verify that all features work seamlessly on both macOS and Windows.
- **User Acceptance Testing:**
  - Gather feedback from beta users to identify usability issues.

Quality Assurance

- **Code Reviews:**
  - Regular peer reviews to maintain code quality.
- **Issue Tracking:**
  - Use a project management tool to track bugs and feature requests.

---

9. Deployment Plan

Packaging and Distribution

- **Installers:**
  - Use `electron-builder` to create platform-specific installers.
  - Ensure that installers handle dependency installation.
- **Distribution Channels:**
  - Release the application on GitHub with proper versioning.
  - Provide checksums and signatures for security.

User Support

- **Documentation:**
  - Include a README and a detailed user manual.
  - Provide FAQs and troubleshooting guides.
- **Feedback Mechanism:**
  - Incorporate a feedback form within the application.
  - Monitor user reports to improve future versions.

---

10. Future Enhancements

- **Additional Cloud Services:**
  - Direct API integration with more cloud services if API keys become viable.
- **Advanced Search Features:**
  - Implement natural language processing for more intuitive queries.
- **Machine Learning Improvements:**
  - Train custom models to better suit user-specific data.
- **Plugin Support:**
  - Allow third-party developers to extend functionality through plugins.
- **Mobile Application:**

- Develop companion apps for mobile devices.

Conclusion

This project aims to deliver a powerful, AI-driven file search tool that operates entirely locally, ensuring user privacy and data security. By leveraging free, open-source technologies and focusing on user-centric design, the application will provide an efficient solution to the challenge of managing and locating files across multiple directories and cloud services.

--