**Image Processing System Documentation**

**Overview**

The Image Processing System allows users to upload CSV files containing image URLs. The system processes these images by compressing them and then provides various functionalities to check the processing status and download the results.

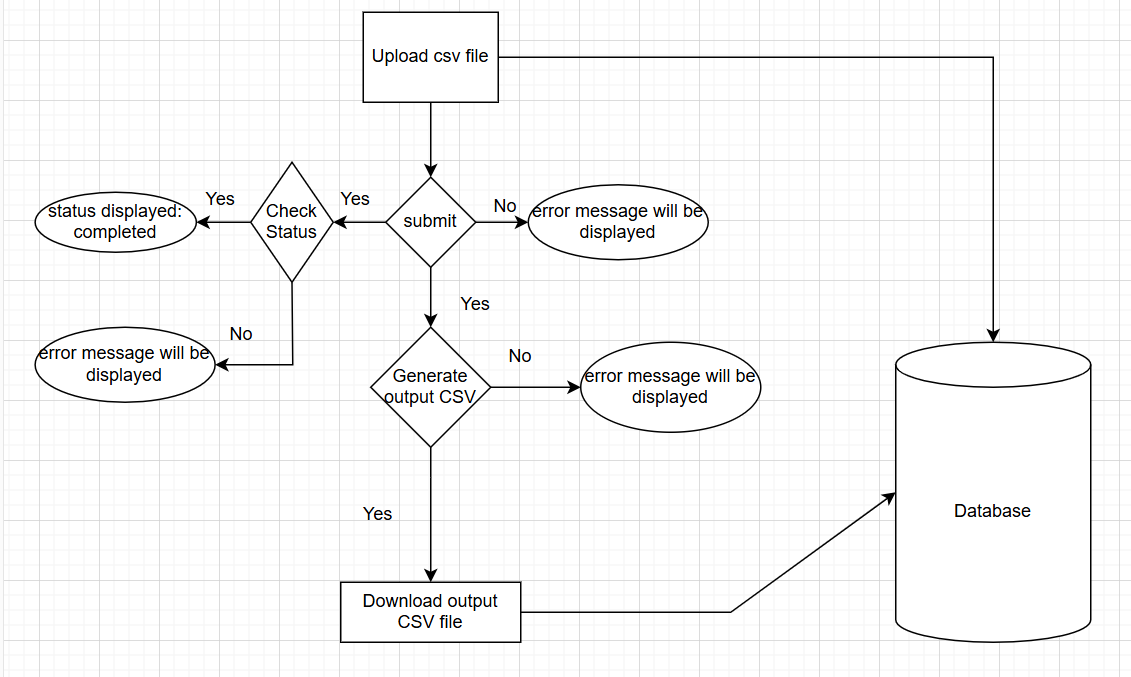
**Table of Contents**

1. [System Requirements](#system-requirements)
2. [Setup Instructions](#setup-instructions)
3. [Folder Structure](#folder-structure)
4. [Backend Code](#backend-code)
5. [Frontend Code](#frontend-code)
6. [API Endpoints](#api-endpoints)
7. [Database Schema](#database-schema)
8. [How to Use](#how-to-use)

**System Requirements**

* Python 3.7 or higher
* Flask
* SQLAlchemy
* MySQL Connector
* aiohttp
* Pillow
* pandas
* A MySQL database

**Architecture Diagram**

****

**Application Workflow**

1. Introduction

The system is designed to process images listed in a CSV file. Users upload a CSV file containing URLs of images, which are then processed (e.g., compressed), and the results are made available for download. The system provides functionalities to check the processing status and download the results.

2. User Workflow

Upload CSV File: Users upload a CSV file through the web interface. The CSV file should contain columns for serial numbers, product names, and image URLs.

Processing: Once the file is uploaded, the backend processes the images listed in the CSV file. It performs operations such as downloading, compressing, and storing the images.

Check Status: Users can check the status of their processing request using a unique request ID provided after the upload.

Generate Output CSV: After processing, users can generate and download an output CSV file that includes the processed image URLs.

3. Detailed Workflow

* File Upload

The user selects a CSV file and clicks the "Submit" button.

