Microservice Requirement Document: Social Media Scraping

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

1. Introduction & Purpose

This document outlines the functional and non-functional requirements for the **Social Media Scraping Microservice**. This service is a core component of a larger OSINT application designed for Indian law enforcement.

The primary purpose of this microservice is to accept various identifiers (like a phone number, email, or username) and, in response, scrape publicly available information from specified social media platforms to uncover associated profiles and related data. The service will be built using **Python** with the **FastAPI** framework.

2. Functional Requirements

2.1. Supported Input Identifiers

The service must be able to initiate a scrape based on the following input types:

- **Phone Number:** A valid 10-digit Indian mobile number.
- **Email Address:** A valid email address.
- **User ID:** A username or handle from a social media platform.

2.2. Target Social Media Platforms

The microservice must be capable of scraping data from the following platforms:

- Facebook
- Instagram
- X (formerly Twitter)
- LinkedIn
- Telegram (Public Channels/Groups)
- Koo

2.3. Scraping Logic & Data Extraction

The service should intelligently use the input identifiers to find associated profiles.

- **Given a Phone Number:**
    - Check "Forgot Password" or account recovery flows on platforms to see if an account is linked to the number.

    - Search for the number directly on platforms where it might be public (e.g., older Facebook profiles).
    - Identify potential UPI IDs and use the associated names to search for profiles.
- **Given an Email Address:**
    - Check "Forgot Password" or account recovery flows to confirm account existence.
    - Search for the email address on platforms where it might be publicly visible.
- **Given a User ID:**
    - Directly navigate to the profile on the corresponding platform.
    - Use the User ID to search on other platforms, as users often reuse handles.

2.4. Required Output Data Points

When a profile is successfully found, the service must extract the following publicly available information and return it in a structured JSON format. All fields should be included, with null values if the data is not found.

- platform: The name of the social media platform (e.g., "Facebook").
- profile_url: The direct URL to the user's profile.
- username: The user's handle/ID on the platform.
- full_name: The user's displayed full name.
- profile_picture_url: A direct link to the user's current profile picture.
- bio_description: The text content of the user's bio.
- location: Any location information mentioned in the profile.
- follower_count: Number of followers.
- following_count: Number of accounts the user is following.
- external_links: Any websites or links listed in the bio/profile.
- recent_posts: A list of the 5 most recent public posts, including text content, media URLs, and post timestamp.

3. API Endpoints Specification

The service will expose a single primary endpoint for initiating scrapes.

3.1. POST /scrape

This endpoint initiates the scraping process. It should handle the task asynchronously to avoid blocking the client.

- **Method:** POST
- **Description:** Submits a scraping job based on a given identifier.
- **Request Body:**
```
{
  "identifier_type": "phone_number" | "email" | "user_id",
  "identifier_value": "string",
  "platforms": ["facebook", "instagram", "twitter"]
}
```

- Success Response (202 - Accepted):
  The service should immediately accept the request and return a unique task_id that the client can use to poll for results.
  ```
  {
    "message": "Scraping task accepted.",
    "task_id": "a1b2c3d4-e5f6-7890-1234-567890abcdef"
  }
  ```

3.2. GET /results/{task_id}

This endpoint is used to check the status and retrieve the results of a scraping job.

- **Method:** GET
- **Description:** Retrieves the result of a previously submitted scraping task.
- **Success Response (200 - OK):**
  - **If Pending:**
    ```
    {
      "task_id": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
      "status": "PENDING",
      "data": null
    }
    ```
  - **If Complete:**
    ```
    {
      "task_id": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
      "status": "COMPLETE",
      "data": [
        {
          "platform": "Facebook",
          "profile_url": "https://facebook.com/example.user",
          "username": "example.user",
          "full_name": "John Doe",
          // ... other data points
        },
        {
          "platform": "Twitter",
          "profile_url": "https://twitter.com/exampleuser",
          "username": "exampleuser",
          "full_name": "John Doe",
          // ... other data points
        }
      ]
    }
    ```

- Error Response (404 - Not Found):
  If the task_id is invalid.
  ```
  {
    "detail": "Task not found."
  }
  ```

4. Non-Functional Requirements

- **Security:**
  - The API must be secured with an API key (X-API-KEY header).
  - All input must be sanitized to prevent injection attacks.
  - The service must employ robust proxy management (IP rotation) to avoid being blocked by social media platforms.
- **Performance:**
  - The POST /scrape endpoint must respond in under **200ms**.
  - A full scrape across all platforms should be completed within a reasonable timeframe (target: under 2 minutes).
- **Scalability:**
  - The service should be containerized (Docker) for easy deployment and scaling.
  - It must be able to handle at least 50 concurrent scraping requests.
- **Logging:**
  - Comprehensive logging is required. Logs must include the timestamp, input identifier (masked for privacy), platforms scraped, and the outcome (success/failure).
  - Error logging must be detailed enough to diagnose scraping failures (e.g., HTML structure changes, rate limiting).
- **Error Handling:**
  - The service must gracefully handle common scraping issues like CAPTCHAs, rate limits, and profile-not-found errors.
  - The final result should clearly indicate which platforms failed to yield results and why.

5. Technology Stack

- **Language:** Python 3.9+
- **Framework:** FastAPI
- **Asynchronous Tasks:** Celery with Redis or RabbitMQ
- **HTTP Requests:** httpx (for async support)
- **Web Scraping:** BeautifulSoup4, Scrapy, or a browser automation tool like Playwright (for JavaScript-heavy sites).
- **Deployment:** Docker