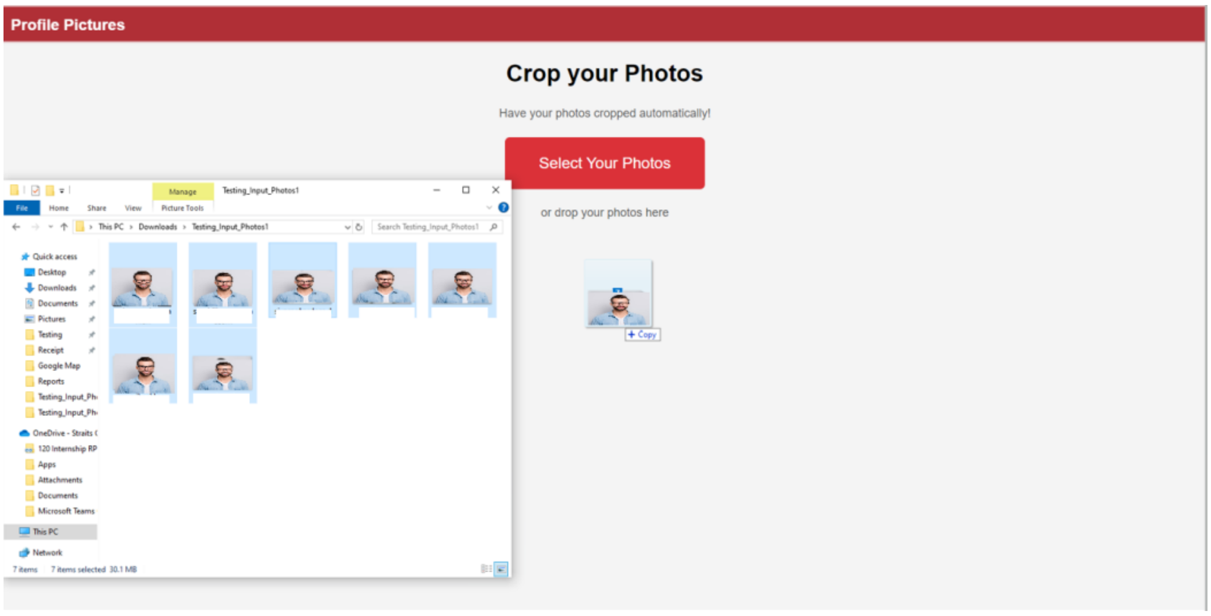
Profile Pictures Editor.EXE (Flask app)

Purpose:

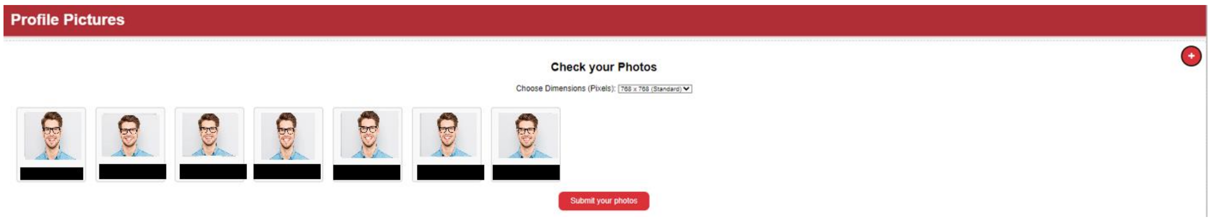
To do easy cropping for employee profile photos.

Features:

- Drag and Drop photo function & Selecting it

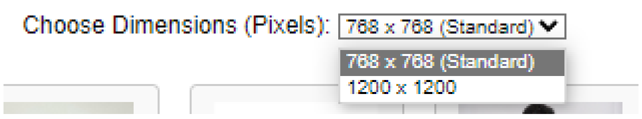


- Add more photos inside by pressing the plus button or dragging and dropping into the white box



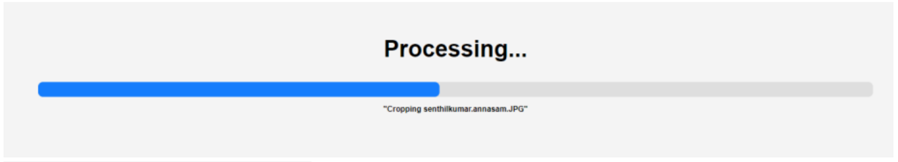
Choose Dimensions (Pixels): 768 x 768 (Standard)

- Able to choose the dimensions of the photo you need



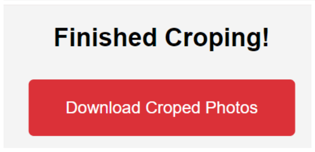
Choose Dimensions (Pixels): 768 x 768 (Standard)

- Process Bar to tell you when it is done and you can download the photos



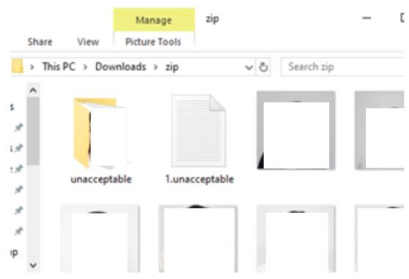
Processing...

