Microservice Requirement Document: Reverse Image Search

Version: 1.0
Date: 22-08-2025
Author: Aditya Pal

1. Introduction & Purpose

This document outlines the requirements for the **Reverse Image Search Microservice**. This service is a key investigative component of the OSINT application, enabling law enforcement to find the online presence and origin of a specific image.

The primary purpose of this microservice is to accept an image file and perform a comprehensive reverse image search across multiple search engines. It will then aggregate the results, identify visually similar images, and return a list of URLs where the image or similar ones appear. The service will be built using **Python** and the **FastAPI** framework.

2. Functional Requirements

2.1. Supported Input

The service must accept an image as input in one of the following formats:

- **Image File Upload:** A direct multipart/form-data upload of an image file. Supported formats are JPEG, PNG, and WEBP.
- **Image URL:** A publicly accessible URL pointing to an image.

2.2. Target Search Engines

To ensure comprehensive results, the microservice must perform the reverse image search across the following platforms simultaneously:

- Google Images
- Yandex Images
- Bing Images
- PimEyes (for facial recognition-focused searches)

2.3. Internal Database Integration

As a preliminary step, the service must compare the input image against an internal hash database of previously processed images from other investigations.

- **Hashing:** Upon receiving an image, the service will generate a perceptual hash (pHash).
- **Internal Check:** It will first query an internal database (e.g., Elasticsearch or a dedicated vector database) to see if an identical or visually similar image (based on hash distance)

has been seen before. If a match is found, it should be flagged in the results.

2.4. Required Output Data Points

The service must aggregate and deduplicate the results from all search engines and return a structured JSON object.

- match_type: Indicates the type of match (e.g., "exact_match", "visually_similar", "partial_match").
- page_url: The URL of the webpage where the matching image was found.
- image_url: The direct URL to the matching image file.
- page_title: The title of the webpage.
- snippet: A brief text snippet from the page, providing context for the image's use.
- source_engine: The search engine that found the result (e.g., "Google Images", "Yandex").

3. API Endpoints Specification

The service will follow the same asynchronous task-based model as other modules.

3.1. POST /search

Submits an image for reverse searching.

- **Method:** POST
- **Description:** Initiates a reverse image search task. The request body must be multipart/form-data to handle file uploads.
- **Request Body Fields:**
  - image_file: The image file (optional if image_url is provided).
  - image_url: The URL of the image (optional if image_file is provided).
- Success Response (202 - Accepted):
  Returns a task_id for polling results.
  ```
  {
    "message": "Reverse image search task accepted.",
    "task_id": "c3d4e5f6-a7b8-9012-3456-7890abcdef12"
  }
  ```

3.2. GET /results/{task_id}

Checks the status and retrieves the results of a search job.

- **Method:** GET
- **Description:** Retrieves the result of a previously submitted task.
- **Success Response (200 - OK):**
  - **If Pending:**
    ```
    {
      "task_id": "c3d4e5f6-a7b8-9012-3456-7890abcdef12",
    ```

```
  "status": "PENDING",
  "data": null
}
```


- **If Complete:**


```
{
  "task_id": "c3d4e5f6-a7b8-9012-3456-7890abcdef12",
  "status": "COMPLETE",
  "data": [
    {
      "match_type": "exact_match",
      "page_url": "https://example.com/profile/johndoe",
      "image_url": "https://example.com/images/johndoe.jpg",
      "page_title": "John Doe - About Me",
      "snippet": "John Doe is a software engineer based in...",
      "source_engine": "Google Images"
    },
    {
      "match_type": "visually_similar",
      "page_url": "https://anotherexample.net/article/123",
      "image_url": "https://anotherexample.net/img/person.png",
      "page_title": "Tech Conference 2024",
      "snippet": "The conference featured many speakers including...",
      "source_engine": "Yandex"
    }
  ]
}
```


- Error Response (404 - Not Found):
  If the task_id is invalid.

4. Non-Functional Requirements

- **Security:**
  - API must be secured with an API key (X-API-KEY header).
  - All uploaded files must be scanned for malware before processing.
  - Temporary image files must be deleted immediately after processing is complete.
- **Performance:**
  - The POST /search endpoint must respond in under **500ms** (allowing for file upload).
  - A complete search across all engines should ideally complete in under 90 seconds.
- **Scalability:**
  - The service must be containerized (Docker).
  - It should be able to process multiple image searches concurrently, using a worker pool to manage the scraping tasks.

- **Logging:**
  - Log every search request, including a hash of the image and the final count of results from each engine.
  - Error logs must be detailed, especially for failures in interacting with external search engine APIs or websites.
- **Data Handling:**
  - The service must handle various image sizes but should enforce a reasonable maximum file size limit (e.g., 10MB) to prevent abuse.
  - Images should be resized/recompressed to a standard format before being submitted to external search engines to optimize performance.

5. Technology Stack

- **Language:** Python 3.9+
- **Framework:** FastAPI
- **Asynchronous Tasks:** Celery with Redis or RabbitMQ
- **Image Processing:** Pillow or OpenCV
- **Image Hashing:** ImageHash (for pHash)
- **Web Scraping/Automation:** Playwright or Selenium to interact with search engine interfaces.
- **Deployment:** Docker