**Installation**

Clone the repository:

git clone <repository\_url>

cd <repository\_name>

Install dependencies:

pip install -r requirements.txt

**Usage**

**Running the Flask Application**

Run the Flask application:

flask run

A link will be provided in the terminal (e.g., Running on http://127.0.0.1:5000). Follow that link using ctrl+click.

**API Endpoints**

**1. Upload Image**

* **Endpoint:** /upload
* **Method:** POST

**Example Request:**

curl -X POST -F "pic=@/path/to/image.jpg" http://localhost:5000/upload

**Explanation:**

* This endpoint allows users to upload an image file to the server.
* The -F flag with curl is used to specify a form field. In this case, pic is the form field name, and @/path/to/image.jpg represents the path to the image file to be uploaded.
* Upon successful upload, the server responds with a 200 OK status code and a message indicating the successful upload.
* If no image is uploaded or the filename is invalid, the server responds with a 400 Bad Request status code and an appropriate error message.
* If the image already exists in the database, the server responds with a 400 Bad Request status code and a message indicating that the image already exists.

**2. Display All Images**

* **Endpoint:** /images
* **Method:** GET

**Example Request:**

curl http://localhost:5000/images

**Explanation:**

* This endpoint retrieves all uploaded images from the server.
* A GET request is sent to the /images endpoint to retrieve the list of images.
* The server responds with an HTML page containing the images along with their descriptions (if available).
* This HTML page can be viewed in a web browser.

**3. Get Individual Image**

* **Endpoint:** /<int:id>
* **Method:** GET

**Example Request:**

curl http://localhost:5000/1

**Explanation:**

* This endpoint retrieves an individual image by its unique identifier (id) from the server.
* A GET request is sent to the endpoint with the specific image ID (e.g., 1).
* If the image with the specified ID exists, the server responds with the image data along with the correct MIME type.
* The client can then display or process the image data accordingly.

**Assumptions and design decisions made during the development of this application:**

1. **Image File Formats**: The application assumes that users will upload images in formats such as PNG, JPG, JPEG, or GIF. Other formats are not supported.
2. **SQLite Database Usage**: The application uses SQLite as its database system. This decision was made due to its simplicity and ease of integration with Flask applications. However, for larger-scale applications with higher concurrency requirements, a more robust database system like PostgreSQL or MySQL might be preferred.
3. **Image Upload Directory**: Uploaded images are temporarily stored in the ./uploads/ directory on the server. This directory is created if it does not exist. However, it's assumed that this directory is writable by the application for storing uploaded images.
4. **Error Handling**: The application includes basic error handling for various scenarios such as missing files, invalid file formats, and database-related errors. However, it's assumed that more robust error handling and logging mechanisms may be implemented for production-grade applications.
5. **Hugging Face Transformers Library**: The application utilizes the Hugging Face transformers library for implementing the image-to-text pipeline. This library provides pre-trained models for various natural language processing tasks, including image captioning. However, it's assumed that users have installed the required dependencies and have access to a reliable internet connection for model downloading.
6. **Flask Application Structure**: The application follows a basic Flask application structure with separate files for routes (app.py), database initialization (db.py), and database models (models.py). This structure provides a clear separation of concerns and facilitates maintainability and scalability.