The file is sent to the /submit\_file endpoint.

The server saves the file and assigns a unique request ID.

A new entry is created in the ProcessingRequest table with the status set to "Pending."

* Image Processing

Asynchronously, the server reads the CSV file and validates its contents.

For each image URL in the CSV, the server downloads the image and performs processing (e.g., compressing the image).

Processed images are saved to the static/images directory.

Information about each processed image is saved to the ProductImage table in the database, including the original and processed image URLs.

The status of the processing request is updated to "Completed" or "Failed" based on the outcome.

* Status Check

Users can check the status of their request by providing the request ID to the /check\_status endpoint.

The server retrieves the status from the ProcessingRequest table and returns it.

* Generate Output CSV

Users can request to generate an output CSV file by providing the request ID to the /generate\_output\_csv endpoint.

The server compiles data from the ProductImage table and creates a CSV file listing the processed image URLs.

The generated CSV file path is returned, allowing the user to download the file.

4. Key Components

Backend

Flask Application: Handles HTTP requests and responses, processes the uploaded CSV, and interacts with the database.

Database Models: Defined in models.py, including ProcessingRequest for tracking processing requests and ProductImage for storing processed image details.

Image Processing: Uses aiohttp to fetch images asynchronously and Pillow for image manipulation.

Frontend

HTML/CSS: Provides a user interface for file upload, status checking, and result downloading.

JavaScript: Handles the asynchronous requests to the backend and updates the user interface based on the responses.

5. Error Handling

Validation Errors: Ensures that the CSV file has the required columns and no duplicate or empty values.

Processing Errors: Handles errors during image downloading or processing and updates the request status accordingly.

File Handling: Ensures that files are saved correctly and paths are managed properly.

6. Database Schema

ProcessingRequest: Tracks each processing request with its status and file path.

ProductImage: Stores details about each processed image, including original and processed URLs.

This system provides a complete workflow for processing image URLs from a CSV file, with functionalities for status checking and result generation.

**Setup Instructions**

1. **Clone the Repository**

git clone https://github.com/varun0122/Image\_Processing\_Project

cd Image\_Processing\_Project

1. **Create and Activate a Virtual Environment**

python -m venv venv

\venv\Scripts\activate

1. **Install Required Packages**

pip install flask sqlalchemy mysql-connector-python aiohttp pillow pandas

1. **Set Up the Database**
   * Open MySQL Workbench and execute the provided SQL script to create the necessary database and tables.
2. **Update Configuration**
   * Ensure that app.py has the correct database configuration.
3. **Run the Flask Application**

image-processing-system/

app.py

models.py

requirements.txt

templates/

index.html

uploads/

output\_csv/

* **app.py**: The main application file containing Flask routes and logic.
* **models.py**: Defines the database models.
* **requirements.txt**: Lists the Python dependencies.
* **templates/index.html**: HTML file for the user interface.
* **uploads/**: Directory for storing uploaded files and processed images.\

**Backend Code**

app.py

**Database Code**

Models.py

**Frontend Code**

templates/index.html

**API Endpoints**

**/submit\_file**

* **Method**: POST
* **Description**: Submits a CSV file for processing.
* **Request**: Form data containing the CSV file.
* **Response**: JSON with the request\_id.

**/check\_status**

* **Method**: GET
* **Description**: Checks the processing status of a request.
* **Query Parameters**: request\_id
* **Response**: JSON with request\_id and status.

**/generate\_output\_csv**

* **Method**: GET
* **Description**: Generates an output CSV file containing the results.
* **Query Parameters**: request\_id
* **Response**: JSON with output\_csv\_path.

