Stage 1. Idea Exploration & Planning (deadline 2/23/2025)

Define Goals & Objectives: Clearly outline the problem the application aims to solve.
Market Research & Feasibility Analysis: Identify competitors, market needs, and potential customers. Explore existing solutions.
Risk Assessment: Identify technical, financial, and security risks.

1. Identify the Purpose & Goals

- Provide a **local** image similarity finder with text-based and image-based search.
- Make the UI **user-friendly, visually appealing, and marketable**.
- Ensure **fast and accurate** search using pre-indexed vector storage.
- Deliver a **web-like experience** without using a web framework.
- Keep the **app lightweight, simple, and efficient**.

2. Define Key Features & Functionalities

✅ Search Functionalities

- Text/context search (e.g., "hand holding a coconut").
- Image-based search (upload an image to find similar ones).

✅ Preprocessing & Indexing

- Use a **Vision-Language Model (VLM)** to generate image descriptions.
- Store embeddings in a **vector database** (e.g., FAISS, ChromaDB).

✅ Filters & Refinements

- **Folder filtering** – Search within specific directories.
- **Similarity threshold** – Adjust match sensitivity.

✅ User Interface (UI/UX)

- **Modern, web-like experience** without a web framework.
- **Sidebar navigation** for search options.
- **Thumbnail grid** for displaying results.
- **Basic image viewer** with preview capabilities.
- **Drag-and-drop** support for image uploads.

✅ Performance & Optimization

- **Pre-indexed search** for fast retrieval.
- Lightweight and **offline-first** operation.

3. Identify Target Audience & Use Cases

🎯 Who will use it?

- Designers, photographers, and researchers looking to search for images quickly.
- Casual users managing large image collections.

📌 Use Cases

- Searching for images using a descriptive phrase.
- Finding duplicate or similar images in a directory.
- Organizing image collections efficiently.

4. Define Technology Stack

Frontend (GUI):

- **PyQt (QML)** – For a web-like, modern desktop UI.
- **Dear PyGui** – If you want GPU acceleration and a snappier feel.

Backend:

- **Python** (Full-stack development).
- **Vision-Language Model (CLIP, BLIP, etc.)** for text-based image search.
- **Vector Database (FAISS, ChromaDB, or Pinecone)** for fast retrieval.

5. Define Constraints & Challenges

⚠ Potential Challenges

- Optimizing search speed for large image datasets.
- Handling memory efficiently when dealing with large image collections.
- Ensuring UI remains lightweight and smooth.

🔍 Constraints

- Must run **locally**, not cloud-based.
- Should work on **Windows, macOS, and Linux** if possible.

6. Define Success Criteria

✅ The app should be **able to find images accurately** using text descriptions.
✅ The UI should be **simple yet polished**, with an intuitive experience.
✅ The search process should be **fast and efficient** for thousands of images.
✅ The app should be **lightweight and self-contained** for easy distribution.

Stage 2. Requirements Gathering & Analysis (deadline 2/23/2025)

Functional Requirements: Define core functionalities and features, user interactions, business rules, etc.
Non-Functional Requirements: Performance, Security, Scalability, Accessibility, Compliance.
Use Cases.

Stage 2: Requirements Gathering & Analysis

📅 **Deadline: 02/23/2025**

---

1. Functional Requirements

These define the core functionalities, features, user interactions, and business rules of the app.

Core Functionalities & Features

✅ **Text-Based Search (Context Search)** – Users can search for images using natural language descriptions (e.g., *"a dog sitting on a couch"*).
✅ **Image-Based Search** – Users can upload an image to find visually similar ones.
✅ **Pre-Indexed Search** – The app will preprocess images using a **Vision-Language Model (VLM)** and store embeddings in a **vector database** for fast retrieval.
✅ **Folder Filtering** – Users can limit searches to specific folders or directories.
✅ **Similarity Threshold Adjustment** – Users can choose how similar the search results are.
✅ **Basic Image Viewer** – Clicking an image result opens a preview with file details (e.g., path, date).