"Cropping senthilkumar.annasam.JPG"



Finished Croping!

Output:



The photos that can not be cropped properly will be placed into the unacceptable folder and the photo's name, pixel size will be in the 1.unacceptable txt file, while the others that are accepted will be not in the unacceptable folder and txt file.

The process:

1. Create temporary directories for each process that the photos will be going through

main.py (Line 20)

```
def create_temp_dirs(): #create temporary directory
    global UPLOAD_FOLDER, WRONG_FOLDER, OUTPUT_FOLDER, ZIP_FOLDER
    UPLOAD_FOLDER = tempfile.TemporaryDirectory()
    OUTPUT_FOLDER = tempfile.TemporaryDirectory()
    WRONG_FOLDER = os.path.join(OUTPUT_FOLDER.name, "unacceptable")
    ZIP_FOLDER = tempfile.TemporaryDirectory()
    os.makedirs(WRONG_FOLDER)
```

2. Once the user puts in the photos, it will fetch the app.route '/upload' and then get the files from the website that is dropped in and save it in the temporary directory called UPLOAD_FOLDER (same applies to the reorder.html when the user wants to add more photos to the existing photos)

index.html (Line 47)

```
function handleFiles(files) {
            const formData = new FormData();
            for (const file of files) {
                formData.append('files', file);
            }

            fetch('/upload', {
                method: 'POST',
                body: formData,
            })
            .then(response => response.json())
            .then(data => {
                console.log(data);
                alert('Files uploaded successfully');
                window.location.href = '/reorder';
            })
            .catch(error => {
                console.error(error);
                alert('Error uploading files');
            });
        }
```

main.py (Line 173)

To get the files from the website and store it in UPLOAD_FOLDER

```
@app.route('/upload', methods=['POST'])
def upload_files():
    if 'files' not in request.files:
        return jsonify({'error': 'No files part'}), 400

    files = request.files.getlist('files')
    for file in files:
        if file:
            filename = file.filename
            file.save(os.path.join(UPLOAD_FOLDER.name, filename))

    return jsonify({'message': 'Files uploaded successfully'}), 200
```

3. Brings to /reorder and the user will choose the dimension. Once dimension is chosen the value from that option will be sent to the main.py

reorder.html (Line 78)

```
const dimensionsSelect = document.getElementById('dimensions');
            const selectedDimensions = dimensionsSelect.value;

            fetch('/submit', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({
                    files: reorderedFiles,
                    dimensions: selectedDimensions
                })
```

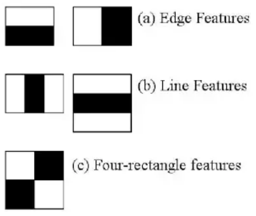
app.py (Line 196)

It will take the selected dimensions and be passed over to cropfaces which is a function that will be activated. That is when the cropping starts.

```
@app.route('/submit', methods=['POST'])
def save_order():
    global selected_dimensions

    data = request.json
    files = data.get('files')
    selected_dimensions = data.get('dimensions')
    Thread(target=cropfaces).start()
    return jsonify({'message': 'Order saved successfully'}), 200
```

4. In cropfaces, it retrieves a face detection algorithm to detect the face of the person which is fast as it uses rectangular features that are similar to kernel that uses to detect different features of the face like eyes for example:



It uses this to calculate the difference in intensity between the eye regions and cheeks regions of the face and so on.
For more information click here: Face Detection with HAAR Cascade in OpenCV Python - MLK - Machine Learning Knowledge

main.py (Line 50)

```
cascade_file_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
'haarcascade_frontalface_alt2.xml')
    face_cascade = cv2.CascadeClassifier(cascade_file_path)
```

main.py (Line 70)

```
for i, file in enumerate(files):
        message = f'Cropping {file}'
        filepath = os.path.join(uploadfolder, file)
        img = cv2.imread(filepath)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #turn to gray easier to
detect
        faces = face_cascade.detectMultiScale(gray, 1.1, 4) #to detect face
```