**Database Schema**

**ProcessingRequest**

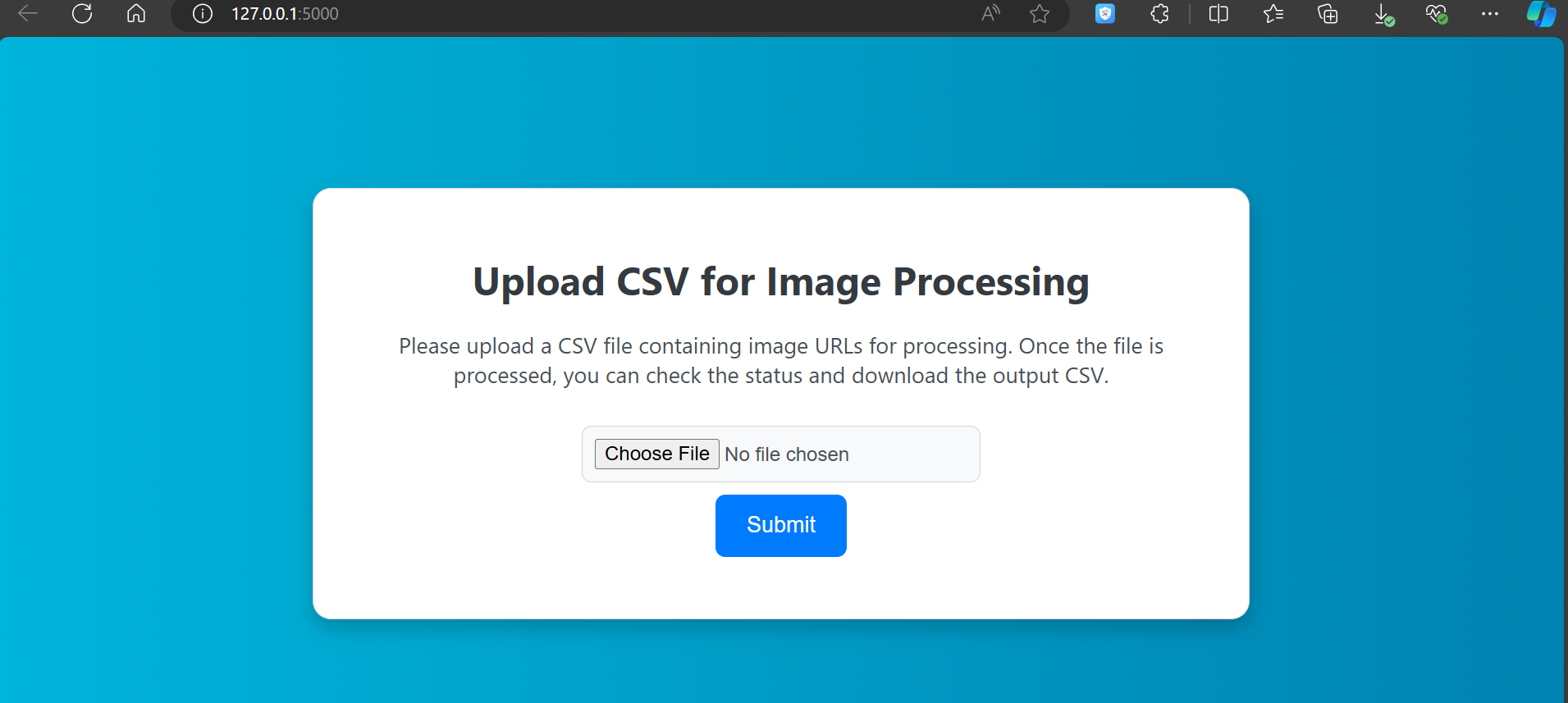
* **request\_id** (Primary Key): UUID
* **status**: String (Pending, In Progress, Completed, Failed)
* **csv\_file\_path**: String

