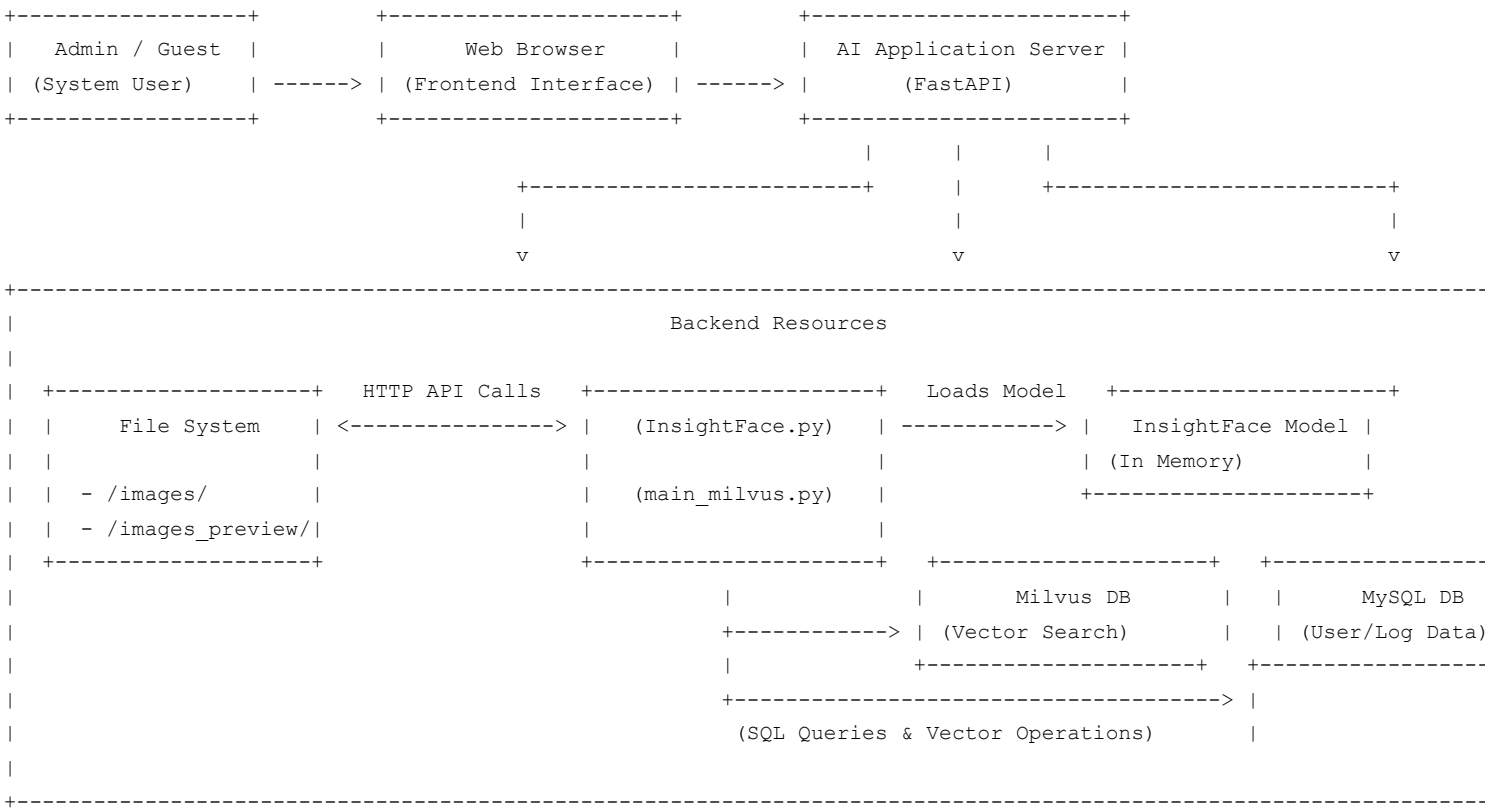
FaceSearch AI: System Architecture & Workflow

1. High-Level Architecture Diagram

This diagram shows the main components of the system and how they interact with each other.