main.py (Line 75 - 128)

```
for (x, y, w, h) in faces:
            img_height, img_width, _ = img.shape

            size_face_height = (y + h) - y

            if img_height > img_width:
                size = img_width
            else:
                size = img_height

            gaps = size - size_face_height
            dividegaps = int(gaps / 2)

            startx = x - dividegaps
            endx = x + w + dividegaps

            if startx < 0:
                startx = 0

            starty = y - dividegaps
            endy = y + h + dividegaps

            if starty < 0:
                starty = 0
            elif starty > 100:
                starty = y - dividegaps - 100
                endy = y + h + dividegaps + 100

            if endx > img_width:
                endx = img_width
            if endy > img_height:
                endy = img_height

            if (endx - startx) > (endy - starty):
                gap = endx - startx -  endy
                dividegaps = int(gap / 2)
                if starty >= dividegaps:
                    endy += dividegaps
                    starty -= dividegaps
                else:
                    starty = 0
                    endy += gap

            elif (endx - startx) < (endy - starty):
                gap = endy - starty - endx
                dividegaps = int(gap / 2)
                if startx >= dividegaps:
                    endx += gap
                    startx -= gap
                else:
                    startx = 0
                    endx += gap

            face = img[starty:endy, startx:endx]
```

After detecting the face, it will tell us the position of it giving the x, y axis, width and height. Using that information, it is needed to have gaps so that it can get the shoulder and the hair.

5. Then, take the crop photo and get the height and width and check if both of them are the same length so that when it is being resized to a certain pixel which is a square it will not be stretched

main.py (Line 78)

Original photo size

```
img_height, img_width, _ = img.shape
```

main.py (Line 134)

```
face_height, face_width, _ = face.shape
if face_width != face_height:
                output += f'{file}\nFace size: {face_width} x {face_height}
pixels\n'
                cv2.imwrite(os.path.join(WRONG_FOLDER, file), img)
            else:
                face = cv2.resize(face, (width, height))
                cv2.imwrite(os.path.join(OUTPUT_FOLDER.name, file), face)
```

6. While this is processing there will be a progress bar that is slowly increasing.

main.py (Line 66)

```
each_file_progress = 1/total_files * 100
```

main.py (Line 151)

```
progress += int(each_file_progress)
        time.sleep(0.05)
```

main.py (Line 205)

It will take the progress from the cropfaces function and then be sent over to reorder.html and the progress bar will increase by the percentage from the each_file_progress

```
@app.route('/progress', methods=['GET']) #show progress when cropping and
zipping
def check_progress():
    global progress
    return json.dumps({'progress': progress, 'message': message})
```

reorder.html (Line 109)

It will show the progress bar and the messages showing what the system is doing right now

```
function checkProgress() { // show the progress bar and what it is doing
            fetch('/progress', {method:'GET'})
                .then(response => response.json())
                .then(data => {
                    progressMessage.innerHTML = JSON.stringify(data.message)
                    progressMessage.style.fontSize = '1rem';
                    progressBarFill.style.width = data.progress + '%';
                    if (data.progress < 100) {
                        setTimeout(checkProgress, 500);
                    } else {
                        progressContainer.style.display = 'none';
                        downloadButton.style.display = 'block';
                        downloadsay.style.display = 'block';
                    }
                })
                .catch(error => {
                    console.error(error);
                });
        }
```

7. After that is done it will zip up the file which is a temporary directory.

main.py (Line 158)

```
zip_filepath = os.path.join(ZIP_FOLDER.name, 'download.zip')

    with zipfile.ZipFile(zip_filepath, 'w', zipfile.ZIP_DEFLATED) as zip_file:
        for root, dirs, files in os.walk(OUTPUT_FOLDER.name):
            for file in files:
                filepath = os.path.join(root, file)
                zip_file.write(filepath, os.path.relpath(filepath,
OUTPUT_FOLDER.name))
```

8. When user press the button to download, it will grab the download app.route and the file will be sent to the reorder.html to download

reorder.html (Line 170)

```
function downloadphotos() {
            fetch('/download', {
                method: 'POST'
            })
            .then(response => response.blob())
            .then(blob => {
                const url = window.URL.createObjectURL(blob);
                const a = document.createElement('a');
                a.href = url;
                a.download = '{{ filename }}.zip';
                a.click();
                window.URL.revokeObjectURL(url);
                window.location.href = '/'
            })
            .catch(error => {
                console.error(error);
                alert('Error downloading');
            });
        }
```

app.py (Line 210)

```
@app.route('/download', methods=['POST']) #to download the zip file
def download():
    global zip_filepath
    global filename

    _, extension = os.path.splitext(zip_filepath)
    custom_filename = f"{filename}{extension}"

    return send_file(zip_filepath, as_attachment=True,
download_name=custom_filename)
```

Once done make a .EXE application of this using pyinstaller

```
pyinstaller --onefile --add-data "haarcascade_frontalface_alt2.xml;." --add-data "static;static" --add-data "templates;templates" main.py
```

Since this handle confidential files, it will be better for it to be local application instead of putting it on the website as some photos may leak if the code it does not secure enough. In addition to the security and to keep people's confidentiality, I have also cleared the file once the user has finished using and before using.