**ProductImage**

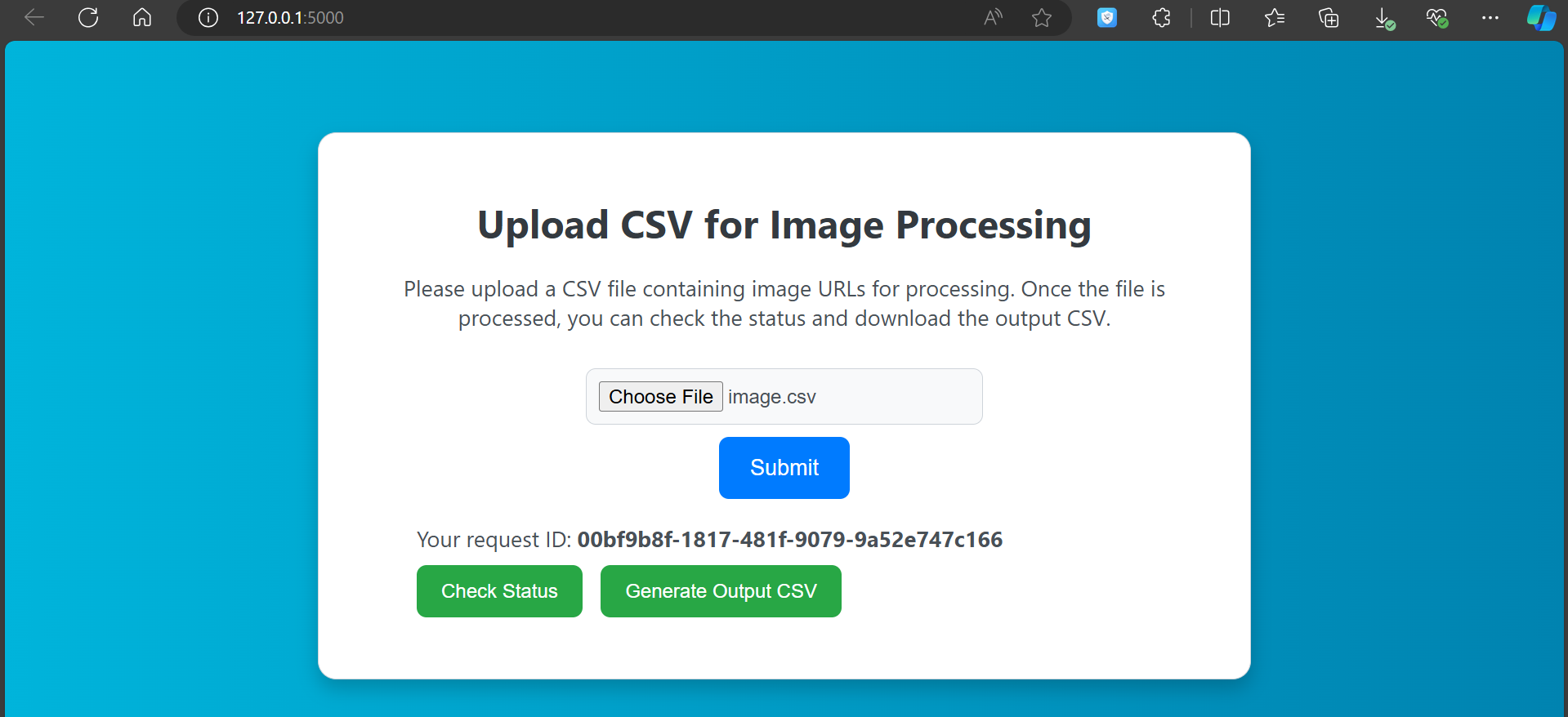
* **id** (Primary Key): Integer
* **serial\_number**: Integer
* **product\_name**: String
* **original\_url**: String
* **processed\_image\_path**: String
* **output\_url**: String
* **request\_id** (Foreign Key): UUID

