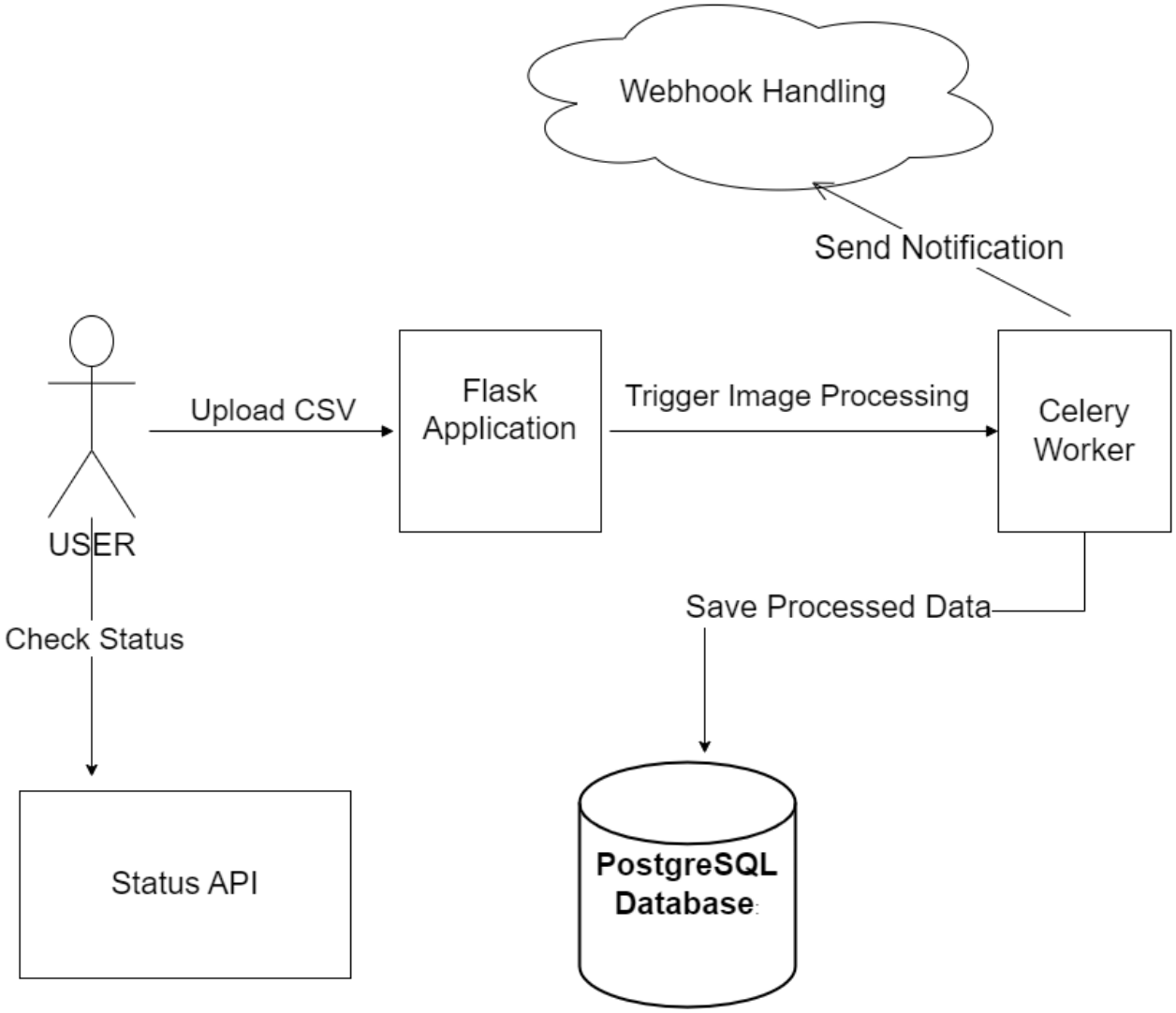
Low-Level Design Document

1. Introduction

- **Purpose**: This document provides a detailed technical design of the image processing system, including descriptions of the system components, data flow, and interactions.

2. System Overview

- # Architecture Diagram:



- **Components**:
  - Flask Application

- Celery Worker
- PostgreSQL Database
- Webhook Handling

3. Components

3.1 Flask Application

- **Role**: Manages API requests, validates input, triggers Celery tasks.
- **Key Files**:
  - app.py: Main application setup, including Flask, SQLAlchemy, Migrate, and Celery initialization.
  - upload_routes.py: Handles CSV file uploads, validates the CSV content, stores product info, and starts Celery tasks.
  - status_routes.py: Provides an API to check the status of Celery tasks using their task ID.
- **Data Flow**:
  - Receives the CSV file from the user.
  - Validates the file format and data.
  - Initiates Celery tasks for image processing.

3.2 Celery Worker

- **Role**: Asynchronously processes images by downloading, resizing, and saving them. Updates task status.
- **Key Files**:
  - image_tasks.py: Contains tasks for image processing, including downloading, resizing, compressing, and saving images.
- **Data Flow**:

    - Retrieves images from URLs provided in the CSV file.
    - Processes each image (resizing, compressing).
    - Saves processed image data to the database.

3.3 PostgreSQL Database

- **Role**: Stores product information, image URLs, and processing statuses.
- **Key Files**:
    - models.py: Defines database models including Product and Image.
- **Schema**:
    - **Tables**:
        - products: Stores information about products.
        - images: Stores processed image data and metadata.

3.4 Webhook Handling

- **Role**: Notifies external systems or users when image processing is complete.
- **Key Files**:
    - webhooks_routes.py: Handles webhook notifications and logs incoming data.
- **Data Flow**:
    - Triggers a webhook notification once image processing is complete.

4. Data Flow

1. **User Uploads CSV File**: User initiates the process by uploading a CSV file via the Upload API.

2. **Flask Application Validation**: The Flask app validates the CSV file format and content, then starts Celery tasks.
3. **Celery Worker Processing**: The Celery worker downloads, resizes, and processes the images as specified in the CSV file. It then stores the processed data in the PostgreSQL database.
4. **Status Query**: The user can check the processing status via the Status API.
5. **Webhook Notification**: Once processing is complete, a webhook notification is sent to inform the user or external systems.

5. Conclusion

- **Summary**: The design outlines a scalable and efficient system for asynchronous image processing, utilizing Flask for handling API requests, Celery for background processing, PostgreSQL for data storage, and webhooks for notifications.