2. Component Breakdown

Component	Technology	Responsibility
Frontend	HTML, CSS, JavaScript	Provides the user interface for both guests and admins. It runs entirely in the user's web browser and communicates with the backend via API calls.
AI Application Server	Python, FastAPI	The central "brain" of the system. It handles all incoming requests, manages user authentication, processes business logic, and orchestrates all interactions with the databases and the AI model.
InsightFace Model	OnnxRuntime	A pre-trained AI model loaded into memory by the FastAPI server. Its only job is to analyze an image, detect faces, and convert each face into a unique mathematical signature called a **vector** or **embedding**.
Milvus Database	Milvus (Docker)	A specialized AI database designed for one purpose: to store millions of face vectors and perform incredibly fast similarity searches. When given a new face vector, it can find the most similar ones in the database almost instantly.
MySQL Database	MySQL	A traditional relational database that stores all structured data: guest accounts, admin users, activity logs, payment records, and metadata about photo collections (like names and creation dates).
File System	Server's Hard Drive	The physical storage location for the actual image files. This is split into two critical folders: `/images` (for originals) and `/images_preview` (for watermarked thumbnails).

---

3. How It Works: Step-by-Step Workflows

Here are the two most important processes in the system.

Workflow 1: Admin Indexes a New Day's Photos

This is the process of adding new photos to make them searchable.

1. **Preparation (Manual):** An admin places a new folder of high-resolution photos onto the server inside the `/images` directory (e.g., `/images/park_day_oct_28`).

2. **Initiation (Admin UI):** The admin logs into the dashboard, clicks **"New Collection,"** selects the new folder, gives the collection a name (e.g., `park_day_oct_28`), and clicks **"Start Indexing."**
3. **API Request:** The browser sends an API request to the **FastAPI Server**.
4. **Image Processing Loop:** The server begins a loop, processing each image from the specified folder one by one. For each image, it performs two main tasks:
    - **Face Vectorization:**
        - The image is loaded and passed to the **InsightFace Model**.
        - The model detects all faces in the image and calculates a 512-dimension **vector** for each one.
    - **Preview Image Creation:**
        - A low-resolution, watermarked version of the image is created.
        - This "preview" is saved to the `/images_preview` folder on the **File System**.
5. **Storing AI Data:** The server sends the face vectors to the **Milvus Database**. It tells Milvus, "Store these vectors in the `park_day_oct_28` collection, and remember that they came from `image_path_abc.jpg`."
6. **Storing Metadata:** The server sends a query to the **MySQL Database** to log the creation of the new collection, including its name, source folder, and timestamp.
7. **Confirmation:** Once the loop is finished, the server sends a success message back to the admin's browser. The photos are now live and searchable.

Workflow 2: Guest Searches for Their Photos

This is the core user-facing function of the application.

1. **Upload (Guest UI):** A guest logs in, selects the date of their visit from the collection dropdown (e.g., `park_day_oct_28`), and uploads a clear picture of their face.
2. **API Request:** The browser sends the uploaded image and the selected collection name to the **FastAPI Server**.
3. **Query Face Vectorization:** The server takes the uploaded image and uses the **InsightFace Model** to generate its face **vector**. This is the "query vector."
4. **Vector Search:** The server sends this query vector to the **Milvus Database** and issues a command: "Find the top 100 vectors in the `park_day_oct_28` collection that are most similar to this query vector."
5. **Receiving Results: Milvus** instantly performs the search and returns a list of matching results. Each result contains the original `image_path` and a `distance` score (lower is better).
6. **Processing and Filtering:** The server filters these results, discarding any matches that are not close enough (i.e., the distance score is too high).
7. **Preparing Response:** For each valid match, the server constructs the URL for its corresponding **preview image** (e.g., `/images_preview/photo_xyz.jpg`). **It never sends the path to the original, high-res image.**
8. **Final Response:** The server sends a clean JSON list of the preview image URLs back to the guest's browser.
9. **Displaying Results:** The browser's JavaScript code receives the list and dynamically builds the gallery of watermarked photos for the guest to view, select, and purchase.