**How to Use**

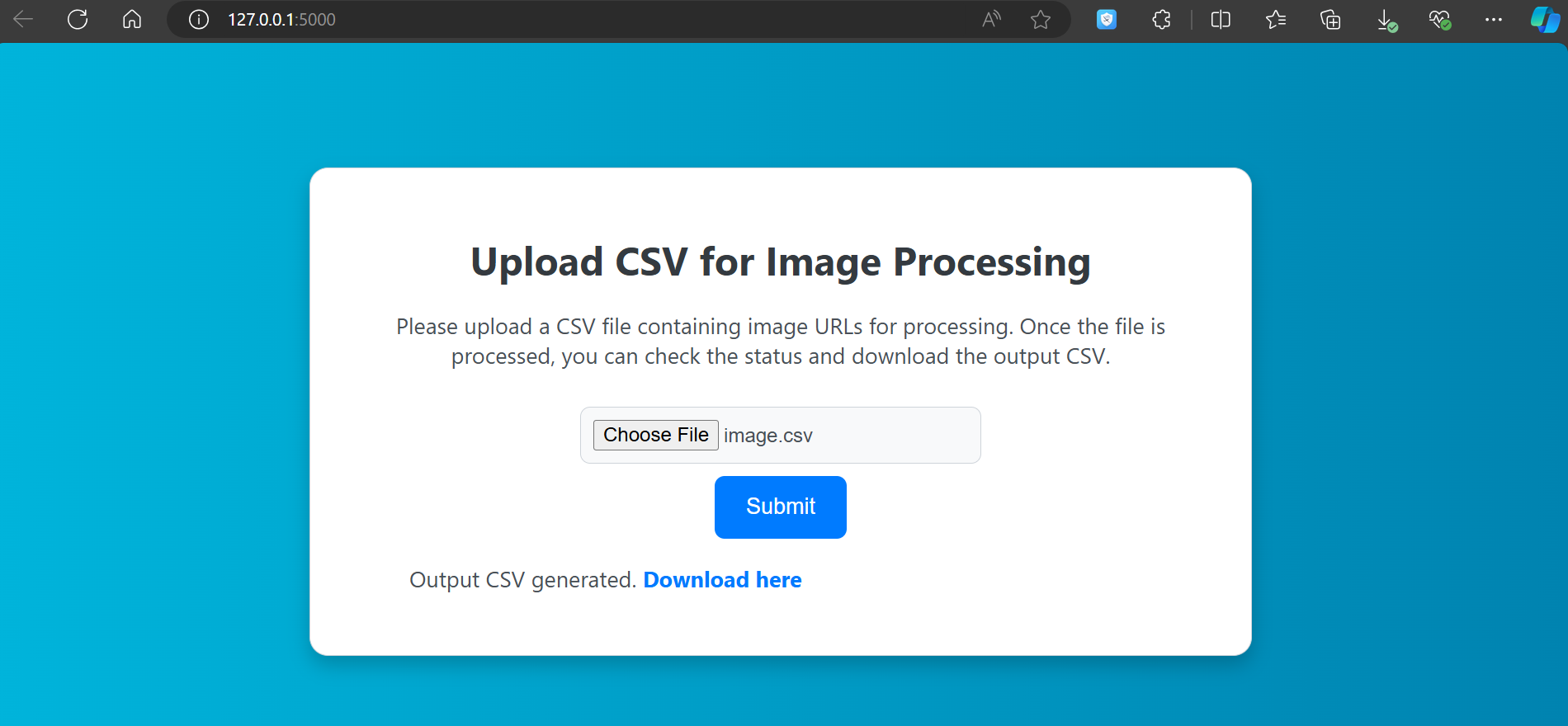
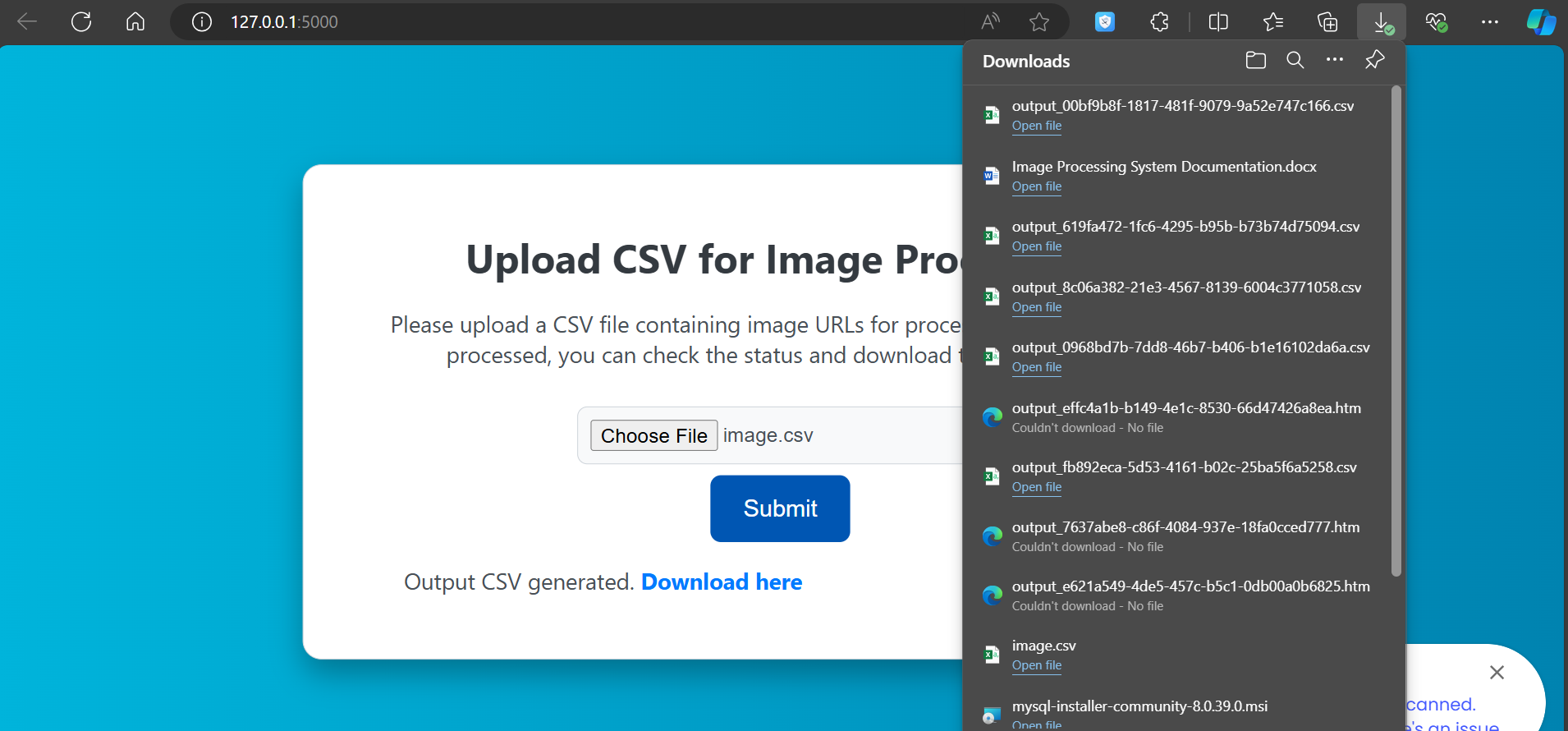
1. Open the application in a web browser.



