Engineering RFC - Template

Authors

- Martin Li
- Ethan Ho
- David Brenes

Reviewers

- Suyash Kumar
- Mark Palmeri

Table of Contents

Abstract

This service will allow a user to upload an image or images to our web-server, preprocess the files, and send a RESTful API request to a cloud service that will process the image with options of histogram equalization, contrast stretching, log compression, and reverse video. The

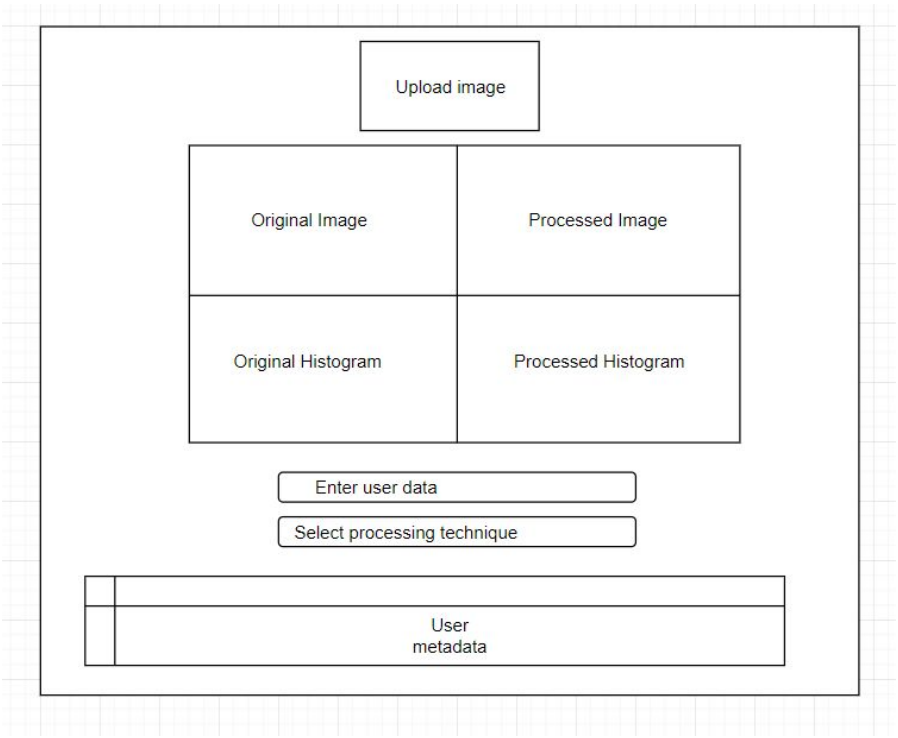
front-end will enable users to upload their images and see the results while a database will be implemented to store user actions and images.

Background

This project aims to process images using a database, web server, and a front end UI. The project aims to combine the skills we have learned in this class, using Python, React, and running a Flask Web server to deliver a complete product in which a user is able to upload an image or an archive of images to a web-server, perform image processing tasks on the web-server, and then display/download the processed images.

Designs [Mostly for frontend]

The front end hopes to be very simplistic, and easy to read and use. See the following design, where the textboxes are placeholders for functionality. This is a preliminary design with text as placeholders for actual functionality.



Architecture

The delegation of workflow will likely be:
Everyone will work on the original image processing methods
Ethan Ho will be in charge of frontend React development
Martin Li will be working on deployment of the webserver

David Brenes will be working on both React and the webserver (more webserver)

Plan:

- Class for the image (Models.py)
    - User has email as primary Key (Martin)
    - User has base64 string version of image (Martin)
    - User has process requested (Martin)
    - User has upload timestamp (Martin)
    - User has size of Image (Martin)
    - User has actual image (Reconstituted from base-64 string) (David)
    - User has histogram of corresponding images (Ethan)
- Originally main.py would handle all of the image processing and communication with the database but it was split between two functions, image_functions.py for image processing and user_funcitons.py to deal with database/user related processes
- Main.py
    - Decodes, finds histogram of, and pre-processes (if necessary) the image
    - Runs processing technique (from image_functions.py) and returns encoded processed image and histogram
    - Records processing duration
    - Saves everything to user
- Function for image processing (image_functions.py)
    - Hist. eq. (Ethan)
    - Contrast stretching (Martin)
    - Log compressions (David)
    - Reverse video (David)
    - These functions were added as we were working on the project:
        - Grayscale detection and conversion for pre-processing
        - Encode and decode image for storing in database and frontend-backend interface
        - Function to find size of images for user
        - Calculate histogram to create histogram of image and save a base64 representation of it
- Function for user data
    - Create user
    - Add user data if user is created
    - Return user metadata (for frontend)
    - Save filename and base64 conversion of image
    - Save histograms of corresponding images
    - Find processing duration time
- API calls (Basic.py)
    - Connects main.py to frontend

Diagram

Endpoints

Endpoints:

- API POST requests
    - Image Upload to Database and Preprocessing
    - Image Processing Method
    - Obtain metadata for a specific user
- API GET requests
    - Show processed images
    - Download processed images

Database

A database should be implemented in some form to do one or more of the following:

- Store previous user actions / metrics (e.g. how many times has a user run Histogram Equalization, latency for running different processing tasks, etc).
- Store uploaded images and timestamps for a user

    - The database will likely have an email as a user key as well as an image path, and the image will live on the hard drive of where the server is running.
    - React sends an encoded base64 representation of the image which the backend decodes and processes. This base64 encoding is what is stored in the database
- Store processed images (along with what processing was applied) and timestamps for a user
    - Also a base64 encoded version

Infrastructure

1. Where will instances of this feature/service run (within Hospital networks, cloud, etc).
    a. The service will run on a cloud server, likely hosted by the Duke VMs
2. Does this feature/service have specific infrastructure requirements?
    a. This service will not have any specific infrastructure requirements

Security Considerations-

1. Does this service handle and/or persist PHI? If so, what precautions will be taken to safeguard this data?
    a. No
2. Does this service rely on secrets? How are those secrets distributed/safeguarded?
    a. No

Failure Modes & Mitigation

1. What are the service dependencies?
2. What problems occur if those downstream services are malfunctioning (high latencies) or down altogether?
3. What other failure modes exist for this feature/service, and what impact do those have? How can they be mitigated?

Not applicable.

Alerting & Monitoring

1. What logging and monitoring will be in place for this feature/service
2. What alerts make sense for this service (if any) and how will those alerts be configured and set?

Not applicable.