User Interactions

👤 **User Actions**:

- Upload an image or enter a text query.
- View and refine search results.
- Filter by folder and adjust similarity settings.
- Click on an image for a quick preview.
- Open the image location in the file system.

🎯 **Business Rules**:

- The app should work **entirely offline** (no cloud processing).
- Users should be able to **index large image libraries** without performance degradation.
- When indexing pictures, it could be long . However, when searching, it should return results within **a few seconds**, even for large datasets.

---

2. Non-Functional Requirements

These define performance, security, scalability, and compliance aspects of the app.

Performance

⚡ **Fast Search Speed** – Search results should be retrieved within 1-2 seconds.
📂 **Optimized Storage** – Pre-indexed data should be stored efficiently to avoid excessive disk usage.
🚀 **Low Resource Consumption** – The app should run smoothly on mid-range PCs without high CPU/GPU usage.

Security

🔒 **Privacy-Focused** – No images or user data should be sent to external servers.
🛠️ **Local-Only Processing** – All indexing and searching must be done on the user's machine.

Scalability

📈 **Supports Large Image Collections** – The app should be able to index **at least 100,000+(lol) images** efficiently.
🔄 **Incremental Indexing** – Users should be able to update the index database without reprocessing all images.

Accessibility

🖥️ **Cross-Platform Support** – Should work on **Windows first then macOS, and Linux**.
🎨 **User-Friendly UI** – The interface should be intuitive and visually appealing.

Compliance

📜 **Licensing & Open-Source Considerations** – If using third-party models (e.g., CLIP, BLIP), need to ensure proper licensing compliance.

---

3. Use Cases

Use Case 1: Searching for an Image Using Text

Actors: User
Description: A user enters a text description (e.g., *“beach sunset with palm trees”*), and the app returns matching images from the indexed library.
Preconditions: The image library must be pre-indexed.
Postconditions: The user can view, refine, or open the found images.

Use Case 2: Searching for an Image Using Another Image

Actors: User
Description: A user uploads an image, and the app retrieves visually similar images.
Preconditions: The database must be pre-indexed.
Postconditions: The user can view, refine, or open the results.

Use Case 3: Filtering Search by Folder

Actors: User
Description: The user selects specific folders to limit the search scope.
Preconditions: The app must have access to the selected directories.
Postconditions: The search results update to reflect the filtered folders.

Use Case 4: Adjusting Similarity Threshold

Actors: User
Description: The user adjusts the similarity threshold to make search results more or less strict.
Preconditions: Indexed images must exist.
Postconditions: Search results update dynamically based on the threshold.

Use Case 5: Viewing Image Details & Location

Actors: User
Description: The user clicks on an image result to preview it and see file details.
Preconditions: The image must exist on the local system.
Postconditions: The preview window displays the image with metadata, and the user can open the file location.

Stage 3. System Design, Architecture & UI/UX Design (deadline 3/2/2025)

High-Level Architectural Planning
Architecture Style: Decide between monolithic, microservices, or serverless based on project scale and complexity.
Tech Stack Selection: Choose appropriate frameworks and languages for the frontend (e.g., React, Angular) and backend (e.g., Node.js, Django, Spring Boot), as well as database systems (SQL, NoSQL).
Integration Strategy: Plan how the new system will integrate with existing infrastructure or third-party services.

API & Data Management
API Design: Outline API structure (REST vs. GraphQL), versioning strategy, and documentation practices (Swagger, Postman).
Database Design: Develop initial data models, consider indexing strategies, partitioning/sharding, and backup/disaster recovery plans.

UI/UX Design
Wireframes & Prototyping (Figma)
Style Guides: Define color schemes, typography, and UI components to ensure a consistent look and feel.
Component Libraries: Consider using or building a reusable component library to speed up development and maintain uniformity.
Accessibility Standards: Ensure designs comply with WCAG guidelines for accessibility.
Responsive Layouts: Design for various screen sizes and devices, emphasizing mobile-first design principles.