1. Upload a CSV file containing image URLs.
2. Wait for the processing to complete.
3. Check the status of the request using the provided request ID.



1. Download the processed images or the output CSV from the status section.

**API Endpoints**

**1. Submit File**

* **Endpoint:** /submit\_file
* **Method:** POST
* **Description:** Upload a CSV file for processing.
* **Request:**
  + **Form Data:** file (CSV file)
* **Response:**
  + **JSON:** { "request\_id": "<unique\_request\_id>" }

**2. Check Status**

* **Endpoint:** /check\_status
* **Method:** GET
* **Description:** Check the status of a processing request.
* **Query Parameters:** request\_id (unique request ID)
* **Response:**
  + **JSON:** { "request\_id": "<request\_id>", "status": "<status>" }
  + Statuses: Pending, In Progress, Completed, Failed

**3. Generate Output CSV**

* **Endpoint:** /generate\_output\_csv
* **Method:** GET
* **Description:** Generate and download a CSV file with processed image URLs.
* **Query Parameters:** request\_id (unique request ID)
* **Response:**
  + **JSON:** { "output\_csv\_path": "/uploads/output\_<request\_id>.csv" }
  + The output CSV file contains columns: Serial Number, Product Name, Input Image Urls, Output Image Urls.

**4. Serve Files**

* **Endpoint:** /uploads/<path:filename>
* **Method:** GET
* **Description:** Serve files from the uploads directory.

**Path Parameter:** filename

**Troubleshooting**

* **404 Errors:** Ensure the file paths and URLs are correctly set.
* **FileNotFoundError:** Check the existence of required directories.
* **Database Errors:** Verify database configuration and connectivity.